Knowledge-Based Programs*

Ronald Fagin†
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120–6099
fagin@almaden.ibm.com

Joseph Y. Halpern‡
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120–6099
halpern@almaden.ibm.com

Yoram Moses§
Department of Applied Math. and CS
The Weizmann Institute of Science
76100 Rehovot, Israel
yoram@wisdom.weizmann.ac.il

Moshe Y. Vardi¶
Department of Computer Science
Rice University
Houston, TX 77005-1892
vardi@cs.rice.edu

December 11, 1996

Abstract

Reasoning about activities in a distributed computer system at the level of the *knowledge* of individuals and groups allows us to abstract away from many concrete details of the system we are considering. In this paper, we make use of two notions introduced in our recent book to facilitate designing and reasoning about systems in terms of knowledge. The first notion is that of a *knowledge-based program*. A knowledge-based program is a syntactic object: a program with tests for knowledge. The second notion is that of a *context*, which captures the setting in which a program is to be executed. In a given context, a standard program (one without tests for knowledge) is *represented by* (i.e., corresponds in a precise sense to) a unique system. A knowledge-based program, on the other hand, may be represented by no system, one system, or many systems. In this paper, we provide a sufficient condition for a knowledge-based program to be represented in a unique way in a given context. This condition applies to many cases of interest, and covers many of the knowledge-based programs considered in the literature. We also completely characterize the complexity of determining whether a given knowledge-based program has a unique representation, or any representation at all, in a given *finite-state* context.

*A preliminary version of this paper appear in *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*, 1995, pp. 153–163. This version is essentially identical to one that will appear in *Distributed Computing*.

†URL: http://www.almaden.ibm.com/cs/people/fagin/

‡Work supported in part by the Air Force Office of Scientific Research (AFSC), under Contract F49620-91-C-0080. Current address: Cornell University, Department of Computer Science, Ithaca, NY 14853. URL: http://www.cs.cornell.edu/home/halpern.

§Part of this research was performed while this author was on sabbatical at Oxford. His work is supported in part by a Helen and Milton A. Kimmelman career development chair. URL: http://www.wisdom.weizmann.ac.il/~yoram

¶Part of this research was done while this author was at the IBM Almaden Research Center. URL: http://www.cs.rice.edu/~vardi

1 Introduction

Reasoning about activities in a distributed computer system at the level of the knowledge of individuals and groups allows us to abstract away from many concrete details of the system we are considering. One approach to program development is to work top-down, first designing a high-level protocol, and then implementing the high-level constructs in a way that may depend on properties of the particular setting at hand. This style of program development will generally allow us to modify the program more easily when considering a setting with different properties, such as a different communication topology, different guarantees about the reliability of various components of the system, etc.

Motivated by these considerations, Halpern and Fagin [HF85] suggested a notion of *knowledge-based protocols*, in which an agent's actions depend explicitly on the agent's knowledge. Their goal was to provide a formal semantics for programs with tests for knowledge such as

$$\textbf{if } K(x=0) \textbf{ do } y := y+1,$$

where $K(x=0)$ should be read as "you know that $x=0$". Unfortunately, the technical definition of knowledge-based protocols given in [HF85] (later simplified somewhat in [HF89, Maz91, NT93]), had a number of deficiencies, which made it somewhat difficult to use as a tool for program design. For one thing, a knowledge-based protocol was defined as a function from local states and systems to actions (we provide details in Section 3.2). Thus, the definition did not directly capture the intuition that knowledge-based programs were meant to be programs with tests for knowledge. Moreover, the approach did not provide a clean distinction between the protocol (or program) and the setting in which it is to be executed. As a result, the high-level and model-independent reasoning we wish to use knowledge for was not facilitated by the definition as much as it perhaps could have been. Nevertheless, knowledge-based protocols were used (either formally or informally) in papers such as [DM90, HZ92, MT88, NB92, HMW90].

In [FHMV95], an approach is given that overcomes the deficiencies of the earlier definition. This approach introduces *knowledge-based programs*, which are what knowledge-based protocols were intended to be: (syntactic) programs with tests for knowledge. This approach includes the introduction of *contexts*, which capture the setting in which a program is to be executed. By distinguishing between programs and contexts, and by ascribing meaning to programs in different contexts in a uniform manner, high-level and model-independent reasoning based on knowledge are facilitated.

In a given context, we can associate with every *protocol*—a mapping from local states to actions—a unique system, namely, the system consisting of all possible runs (or executions) of the protocol in that context. We think of this as the system that *represents* the protocol in this context. We similarly want to associate with each knowledge-based program a system that characterizes it in a given context, but there are subtleties. In general, such a system is not guaranteed to exist, and if one exists, it is not guaranteed to be unique. A knowledge-based program should be viewed as a high-level specification; the systems that represent it can be viewed as those systems that satisfy the specification. If there are no systems representing the program, then the specification is inconsistent (at least, in the given context); if there is more than one, that simply means that the specification can be satisfied in more than one way.

Of course, if we are to program at the knowledge level, it is surely useful to be able to tell whether there is a system representing a given knowledge-based program, and if so, whether this system is unique. In instances in which we can determine by the syntactic structure of a knowledge-based program that it has exactly one representation, we can think of the program as a high-level description of a specific behavior of the agents. In such cases, we are justified in thinking of a knowledge-based program simply as a program with tests for knowledge. In this paper, we provide a condition that is sufficient to guarantee that a knowledge-based program is represented by a unique system, and covers many of the simple knowledge-based programs considered in the literature. This condition is somewhat similar in spirit to a condition considered in [HF89] that guarantees the existence of a canonical system corresponding to a knowledge-based protocol. We also completely characterize the complexity of determining whether a given knowledge-based program has a unique representation, or any representation at all, in a given *finite-state* context.

The rest of this paper is organized as follows. The next two sections review material from [FHMV95]: In Section 2, we describe the multi-agent systems framework, and in Section 3, we discuss standard and knowledge-based programs. We present our sufficient condition for the existence of unique representations in Section 4, and examine the complexity of checking whether a knowledge-based program is represented by a unique system, or any system at all, in Section 5.

2 The multi-agent systems framework

We want to be able to view any collection of interacting agents as a multi-agent system. Agents playing a game, processes running a protocol, and interacting robots are all examples of multi-agent systems. Thus, we need a framework that is general enough to allow all of these as special cases. Such a framework was introduced in [HM90, HF85], and further developed in [FHMV95]. We review the details here.

2.1 Runs and systems

We assume that at any point in time, each of the agents in the system is in some *state*. We refer to this as the agent's *local* state, in order to distinguish it from a *global* state, which we define shortly. We assume that an agent's local state encapsulates all the information to which the agent has access. In this abstract framework, we do not make any additional assumptions about the state. If we are modeling a poker game, a player's state might consist of the cards he currently holds, the bets made by the other players, any other cards he has seen, and any information he may have about the strategies of the other players (for example, Bob may know that Alice likes to bluff, but that Charlie tends to bet conservatively). If we are modeling a distributed system, a process's local state might consist of the values of certain variables and a list of messages received and sent.

In addition to the agents, it is useful to have an *environment*, which we can think of as capturing everything else that is relevant to the analysis that is not in the agents' local states. In many ways the environment can be viewed as just another agent, though it typically plays a special role in many analyses. Like the agents, the environment has a local state. If we are analyzing a message-passing system where processes send messages back and forth along communication lines, we might have the environment's local state keep track of the messages that are in transit, and whether a communication line is up or down. A *global state* of a system with n agents is an $(n+1)$-tuple of the form $(\ell_e, \ell_1, \ldots, \ell_n)$, where ℓ_e is the local state of the environment and ℓ_i is the local state of agent i.

A global state describes the system at a given point in time. But a system is not a static entity; it constantly changes. Since we are mainly interested in how systems change over time, time must be built into the model. A *run* is a function from time to global states. Intuitively, a run is a complete description of how the system's global state evolves over time. Time ranges over the natural numbers. Thus, $r(0)$ describes the initial global state of the system in a possible execution r, the next global state is $r(1)$, and so on. The run r can be viewed as the sequence $r(0), r(1), \ldots$ of the global states that the system goes through. Time is measured on some clock external to the system. We do *not* assume that agents in the system necessarily have access to this clock; at time m measured on the external clock, agent i need not know it is time m. If an agent does know the time, then this information is encoded in his local state (we return to this issue later). This external clock need not measure "real time".

A system can have many possible runs, since the system's global state can evolve in many possible ways: there are a number of possible initial states and many things that could happen from each initial global state. For example, in a poker game, the initial global states could describe the possible deals of the hand, with player i's local state ℓ_i describing the cards held initially by player i. For each fixed deal of the cards, there may still be many possible betting (and discarding) sequences, and thus many runs. In a message-passing system, a particular message may or may not be lost, so again, even with a fixed initial global state, there are many possible runs. (A formal definition of runs is given in the next paragraph.) To capture this, a *system* is formally defined to be a *nonempty set of runs*. This definition abstracts the intuitive view of a system as a collection of interacting agents. Instead of trying to model the system directly, this definition models the possible *behaviors* of the system. The requirement that the set of runs be nonempty captures the intuition that the system being modeled has *some* behaviors.

We summarize this discussion as follows:

Definition 2.1 Let L_e be a set of possible states for the environment and let L_i be a set of possible local states for agent i, for $i = 1, \ldots, n$. The set of *global states* is $\mathcal{G} = L_e \times L_1 \times \cdots \times L_n$. A *run over* $\mathcal{G}$ is a function from the time domain—the natural numbers in our case—to $\mathcal{G}$. Thus, a run over $\mathcal{G}$ can be identified with a sequence of global states in $\mathcal{G}$. A pair (r, m) consisting of a run r and time m is called a *point*. If $r(m) = (\ell_e, \ell_1, \ldots, \ell_n)$ is the global state at the point (r, m), define $r_e(m) = \ell_e$ and $r_i(m) = \ell_i$, for $i = 1, \ldots, n$; thus, $r_i(m)$ is agent i's local state at the point (r, m). A *round* takes place between two time points. Round m in run r is defined to take place between time $m-1$ and time m. A *system* $\mathcal{R}$ *over* $\mathcal{G}$ is a set of runs over $\mathcal{G}$. We say that (r, m) is a *point in system* $\mathcal{R}$ if $r \in \mathcal{R}$. ∎

2.2 Actions

In our discussion of runs, we avoided consideration of where the runs came from. Starting in some initial global state, what causes the system to change state? Intuitively, it is clear that this change occurs as a result of *actions* performed by the agents and the environment. It is often convenient for us to view these actions as being performed during a round. Neiger [Nei88] explicitly includes actions in his model of a run; for simplicity, we do not. However, we can easily model actions as part of the state: If an agent knows his actions, then they can be part of the agent's local state. Otherwise, they can be included in the environment's state.

For us, actions are simply elements of some specific set. Thus, we assume that for each agent i there is a set ACT_i of actions that can be performed by i. For example, in a distributed system, an action $\mathsf{send}(x, j, i)$—intuitively, this action corresponds to i sending j the value of variable x—might be in ACT_i if x is a variable that is local to agent i. On the other hand, if x is not a local variable of i, then it would usually be inappropriate to include $\mathsf{send}(x, j, i)$ in ACT_i. In keeping with the policy of viewing the environment as an agent (albeit one whose state of knowledge is not of interest), the environment is allowed to perform actions from a set ACT_e. In message-passing systems, it is perhaps best to view message delivery as an action performed by the environment. If we consider a system of sensors observing a terrain, we may want to view a thunderstorm as an action performed by the environment. For both the agents and the environment, we allow for the possibility of a special *null* action Λ which corresponds to the agents or the environment performing no action.

Knowing which action was performed by a particular agent is typically not enough to determine how the global state of the system changes. Actions performed simultaneously by different agents in a system may interact. If two agents simultaneously pull on opposite sides of a door, the outcome may not be easily computed as a function of the outcomes of the individual actions when performed in isolation. If two processes try simultaneously to write a value into a register, it is again not clear what will happen. To deal with potential interaction between actions, we consider *joint actions*. A joint action is a tuple of the form $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)$, where a_e is an action performed by the environment and a_i is an action performed by agent i, for $i = 1, \ldots, n$.

How do joint actions cause the system to change state? We would like to associate with each joint action $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)$ a *global state transformer* $\mathcal{T}$, where a global state transformer is simply a function mapping global states to global states, i.e., $\mathcal{T} : \mathcal{G} \to \mathcal{G}$. Joint actions cause the system to change state via the associated global state transformers; if the system is in global state g when the action $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)$ is being performed, then the system changes its state to $\mathcal{T}(g)$. Thus, whenever we discuss actions we will also have a mapping τ that associates with each joint action $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)$, a global state transformer $\tau(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)$. The mapping τ is called the *transition function*. Note that the definition requires that $\tau(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)(\ell_e, \ell_1, \ldots, \ell_n)$ be defined for each joint action $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)$ and each global state $(\ell_e, \ell_1, \ldots, \ell_n)$. In practice, not all joint actions and all global states are going to be of interest when we analyze a multi-agent system, since certain combinations of actions or certain combinations of local states will never actually arise. In such cases, we can let $\tau(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)(\ell_e, \ell_1, \ldots, \ell_n)$ be defined arbitrarily. Typically, we define $\tau(\Lambda, \ldots, \Lambda)$ to be the *no-op* transformer ι, where $\iota(\ell_e, \ell_1, \ldots, \ell_n) = (\ell_e, \ell_1, \ldots, \ell_n)$. We make this assumption in this paper.

To summarize, we have the following definitions:

Definition 2.2 A *joint action* is an element of the set $ACT_e \times ACT_1 \times \cdots \times ACT_n$. Given a set $\mathcal{G}$ of global states, a *global state transformer* is a mapping from $\mathcal{G}$ to $\mathcal{G}$. A *transition function* maps joint actions to global state transformers. ∎

2.3 Protocols

Intuitively, a *protocol* for agent i is a description of what actions agent i may take, as a function of her local state. A *protocol* P_i for agent i is formally defined to be a function from the set L_i of agent i's local states to nonempty sets of actions in ACT_i. The fact that we consider a set of possible actions allows us to capture nondeterministic protocols. Of course, at a given step of the protocol, only one of these actions is actually performed; the choice of action is nondeterministic. A *deterministic* protocol is one that maps states to actions, i.e., it prescribes a unique action for each local state. Formally, P_i is deterministic if $P_i(\ell_i)$ is a singleton set for each local state $\ell_i \in L_i$. We remark that if P_i is deterministic, we typically write $P_i(\ell_i) = \mathsf{a}$ rather than $P_i(\ell_i) = \{\mathsf{a}\}$. If we had wanted to consider probabilistic protocols (which we do not here, since it would only complicate the exposition), we would need to put a probability distribution on the set of actions that an agent can perform at a given state. This would then generate a probability space on the set of possible runs of the protocol.

Just as it is useful to view the environment as performing an action, it is also useful to view the environment as running a protocol. A protocol for the environment is defined to be a function from L_e to nonempty subsets of ACT_e. For example, in a message-passing system, we can use the environment's protocol to capture the possibility that messages are lost or that messages may be delivered out of order. If all the agents and the environment follow deterministic protocols, then there is only one run of the protocol for each initial global state. In many examples, the agents follow deterministic protocols, but the environment does not.

While this notion of protocol is quite general, there is a crucial restriction: a protocol is a function on *local* states, rather than a function on *global* states. This captures the intuition that all the information that the agent has is encoded in his local state. Thus, what an agent does can depend only on his local state, and not on the whole global state. This definition of protocol is so general that it allows protocols that are arbitrary functions on local states, including ones that cannot be computed. Of course, in practice we are typically interested in *computable* protocols, i.e., protocols for which there is an algorithm that takes a local state as input and returns the set of actions prescribed by the protocol in that state.

Processes do not run their protocols in isolation; it is the combination of the protocols run by all agents that cause the system to behave in a particular way. A *joint protocol* P is a tuple $(P_1, \ldots, P_n)$ consisting of protocols P_i, for each of the agents $i = 1, \ldots, n$. Note that while the environment's action is included in a joint action, the environment's protocol is not included in a joint protocol. This is because of the environment's special role; we usually design and analyze the agents' protocols, while taking the environment's protocol as a given. In fact, when designing multi-agent systems, the environment is often seen as an *adversary* who may be trying to cause the system to behave in some undesirable way. In other words, the joint protocol P and the environment protocol P_e can be viewed as the strategies of opposing players.

In summary:

Definition 2.3 A protocol P_i for agent i is a mapping from the set L_i of agent i's local states to nonempty sets of actions in ACT_i. A protocol P_e for the environment is a mapping from the set L_e of the environment's local states to nonempty sets of actions in ACT_e. ∎

2.4 Contexts

A joint protocol P and an environment protocol prescribe the behavior of all "participants" in the system and therefore, intuitively, should determine the complete behavior of the system. On closer inspection, the protocols describe only the actions taken by the agents and the environment. To determine the behavior of the system, we also need to know the "context" in which the joint protocol is executed. What does such a context consist of? Clearly, the environment's protocol P_e should be part of the context, since it determines the environment's contribution to the joint actions. In addition, the context should include the transition function τ, since it is τ that describes the results of the joint actions. Furthermore, the context should contain the set $\mathcal{G}_0$ of *initial* global states, since this describes the state of the system when execution of the protocol is initiated. In general, not all global states are possible initial states. These components of the context provide us with a way of describing the environment's behavior at any single step of an execution.

There are times when we wish to consider more global constraints on the environment's behavior, ones that are not easily captured by P_e, τ, and $\mathcal{G}_0$. This is the case, for example, with a *fairness* assumption such as "all message sent are eventually delivered". There are a number of ways that we could capture such a restriction on the environment's behavior. Perhaps the simplest is to specify an *admissibility* condition Ψ on runs, that tells us which ones are "acceptable". Formally, Ψ is a set of runs; $r \in \Psi$ iff r satisfies the condition Ψ. Notice that while the environment's protocol can be thought of as describing a restriction on the environment's behavior at any given point in time, the reliable delivery of messages is a restriction on the environment's behavior throughout the run, or, in other words, on the acceptable infinite behaviors of the environment. Indeed, often the admissibility condition Ψ can be characterized by a formula in temporal logic, and the runs in Ψ are those that satisfy this formula. We return to this point when we review the formal definitions of temporal logic in the next section. The condition consisting of all runs is denoted by *True*; this is the appropriate condition to use if we view all runs as "good".

Definition 2.4 A *context* γ is a tuple $(P_e, \mathcal{G}_0, \tau, \Psi)$, where $P_e : L_e \rightarrow 2^{ACT_e} - \{\emptyset\}$ is a protocol for the environment, $\mathcal{G}_0$ is a nonempty subset of the set $\mathcal{G}$ of global states, τ is a transition function, and Ψ is an admissibility condition on runs. ∎

Notice that by including τ in the context, we are also implicitly including the sets $L_e, L_1, \ldots, L_n$ of local states as well as the sets $ACT_e, ACT_1, \ldots, ACT_n$ of actions, since the set of joint actions is the domain of τ and the set of global states is the domain of the transition functions yielded by τ. To minimize notation, we do not explicitly mention the state sets and action sets in the context. We shall, however, refer to these sets and to the set $\mathcal{G} = L_e \times L_1 \times \cdots \times L_n$ of global states as if they were part of the context.

It is only in a context that a joint protocol describes the behavior of a system. As we shall see later on, the combination of a context γ and a joint protocol P for the agents uniquely determines a set of runs, which we shall think of as the system representing the execution of the joint protocol P in the context γ.

2.5 Consistency

We can now talk about the runs of the protocol in a given context.

Definition 2.5 A run r is *weakly consistent with a joint protocol* $P = (P_1, \ldots, P_n)$ *in context* $\gamma = (P_e, \mathcal{G}_0, \tau, \Psi)$ if

1. $r(0) \in \mathcal{G}_0$ (so $r(0)$ is a legal initial state),
2. for all $m \geq 0$, if $r(m) = (\ell_e, \ell_1, \ldots, \ell_n)$, then there is a joint action $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n) \in P_e(\ell_e) \times P_1(\ell_1) \times \cdots \times P_n(\ell_n)$ such that $r(m+1) = \tau(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)(r(m))$ (so $r(m+1)$ is the result of transforming $r(m)$ by a joint action that could have been performed from $r(m)$ according to P and P_e).

The run r is *consistent* with P in context γ if it satisfies in addition

3. $r \in \Psi$ (so that, intuitively, r is admissible according to Ψ). ∎

Thus, the run r is consistent with P in context γ if r is a possible behavior of the system under the actions prescribed by P in γ; the run r is weakly consistent with P in context γ if r is consistent with the step-by-step behavior of protocol P, but not necessarily with its global behavior. Note that while we are always guaranteed to have runs that are weakly consistent with P in γ, it is possible that there is no run r that is consistent with P in γ. This would happen precisely if there is no run in Ψ that is weakly consistent with P in γ. In such a case we say that P is *inconsistent* with γ; otherwise, P is *consistent* with γ. Notice that all joint protocols would be inconsistent with a context γ in which, for example, Ψ contains no run whose initial state is in $\mathcal{G}_0$. We take a situation where the joint protocol is inconsistent with the context as an indication of bad modeling. We implicitly assume that the joint protocols we consider are consistent with their contexts.

Definition 2.6 The *system representing protocol* P *in context* γ, denoted $\mathbf{R}^{rep}(P, \gamma)$, is the system consisting of all runs consistent with P in context γ. ∎

Abadi and Lamport [AL91] introduced an approach that can also be viewed as specifying a system that represents a protocol. In our notation, an *Abadi-Lamport representation* is a four-tuple $(\mathcal{G}, \mathcal{G}_0, \mathcal{N}, \Psi)$, where $\mathcal{G}$ is a set of global states, $\mathcal{G}_0$ is a set of initial states, Ψ is an admissibility condition on runs, and $\mathcal{N}$, the next-state relation, is a subset of $\mathcal{G} \times \mathcal{G}$ such that $(g, g) \in \mathcal{N}$ for all $g \in \mathcal{G}$. Roughly speaking, we can think of $\mathcal{N}$ as encoding all possible transitions of the system. The condition that $(g, g) \in \mathcal{N}$ for all $g \in \mathcal{G}$ ensures that the system can always "stutter". Such stuttering can be thought of as the result of "no-op" actions being performed by each agent in the system and by the environment (in our notation, this amounts to a joint action of the form $(\Lambda, \ldots, \Lambda)$). The definition of $\mathcal{N}$ abstracts away from actions and focuses instead on state transitions. An Abadi-Lamport representation *generates* the set of all runs $r \in \Psi$ such that $r(0) \in \mathcal{G}_0$ and $(r(i), r(i+1)) \in \mathcal{N}$ for all $i \geq 0$. Clearly this notion is similar in spirit to our notion of the system representing a protocol in a given context.

2.6 Incorporating knowledge

When analyzing a message-passing protocol, it is common to make statements such as "A does not *know* for certain that B received its acknowledgement".

To define knowledge in interpreted systems, we assume that we have a set Φ of primitive propositions, which we can think of as describing basic facts about the system. These might be such facts as "the value of the variable x is 0", "process 1's initial input was 17", "process 3 sends the message μ in round 5 of this run", or "the system is deadlocked". (For simplicity, we are assuming that we can describe the basic properties of the system adequately using propositional logic; the extension of the framework to use first-order logic is straightforward [FHMV95].) We then form more complicated formulas by closing off under conjunction, negation, and the epistemic operators $K_1, \ldots, K_n$ (where K_i stands for "agent i knows", $i = 1, \ldots, n$), E ("everyone knows"), and C ("common knowledge"), and the standard temporal operators $\bigcirc$ ("next") and U (for "until") [MP92]. As usual, we define $\Diamond\varphi$ ("eventually φ") to be an abbreviation for $true\, U \varphi$, and $\Box\varphi$ ("always φ") to be an abbreviation for $\neg\Diamond\neg\varphi$). Thus, we get formulas such as $K_1\Diamond p \wedge \neg K_2 K_1 \Diamond p$: agent 1 knows that eventually p will be true, but agent 2 does not know that agent 1 knows this. Formulas without temporal connectives (i.e., without $\bigcirc$ and U) are called *knowledge formulas*. Formulas without knowledge modalities (i.e., without K_i and without C) are called *temporal formulas*.

An *interpreted system* $\mathcal{I}$ consists of a pair $(\mathcal{R}, \pi)$, where $\mathcal{R}$ is a system over a set $\mathcal{G}$ of global states and π is an *interpretation* for the propositions in Φ over $\mathcal{G}$, which assigns truth values to the primitive propositions at the global states. Thus, for every $p \in \Phi$ and state $g \in \mathcal{G}$, we have $\pi(g)(p) \in \{\mathbf{true}, \mathbf{false}\}$. Of course, π induces also an interpretation over the points of $\mathcal{R}$; simply take $\pi(r, m)$ to be $\pi(r(m))$. Notice that Φ and π are not intrinsic to the system $\mathcal{R}$. They constitute additional structure on top of $\mathcal{R}$ that we, as outside observers, add for our convenience, to help us analyze or understand the system better. We refer to the points and states of the system $\mathcal{R}$ as points and states, respectively, of the interpreted system $\mathcal{I}$. That is, we say that the point (r, m) is in the interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ if $r \in \mathcal{R}$, and similarly, we say that $\mathcal{I}$ is a system over state space $\mathcal{G}$ if $\mathcal{R}$ is.

We can now define the truth of a formula at a point (r, m) in an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ in a straightforward way. The truth value of a primitive proposition is determined by π:

$$(\mathcal{I}, r, m) \models p \text{ (for } p \in \Phi) \text{ iff } \pi(r(m))(p) = \mathbf{true}.$$

Negation and conjunction are defined in the standard way:

$$\begin{array}{l}(\mathcal{I}, r, m) \models \varphi \wedge \psi \text{ iff } (\mathcal{I}, r, m) \models \varphi \text{ and } (\mathcal{I}, r, m) \models \psi \\ (\mathcal{I}, r, m) \models \neg\varphi \text{ iff } (\mathcal{I}, r, m) \not\models \varphi.\end{array}$$

We say agent i knows φ if φ is true at all points that i considers possible, where we interpret "i considers (r', m') possible at (r, m)" as $r_i(m) = r'_i(m')$. That is, at (r, m), agent i considers possible all points (r', m') at which he has the same local state. In such a case, we write $(r, m) \sim_i (r', m')$. It is easy to see that $\sim_i$ is an equivalence relation. We define

$$(\mathcal{I}, r, m) \models K_i\varphi \text{ iff } (\mathcal{I}, r', m') \models \varphi \text{ for all } (r', m') \text{ such that } (r, m) \sim_i (r', m').$$

Notice that this interpretation of knowledge is an *external* one, ascribed to the agents by someone reasoning about the system. We do not assume that the agents compute their knowledge in any way, nor that they can necessarily answer questions based on their knowledge. As all the references cited in the introduction show, this definition of knowledge is quite useful. Moreover, it captures the informal way people often think about programs. For example, a system designer may think "once A knows B has received the message μ, then A should stop sending μ to B". In simple examples, our formal definition of knowledge seems to capture exactly what the system designer has in mind when he uses the word "know" here. Nevertheless, there are times when a more computational notion of knowledge is appropriate. We return to this issue in Section 6.

We say that $E\varphi$ ("everyone knows φ") holds if each of the agents knows φ, and $C\varphi$ ("φ is common knowledge) holds if each of the agents knows that each of the agents knows ...that each of the agents knows φ. Defining $E^1\varphi = E\varphi$ and taking $E^{k+1}\varphi$ to be an abbreviation for $E(E^k\varphi)$, we have

$$\begin{array}{l}(\mathcal{I}, r, m) \models E\varphi \text{ iff } (\mathcal{I}, r, m) \models K_i\varphi \text{ for } i = 1, \ldots, n \\ (\mathcal{I}, r, m) \models C\varphi \text{ iff } (\mathcal{I}, r, m) \models E^k\varphi \text{ for } k = 1, 2, \ldots\end{array}$$

Finally, for the temporal operators, we have

$$\begin{array}{ll}(\mathcal{I}, r, m) \models \bigcirc\varphi & \text{iff } (\mathcal{I}, r, m+1) \models \varphi \\ (\mathcal{I}, r, m) \models \varphi U \psi & \text{iff } (\mathcal{I}, r, m') \models \psi \text{ for some } m' \geq m \text{ and} \\ & \qquad (\mathcal{I}, r, m'') \models \varphi \text{ for all } m'' \text{ such that } m \leq m'' < m'.\end{array}$$

Note that if φ is a temporal formula, then the truth of φ at a point (r, m) does not depend on $\mathcal{R}$ at all, but only on π, so we can write $(\pi, r, m) \models \varphi$. We say that r *satisfies* φ if $(\pi, r, 0) \models \varphi$ holds.

We use knowledge formulas, as the examples above suggested, to describe the knowledge necessary for agents to perform certain actions. We use temporal formulas to specify properties that we want our protocols to have, such as *safety properties*—these are invariant properties that have the form "a bad thing never happens", typically expressed with the temporal operator $\Box$—and *liveness properties*—these are properties that say "a good thing eventually happens", typically expressed using $\Diamond$ [OL82]. Admissibility conditions can also often be specified by temporal constraints. For example, to specify reliability of communication, we can use the admissibility condition *Rel* defined by $\mathit{Rel} = \{r \mid \text{all messages sent in } r \text{ are eventually received}\}$. Let $\mathit{send}(\mu, j, i)$ be a proposition that is interpreted to mean "message μ is sent to j by i", and let $\mathit{receive}(\mu, i, j)$ be a proposition that is interpreted to mean "message μ is received from i by j". Then a run r is in *Rel* precisely if $\Box(\mathit{send}(\mu, j, i) \Rightarrow \Diamond \mathit{receive}(\mu, i, j))$ holds at $(r, 0)$ (and thus at every point in r) for each message μ and processes i, j.

3 Standard Programs and Knowledge-Based Programs

3.1 Standard Programs

As discussed above, a protocol is a function from local states to sets of actions. We typically describe protocols by means of *programs* written in some programming language. We now describe a simple programming language, which is still rich enough to describe protocols, and whose syntax emphasizes the fact that an agent performs actions based on the result of a test that is applied to her local state. A (*standard*) *program* for agent i is a statement of the form:

case of
  if t_1 **do** a_1
  if t_2 **do** a_2
  ...
end case

where the t_j's are *standard tests* for agent i and the a_j's are actions of agent i (i.e., $\mathsf{a}_j \in ACT_i$). (We call such programs "standard" to distinguish them from the *knowledge-based* programs of Section 3.2. We typically omit the **case** statement if there is only one clause.) A standard test for agent i is simply a propositional formula over a set Φ_i of primitive propositions. Intuitively, once we know how to evaluate the tests in the program at the local states in L_i, we can convert this program to a protocol over L_i: at a local state ℓ, agent i nondeterministically chooses one of the (possibly infinitely many) clauses in the **case** statement whose test is true at ℓ, and executes the corresponding action.

Standard programs can be viewed as a generalization of UNITY programs [CM88]. A UNITY program consists of a collection of guarded assignment statements, such as "**if** b **then** $x \leftarrow f(x, y)$. Standard programs generalize assignments to arbitrary actions. Note that UNITY requires fairness (each statement must be attempted infinitely often), while in the framework here, fairness is not required, although it can be guaranteed by using the appropriate admissibility condition in the context.

We want to use an interpretation π to tell us how to evaluate the tests. However, not just any interpretation will do. We intend the tests in a program for agent i to be *local*, that is, to depend only on agent i's local state. It would be inappropriate for agent i's action to depend on the truth value of a test that i could not determine from her local state. We say that an interpretation π on the global states in $\mathcal{G}$ is *compatible* with a program Pg_i for agent i if every proposition that appears in Pg_i is *local* to i; that is, if q appears in Pg_i, the states g and g' are in $\mathcal{G}$, and $g \sim_i g'$, then $\pi(g)(q) = \pi(g')(q)$. If φ is a propositional formula all of whose primitive propositions are local to agent i, and ℓ is a local state of agent i, then we write $(\pi, \ell) \models \varphi$ if φ is satisfied by the truth assignment $\pi(g)$, where $g = (\ell_e, \ell_1, \ldots, \ell_n)$ is a global state such that $\ell_i = \ell$. Since all the primitive propositions in φ are local to i, it does not matter which global state g we choose, as long as i's local state in g is ℓ. Given a program Pg_i for agent i and an interpretation π compatible with Pg_i, we define a protocol that we denote Pg_i^π by setting

$$\mathsf{Pg}_i^\pi(\ell) = \begin{cases} \{\mathsf{a}_j : (\pi, \ell) \models t_j\} & \text{if } \{j : (\pi, \ell) \models t_j\} \neq \emptyset \\ \{\Lambda\} & \text{if } \{j : (\pi, \ell) \models t_j\} = \emptyset. \end{cases}$$

Intuitively, Pg_i^π selects all actions from the clauses that satisfy the test, and selects the null action Λ if no test is satisfied. In general, we get a nondeterministic protocol, since more than one test may be satisfied at a given state.

Many of the definitions that we gave for protocols have natural analogues for programs.

Definition 3.1 A *joint* program is a tuple $\mathsf{Pg} = (\mathsf{Pg}_1, \ldots, \mathsf{Pg}_n)$, where Pg_i is a program for agent i. An interpretation π is *compatible* with Pg if π is compatible with each of the Pg_i's.

From Pg and π we get a joint protocol $\mathsf{Pg}^\pi = (\mathsf{Pg}_1^\pi, \ldots, \mathsf{Pg}_n^\pi)$. An *interpreted context* is a pair (γ, π) consisting of a context γ and an interpretation π. An interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ *represents* a joint program Pg in the interpreted context (γ, π) exactly if π is compatible with Pg, and $\mathcal{R}$ represents the corresponding protocol Pg^π in context γ. The interpreted system representing Pg in (γ, π) is denoted $\mathbf{I}^{rep}(\mathsf{Pg}, \gamma, \pi)$. ∎

Note that the definition of $\mathbf{I}^{rep}(\mathsf{Pg}, \gamma, \pi)$ makes sense only if π is compatible with Pg. From now on we always assume that this is the case.

3.2 Knowledge-based programs

The notion of standard programs, in which agents perform actions based on the results of tests that are applied to their local state, is very simple. As we observed earlier, however, this notion is rich enough to describe protocols. Nevertheless, standard programs cannot be used to describe the relationships between knowledge and action that we would often like to capture. The issue is perhaps best understood by considering the muddy children puzzle [Bar81, HM90].

In this puzzle, a number of children are playing in the mud. Their father then comes along and says "At least one of you has mud on your forehead". He then repeatedly asks the children if they know whether they have mud on their forehead. If so, they are supposed to answer "Yes"; otherwise they should answer "No". If we take the proposition p_i to represent "child i has mud on his forehead", then it seems quite reasonable to think of child i as following the program MC_i (the MC stands for "Muddy Children"):

case of
  if $\mathit{childheard}_i \wedge (K_i p_i \vee K_i \neg p_i)$ **do** say "Yes"
  if $\mathit{childheard}_i \wedge \neg K_i p_i \wedge \neg K_i \neg p_i$ **do** say "No"
end case.

Here *childheard*$_i$ is a primitive proposition that is true at a given state if child i heard the father's question "Does any of you know whether you have mud on your own forehead?" in the previous round. Unfortunately, MC_i is not a program as we have defined it. Besides propositional tests, it has tests for knowledge such as $K_i p_i \vee K_i \neg p_i$. Moreover, we cannot use our earlier techniques to associate a protocol with a program, since the truth value of such a knowledge test cannot be determined by looking at a local state in isolation.

We call a program of the form above a *knowledge-based* program, to distinguish it from the standard programs defined in Section 3. Formally, a knowledge-based program for agent i has the form:

case of
  if $t_1 \wedge k_1$ **do** a_1
  if $t_2 \wedge k_2$ **do** a_2
  ...
end case

where the t_j's are standard tests, the k_j's are *knowledge tests* for agent i, and the a_j's are actions of agent i. A knowledge test for agent i is a Boolean combination of formulas of the form $K_i \varphi$, where φ can be an arbitrary formula that may include other modal operators, including common knowledge and temporal operators. Intuitively, the agent selects an action based on the result of applying the standard test to her local state and applying the knowledge test to her "knowledge state", in a sense that will be made precise below. In the program MC_i, the test $childheard_i$ is a standard test, while $K_i p_i \vee K_i \neg p_i$ and $\neg K_i p_i \wedge \neg K_i \neg p_i$ are knowledge tests. In any given clause, we can omit either the standard test or the knowledge test; thus, a standard program is a special case of a knowledge-based program. We define a *joint* knowledge-based program to be a tuple $\mathsf{Pg} = (\mathsf{Pg}_1, \ldots, \mathsf{Pg}_n)$, with one knowledge-based program for each agent.

The notion discussed here of a knowledge-based program is from [FHMV95]. Although the idea of a knowledge-based program was implicit in the discussion in [HF85], the first formal definition seems to have been given by Kurki-Suonio [Kur86] and by Shoham [Sho93]. Kurki-Suonio and Shoham, however, did not work with interpreted systems. Rather, they assumed that an agent's knowledge was explicitly encoded in his local state (and thus, in our terminology, was independent of the interpreted system). This means that their knowledge-based programs are really more like our standard programs, although some of the tests in their programs are intuitively thought of as tests for knowledge.

We have described the syntax of knowledge-based programs. It remains to give formal semantics to knowledge-based programs. Just as we think of a standard program as inducing a protocol that determines an agent's actions, we also want to think of a knowledge-based program as inducing a protocol. It is not obvious, however, how to associate a protocol with a knowledge-based program. A protocol is a function from local states to actions. To go from a standard program to a protocol, all we needed to do was to evaluate the standard tests at a given local state, which we did using interpretations. In a knowledge-based program, we also need to evaluate the knowledge tests. But in our framework, a knowledge test depends on the whole interpreted system, not just the local state. It may well be the case that agent i is in the same local state ℓ in two different interpreted systems $\mathcal{I}_1$ and $\mathcal{I}_2$, and the test $K_i p$ may turn out to be true at the local state ℓ in $\mathcal{I}_1$, and false at the local state ℓ in $\mathcal{I}_2$.

To deal with this problem, we proceed as follows. Given an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$, we associate with a joint knowledge-based program $\mathsf{Pg} = (\mathsf{Pg}_1, \ldots, \mathsf{Pg}_n)$ a joint protocol that is denoted $\mathsf{Pg}^{\mathcal{I}} = (\mathsf{Pg}_1^{\mathcal{I}}, \ldots, \mathsf{Pg}_n^{\mathcal{I}})$. Intuitively, we evaluate the standard tests in Pg according to π, and evaluate the knowledge tests in Pg according to $\mathcal{I}$. As in the case of standard programs, we require that π be compatible with Pg, that is, that every proposition appearing in a standard test in Pg_i should be local to i. Note that we place the locality requirement only on the propositions appearing in the standard tests, not on the propositions appearing in the knowledge tests. We wish to define $\mathsf{Pg}_i^{\mathcal{I}}(\ell)$ for all local states ℓ of agent i. To define this, we first define when a test φ holds in a local state ℓ with respect to an interpreted system $\mathcal{I}$, denoted $(\mathcal{I}, \ell) \models \varphi$. (Note that this overloads $\models$, since previously we had a triple $(\mathcal{I}, r, m)$ on the left-hand-side of $\models$.)

If φ is a standard test and $\mathcal{I} = (\mathcal{R}, \pi)$ then, in analogy to Section 3, we define

$(\mathcal{I}, \ell) \models \varphi$ iff $(\pi, \ell) \models \varphi$.

Since φ is a standard test in Pg_i, it must be local to agent i, so this definition makes sense. If φ is a knowledge test of the form $K_i \psi$, we define

$(\mathcal{I}, \ell) \models K_i \psi$ iff $(\mathcal{I}, r, m) \models \psi$ for all points (r, m) of $\mathcal{I}$ such that $r_i(m) = \ell$.

Finally, for conjunctions and negations of knowledge tests, we follow the standard treatment.

Note that $(\mathcal{I},\ell) \models \varphi$ is defined even if the local state ℓ does not occur in $\mathcal{I}$. In this case it is almost immediate from the definitions that $(\mathcal{I},\ell) \models K_i(\mathit{false})$. This means that one of the standard properties of knowledge fails, namely, that whatever is known is true ($K_i\varphi \Rightarrow \varphi$). On the other hand, if ℓ does occur in $\mathcal{I}$, then K_i behaves in the standard way. This follows since if $\ell = r_i(m)$ for some point (r,m) in $\mathcal{I}$, then it is not hard to show that $(\mathcal{I},\ell) \models K_i\varphi$ iff $(\mathcal{I},r,m) \models K_i\varphi$.

We can now define

$$\mathsf{Pg}_i^{\mathcal{I}}(\ell) = \begin{cases} \{\mathsf{a}_j : (\mathcal{I},\ell) \models t_j \wedge k_j\} & \text{if } \{j : (\mathcal{I},\ell) \models t_j \wedge k_j\} \neq \emptyset \\ \{\Lambda\} & \text{if } \{j : (\mathcal{I},\ell) \models t_j \wedge k_j\} = \emptyset. \end{cases}$$

Intuitively, the actions prescribed by i's protocol $\mathsf{Pg}_i^{\mathcal{I}}$ are exactly those prescribed by Pg_i in the interpreted system $\mathcal{I}$.

Let Pg be a standard program. Then Pg is also a knowledge-based program, with no knowledge tests. Consider an interpreted system $\mathcal{I} = (\mathcal{R},\pi)$. We can associate a protocol with Pg in two ways. We can think of Pg as a standard program, and associate with it the protocol Pg^π, or we can think of Pg as a knowledge-based program and associate with it the protocol $\mathsf{Pg}^{\mathcal{I}}$. It is easy to see that our definitions guarantee that these protocols are identical.

Roughly speaking, the knowledge-based protocols of [HF89] bear the same relation to knowledge-based programs as protocols bear to standard programs. Formally, a knowledge-based protocol is defined in [HF89] to be a function from local states and interpreted systems to actions. We can associate a knowledge-based protocol with a knowledge-based program in an obvious way: Given a knowledge-based program Pg, we can associate with it the knowledge-based protocol P such that $P(\mathcal{I},\ell) = \mathsf{Pg}^{\mathcal{I}}(\ell)$ for all local states ℓ. Knowledge-based protocols can be viewed as an intermediate step between knowledge-based programs and protocols. We find it convenient here to go directly from knowledge-based programs to (standard) protocols, skipping this intermediate step. Thus, we do not deal with knowledge-based protocols in this paper.

The mapping from knowledge-based programs to protocols allows us to define what it means for an interpreted system to represent a knowledge-based program in a given interpreted context by reduction to the corresponding definition for protocols. Thus, an interpreted system $\mathcal{I} = (\mathcal{R},\pi)$ *represents* Pg in (γ,π) if π is compatible with Pg and if $\mathcal{R}$ represents $\mathsf{Pg}^{\mathcal{I}}$ in γ. This means that to check if $\mathcal{I}$ represents Pg, we check if $\mathcal{I}$ represents the protocol obtained by evaluating the knowledge tests in Pg with respect to $\mathcal{I}$ itself. Because of the circularity of the definition, it is not necessarily the case that there is a unique interpreted system representing a knowledge-based program There may be more than one or there may be none. In contrast, there can be at most one interpreted system that represents a standard program. This issue is explored in more detail in Sections 4 and 5, where, among other things, conditions are described under which a knowledge-based program is guaranteed to have a unique system representing it. The notion of representation can be viewed as a notion of *equilibrium*: $\mathcal{I}$ represents Pg, if whenever each agent i runs its program Pg_i with respect to the interpreted system $\mathcal{I}$ (in the interpreted context (γ,π)), then the joint knowledge-based program Pg indeed gives rise to the interpreted system $\mathcal{I}$. This is reminiscent of the notion of *Nash equilibrium* in game theory, where a tuple $\langle f_1,\ldots,f_k\rangle$ of strategies in a k-player game is a Nash equilibrium if, for each player i, the strategy f_i is the best response to player j using f_j for $j \neq i$ [FT91]. That is, if each player believes that the other players play as if they are at the equilibrium point $\langle f_1, \ldots, f_k \rangle$, then the players will indeed end up at that equilibrium point.

4 Getting unique representations

As we mentioned in the previous section, in general there is no unique interpreted system that represents a knowledge-based program in a given context. In this section, we provide sufficient conditions to guarantee the existence of a unique representation. We begin with an example that illustrates why we may get more than one system representing a knowledge-based program.

Example 4.1 Suppose we have a system consisting of only one agent, agent 1, who has a bit that is initially set to 0. Suppose agent 1 runs the following simple knowledge-based program NU (for "not unique"):

if $K_1(\Diamond(\mathit{bit} = 1))$ **do** $\mathit{bit} := 1$.

Intuitively, $\mathit{bit} := 1$ has the effect of assigning the value 1 to the bit. According to NU, agent 1 sets the bit to 1 if she knows that eventually the bit is 1, and otherwise does nothing. It should be clear that there are two ways that agent 1 could be consistent with the program: either by never setting the bit to 1 or by setting the bit to 1 in the first round. We can formalize this by considering the context $\gamma^{nu} = (P_e, \mathcal{G}_0, \tau, \mathit{True})$, defined as follows: We take agent 1's local state to be either 0 or 1; we think of this local state as representing the value of the bit. We take the environment's state to always be λ (the environment plays no role in this example). Since the bit is initially 0, we take $\mathcal{G}_0 = \{(\lambda, 0)\}$. We assume that the environment's action is always Λ, so $P_e(\lambda) = \Lambda$. The agent's action is either Λ or $\mathit{bit} := 1$. The effect of τ is to reset the bit as appropriate; thus, $\tau(\Lambda, \Lambda)(\lambda, k) = (\lambda, k)$ and $\tau(\Lambda, \mathit{bit} := 1)(\lambda, k) = (\lambda, 1)$. This completes the description of γ^{nu}. Finally, we define π^{nu} in the obvious way: $\pi^{nu}((\lambda, k))(\mathit{bit} = 1)$ is true exactly if $k = 1$.

Let r^0 be the run where agent 1 does nothing, starting in the initial state $(\lambda, 0)$; thus, $r^0(m) = (\lambda, 0)$ for all $m \geq 0$. Let r^j, for $j \geq 1$, be the run where agent 1 sets the bit to 1 in round j, after starting in the initial state $(\lambda, 0)$; thus, $r^j(m) = (\lambda, 0)$ for $m < j$, and $r^j(m) = (\lambda, 1)$ for $m \geq j$. It is easy to see that the only runs that we can have in context γ^{nu} are of the form r^j. It is also not hard to see that no run of the form r^j for $j > 1$ can be in an interpreted system consistent with NU. For if r^j is in an interpreted system $\mathcal{I}$ consistent with NU, then since agent 1 sets the bit to 1 in round j of r^j, it must be the case that $(\mathcal{I}, r^j, j-1) \models K_1(\Diamond(\mathit{bit} = 1))$. But clearly $(r^j, 0) \sim_1 (r^j, j-1)$. Thus, $(\mathcal{I}, r^j, 0) \models K_1(\Diamond(\mathit{bit} = 1))$. Since $\mathcal{I}$ is consistent with $\mathsf{NU}^{\mathcal{I}}$, this means that agent 1 should have set the bit to 1 in round 1 of r^j, a contradiction. Thus, the set of runs in any interpreted system consistent with NU must be a nonempty subset of $\{r^0, r^1\}$. Let $\mathcal{R}^j$ be the system consisting of the single run r^j, for $j = 0, 1$, and let $\mathcal{I}^j = (\mathcal{R}^j, \pi^{nu})$. We claim that both $\mathcal{I}^0$ and $\mathcal{I}^1$ represent NU in the context (γ^{nu}, π^{nu}). Clearly, in $\mathcal{I}^1$, agent 1 knows $\Diamond(\mathit{bit} = 1)$, since this formula is true at every point in $\mathcal{I}^1$, so the only possible action that she can take is to set the bit to 1 in round 1, which is precisely what she does in r^1. On the other hand, in $\mathcal{I}^0$, agent 1 never knows $\Diamond(\mathit{bit} = 1)$, since it is false at all points in r^0. This means that according to the protocol $\mathsf{NU}^{\mathcal{I}^0}$, agent 1 never sets the bit to 1, so the only run consistent with $\mathsf{NU}^{\mathcal{I}^0}$ is r^0. It follows that both $\mathcal{I}^0$ and $\mathcal{I}^1$ represent NU in (γ^{nu}, π^{nu}). It is easy to see that the interpreted system $\mathcal{I}^2 = (\mathcal{R}^2, \pi^{nu})$, where $\mathcal{R}^2 = \{r^0, r^1\}$, is not consistent with NU, so that $\mathcal{I}^0$ and $\mathcal{I}^1$ are in fact the only interpreted systems that represent NU in this context.

Now consider the program that intuitively says "set the bit to 1 exactly if you know you will never set the bit to 1". No interpreted system can be consistent with this program, since it amounts to saying "set the bit to 1 exactly if you know you should not". We can capture this intuition by means of the following knowledge-based program NU':

if $K_1(\neg\Diamond(bit = 1))$ **do** $bit := 1$.

There can be no interpreted system consistent with NU' in the context (γ^{nu}, π^{nu}): Arguments similar to those used before show that the only runs that can be in an interpreted system consistent with NU' are r^0 and r^1. Thus, $\mathcal{I}^0$, $\mathcal{I}^1$, and $\mathcal{I}^2$ are the only possible candidates for interpreted systems consistent with NU'. It is straightforward to show that none of these interpreted systems in fact are consistent with NU'. Hence, there is no interpreted system that is consistent with or represents NU'. We take this to mean that the program NU' is inconsistent with the interpreted context (γ^{nu}, π^{nu}). ∎

In Example 4.1, we saw programs that determine an agent's current actions as a function of his knowledge about the actions that he will perform in the future. This direct reference to knowledge about the future seemed to make it possible to define both nonsensical programs such as NU', which cannot be implemented by any standard program, and ambiguous programs such as NU, which can be implemented in more than one way. We remark that the explicit use of future temporal operators such as $\Diamond$ is not crucial to this example. Essentially the same effect can be achieved without such operators (see [FHMV95, Exercise 7.5] for an example).

Example 4.1 shows that a knowledge-based program may not have a unique interpreted system representing it. Is this a problem? Not necessarily. Of course, if there is no interpreted system representing the program, then this program is not of any practical interest. Such programs can be viewed as inconsistent. We return to this issue later in the section. On the other hand, when there is more than one interpreted system representing a knowledge-based program, the program should be viewed as a high-level specification that is satisfied by many interpreted systems. For example, consider the knowledge-based program NU from Example 4.1:

if $K_1(\Diamond(bit = 1))$ **do** $bit := 1$.

This program can be viewed as saying: "if you know that you are going to take an action, then take it as soon as possible". Appropriately, as we have shown, this program is represented by two interpreted systems, one in which the action is taken immediately and one in which the action is never taken. Thus, while a standard program (in a given interpreted context) is a complete description of the behavior of the agents, this is not the case with a knowledge-based program.

In many situations, however, there is a strong intuition that a knowledge-based program does completely describe the behavior of the agents, and consequently, the program ought to be represented by a unique interpreted system. For example, in the case of the muddy children puzzle, we expect the behavior of the children following the knowledge-based program MC, described earlier, to be uniquely determined. In the remainder of this section, we describe necessary and sufficient conditions for there to be a unique interpreted system representing a knowledge-based program. The conditions we consider here are similar in spirit to those shown in [HF89] to guarantee a representation of a knowledge-based protocol that was *canonical* in a certain sense. Nevertheless, there are significant technical differences between the framework here and that of [HF89] (for example, in [HF89] there were no contexts and no programs, and the notion of a system representing a program was not considered). These differences result in significant differences between the proof here and that of [HF89]. One payoff is that the claims we prove in this version are more general, and apply in many cases of practical interest to which those of [HF89] do not apply. We start with an informal discussion of the result and then make things more formal.

Why may one feel that there should be a unique interpreted system representing MC? Intuitively, it is because, once we fix the initial set of states, we can start running the program step by step, generating the run as we go. If r is a run over $\mathcal{G}$, the *prefix of r through time m*, or the *m-prefix of r*, denoted $\mathit{Pref}_m(r)$, is the sequence of the first $m+1$ global states in r, i.e., it is a function ρ from $\{0,\ldots,m\}$ to $\mathcal{G}$ such that $\rho(k) = r(k)$ for $k = 0,\ldots,m$. If $\mathcal{R}$ is a set of runs, then $\mathit{Pref}_m(\mathcal{R})$ is the set of m-prefixes of the runs in $\mathcal{R}$, i.e., $\mathit{Pref}_m(\mathcal{R}) = \{\mathit{Pref}_m(r) \mid r \in \mathcal{R}\}$. If $\mathcal{I} = (\mathcal{R}, \pi)$, we define $\mathit{Pref}_m(\mathcal{I}) = (\mathit{Pref}_m(\mathcal{R}), \pi)$. Suppose that we can generate all m-prefixes of runs. Once we have all m-prefixes, at any given point (r, m), the children in that situation can determine whether they do indeed know whether their own forehead is muddy, and thus can take the appropriate action at the next step. This allows us to generate all $(m+1)$-prefixes.

The key reason that this idea works is that the prefixes that we have already constructed are sufficient to determine the truth of the knowledge tests in the children's program. In general, this might not be the case. To understand why, suppose we have a knowledge-based program $\mathsf{Pg} = (\mathsf{Pg}_1,\ldots,\mathsf{Pg}_n)$, and Pg_i includes a test such as $K_i\varphi$. Suppose that we have indeed constructed all the m-prefixes of runs of Pg. For agent i to know what actions to perform next at a point (r, m), the knowledge test $K_i\varphi$ has to be evaluated. As long as this can be done solely by considering points of the form (r', m') with $m' \leq m$—intuitively, these are the points we have already constructed—then there is no problem. If, on the other hand, φ is a temporal formula such as the formula $\Diamond(\mathit{bit} = 1)$ that appears in the program NU in Example 4.1, then we may not be able to evaluate the truth of φ in the prefixes we have constructed thus far. Even if φ is a nontemporal formula, there may be a problem. For example, suppose the time m is encoded in the environment's state, and φ is the formula $m \leq 1$, which is true at all time m points with m less than or equal to 1. Then $K_1(m \leq 1)$ may be false at a point $(r, 1)$ if agent 1 does not know the time, i.e., if $(r, 1) \sim_1 (r', k)$ for some point (r', k), where $k > 1$. Note, however, that there is no point that occurs in a 1-prefix and "witnesses" the fact that $K_1(m \leq 1)$ fails at $(r, 1)$, since the formula $m \leq 1$ is true at all points of the form $(r', 0)$ or $(r', 1)$. This discussion suggests that to make the inductive construction work, if a test $K_i\varphi$ in the program is false, there must be a "witness" to its falsity in some prefix we have already constructed, i.e., the result of the test $K_i\varphi$ should "depend on the past".

Even if tests "depend on the past", there may be a problem. Suppose we are interested in running the knowledge-based program Pg in the interpreted context (γ, π), where $\gamma = (P_e, \mathcal{G}_0, \tau, \Psi)$, and all tests "depend on the past" in the sense we have just discussed. What should the system representing Pg be? Intuitively, it should consist of all runs in Ψ whose prefixes arise in the inductive construction. But suppose the admissibility condition Ψ does not include a run with a prefix ρ that arises in the construction. This means that we cannot include a run with prefix ρ in the system. This, in turn, might mean that a "witness" that we counted on in the course of the inductive construction may not occur in the system, thus undermining our evaluation of the tests.

We now show that there is a unique system that represents Pg if tests "depend on the past" and if the admissibility condition Ψ is "reasonable". Intuitively, the property we shall require Ψ to satisfy ensures that for every prefix that arises in the inductive construction, there is some run in Ψ with that prefix that we can include in the system we are constructing.

We first formalize *dependence on the past*. Intuitively, a formula φ depends on the past in a class $\mathcal{J}$ of interpreted systems if, in order to determine whether φ is true at the point (r, m) of an interpreted system $\mathcal{I} \in \mathcal{J}$, we need only look at m-prefixes of runs in $\mathcal{I}$; whatever may happen after time m can not affect the truth of φ. The formal definition captures the idea of "whatever may happen" by considering any interpreted system in $\mathcal{J}$ that agrees with $\mathcal{I}$ up to time m.

Definition 4.2 Formula ψ *depends on the past in the class $\mathcal{J}$ of interpreted systems* if its truth at a point (r, m) of an arbitrary interpreted system $\mathcal{I} \in \mathcal{J}$ depends only on $\mathit{Pref}_m(r)$ and $\mathit{Pref}_m(\mathcal{I})$. More precisely, we require that for all m, for all interpreted systems $\mathcal{I}, \mathcal{I}' \in \mathcal{J}$ such that $\mathit{Pref}_m(\mathcal{I}) = \mathit{Pref}_m(\mathcal{I}')$, and for all runs r in $\mathcal{I}$ and r' in $\mathcal{I}'$, if $\mathit{Pref}_m(r) = \mathit{Pref}_m(r')$, then $(\mathcal{I}, r, m) \models \psi$ if and only if $(\mathcal{I}', r', m) \models \psi$. A *knowledge-based program* Pg *depends on the past in* $\mathcal{J}$ if all the tests in Pg depend on the past in $\mathcal{J}$. ∎

In general, it may be difficult to tell if a program depends on the past. As we shall see, however, there are relatively simple sufficient conditions that guarantee dependence on the past and are applicable in many cases of interest.

We next make precise the condition that is required for an admissibility condition Ψ to be "reasonable". Recall that a run r is weakly consistent with a protocol P in context $\gamma = (P_e, \mathcal{G}_0, \tau, \Psi)$ if r is consistent with P except that it may not be in Ψ. Intuitively, Ψ is "reasonable" if it does not rule out prefixes that are "consistent" with P in γ. We formalize this intuition in the following definition.

Definition 4.3 A context γ is *nonexcluding* if (a) $\mathcal{G}_0 \cap \mathit{Pref}_0(\Psi) \neq \emptyset$ (note that a 0-prefix can be viewed both as a prefix and as a global state), and (b) for every protocol P, if a run r is weakly consistent with P in the context γ, and the m-prefix ρ of r is in $\mathit{Pref}_m(\Psi)$, then there is a run $r' \in \Psi$ with m-prefix ρ that is consistent with P in γ. ∎

Note that condition (a) gets our inductive construction started (since Ψ cannot exclude all the initial states), and condition (b) guarantees that Ψ does not exclude a prefix ρ that has been constructed in our inductive construction from being extended to a run. While it may seem difficult to check whether a context is nonexcluding, many contexts of interest are easily shown to be nonexcluding. For one thing, a context $\gamma = (P_e, \mathcal{G}_0, \tau, \Psi)$ is guaranteed to be nonexcluding if Ψ is *True*. More generally, in many contexts of interest the admissibility condition constrains only the "limit" behavior of the run; this is the case, for example, with fairness requirements. In such cases, it is typically not hard to show that the context under consideration is nonexcluding. We remark that the property of being nonexcluding is a property of the context $\gamma = (P_e, \mathcal{G}_0, \tau, \Psi)$ as a whole and not in general a property of Ψ by itself.

We are now almost ready to state our necessary and sufficient conditions for there to be a unique interpreted system representing a knowledge-based program with respect to nonexcluding contexts. We actually break the problem up into two parts. We first provide necessary and sufficient conditions for the existence of at least one system that represents a given program, and then provide necessary and sufficient conditions for there to be at most one system that represents a program. Putting these results together, we get necessary and sufficient conditions for there to be a unique system representing a given program with respect to nonexcluding contexts. Our conditions involve two natural closure conditions on a class $\mathcal{J}$ of interpreted systems. The first says that $\mathcal{J}$ is closed under "application" of Pg.

Definition 4.4 A class $\mathcal{J}$ of interpreted systems is Pg-*closed with respect to* (γ, π) if whenever $\mathcal{I}$ is in $\mathcal{J}$, then so is $\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma, \pi)$. ∎

That is, $\mathcal{J}$ is Pg-closed if $\mathcal{J}$ contains all the interpreted systems that are obtained by running Pg with respect to interpreted systems in $\mathcal{J}$.

We now consider the second closure condition.

Definition 4.5 A sequence $\mathcal{R}^0, \mathcal{R}^1, \ldots$ of systems is *prefix-compatible* if $\mathit{Pref}_m(\mathcal{R}^{m'}) = \mathit{Pref}_m(\mathcal{R}^m)$ for all $m \geq 0$ and $m' \geq m$. ∎

Intuitively, the m-prefix is determined by $\mathcal{R}^m$. Let us define a *limit* of a prefix-compatible sequence to be a system $\mathcal{R}$ such that $\mathit{Pref}_m(\mathcal{R}) = \mathit{Pref}_m(\mathcal{R}^m)$ holds for all $m \geq 0$. It is easy to see that every prefix-compatible sequence has a limit. As we now show, a prefix-compatible sequence can have more than one limit. Assume we have a system where process 1 sends process 2 a message in the first round. Process 1's state changes from s_1 to t_1 after sending the message, and then continues to be t_1 from then on. Process 2's state changes from s_2 to t_2 when it receives the message, and then continues to be t_2 from then on. For each nonnegative integer k, let r^k be a run where process 2 receives the message at round k, so that its state changes to t_2 at time k. Let $\mathcal{R}$ be a system consisting precisely of all of these runs r^k. Now let r^∞ be another run where process 2 never receives the messages, so that it is always in state s_2, and let $\mathcal{R}'$ be the system consisting of the runs in $\mathcal{R}$, along with this new run r^∞. Clearly the constant sequence $\mathcal{R}, \mathcal{R}, \mathcal{R}, \ldots$ is prefix-compatible, and $\mathcal{R}$ is a limit of this sequence. But $\mathcal{R}'$ is also a limit of this sequence, since $\mathit{Pref}_m(\mathcal{R}') = \mathit{Pref}_m(\mathcal{R})$ for every m.[1]

Definition 4.6 A set $\mathcal{H}$ of systems *has limits* if $\mathcal{H}$ contains a limit of every prefix-compatible sequence of members of $\mathcal{H}$. A set $\mathcal{H}$ of systems is *limit closed* if it contains every limit of every prefix-compatible sequence of members of $\mathcal{H}$. ∎

All of these definitions can be extended in a natural way to deal not just with systems but with interpreted systems. For example, a sequence $\mathcal{I}^0, \mathcal{I}^1, \ldots$ of interpreted systems, where $\mathcal{I}^m =$

[1] We note that the reason the limit is not unique is that under the appropriate topology, the space of systems is not *Hausdorff*, that is, two distinct points may not be separable by an open set that contains one and not the other.

$(\mathcal{R}^m, \pi)$ for each m (i.e., all interpreted systems in the sequence have the same interpretation π), is prefix-compatible if the corresponding sequence $\mathcal{R}^0, \mathcal{R}^1, \ldots$ of systems is prefix-compatible. If $\mathcal{I}$ is an interpreted system, then the singleton set $\{\mathcal{I}\}$ has limits (since $\mathcal{I}$ is a limit of the constant sequence $\mathcal{I}, \mathcal{I}, \mathcal{I}, \ldots$), but is not necessarily limit closed (since as we saw above, another interpreted system may also be a limit of this constant sequence).

Theorem 4.7 *Let γ be a nonexcluding context. There is at least one interpreted system representing the knowledge-based program* Pg *in context (γ, π) iff there exists a nonempty set $\mathcal{J}$ of interpreted systems that is* Pg*-closed with respect to (γ, π) and has limits, such that* Pg *depends on the past in $\mathcal{J}$.*

Proof The proof of the "only if" direction is easy, as we now show. Assume that Pg is represented by some system $\mathcal{I}$ in (γ, π). Then the set $\mathcal{J}$ consisting of just $\mathcal{I}$ is nonempty, Pg-closed with respect to (γ, π) (since $\mathcal{I} = \mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma, \pi)$), and has limits. (It has limits, since as we observed above, every singleton set has limits.) Since $\mathcal{J}$ is a singleton set, it easily follows that Pg depends on the past in $\mathcal{J}$.

For the "if" direction, note that finding an interpreted system representing Pg in the interpreted context (γ, π) corresponds precisely to finding a fixed point $\mathcal{I}$ of the "equation" $\mathcal{I} = f(\mathcal{I})$, where $f(\mathcal{I}) = \mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma, \pi)$. We attempt to construct a fixed point by starting at an arbitrary point and continually applying Pg. We define the limit step of this construction by applying the fact that $\mathcal{J}$ has limits. It turns out we reach a fixed point at the $(\omega + 1)$st step of the construction (where ω is the first infinite ordinal). We proceed as follows.

Let $\mathcal{I}^{-1}$ be some member of $\mathcal{J}$. (There is one, since $\mathcal{J}$ is nonempty. The unusual choice of superscript makes some of the technical claims in the proof easier to state.) Suppose inductively that we have constructed $\mathcal{I}^m$. We then define $\mathcal{I}^{m+1} = \mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}^m}, \gamma, \pi)$. Since $\mathcal{J}$ is Pg-closed, it follows by a straightforward induction that $\mathcal{I}^m \in \mathcal{J}$ for each m. We shall show in the appendix that the sequence $\mathcal{I}^0, \mathcal{I}^1, \mathcal{I}^2, \ldots$ is prefix-compatible, given that Pg depends on the past in $\mathcal{J}$ and that γ is nonexcluding (see Claim A.3). Since $\mathcal{J}$ has limits, there is a limit $\mathcal{I}^\omega$ of this sequence in $\mathcal{J}$. We now continue our construction into the infinite ordinals. Define $\mathcal{I}^{\theta+1} = \mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}^\theta}, \gamma, \pi)$, for $\theta = \omega$ and $\theta = \omega + 1$. In the appendix, we show that $\mathcal{I}^{\omega+1} = \mathcal{I}^{\omega+2}$ (see Claims A.4 and A.5). This proves that $\mathcal{I}^{\omega+1}$ is an interpreted system representing Pg in the interpreted context (γ, π). ∎

Although it may not be obvious, this construction actually formalizes the intuition we gave earlier in the section. Our discussion there was in terms of prefixes of runs. The idea was that by inductively assuming that we have defined all m-prefixes, we could then construct all $(m+1)$-prefixes. The desired system would then be a limit of this construction. As we mentioned above, the sequence $\mathcal{I}^0, \mathcal{I}^1, \mathcal{I}^2, \ldots$ is prefix-compatible. Suppose $\mathcal{I}^m = (\mathcal{R}^m, \pi)$, for $m = 0, 1, 2, \ldots$. So the prefixes $\mathit{Pref}_m(\mathcal{R}^m)$, for $m = 0, 1, 2, \ldots$, form an increasing sequence of prefixes (i.e., the prefixes in $\mathit{Pref}_{m+1}(\mathcal{R}^{m+1})$ extend those in $\mathit{Pref}_m(\mathcal{R}^m)$), and correspond precisely to the prefixes we constructed in our informal proof. Since $\mathcal{J}$ consists of systems, which involve sets of runs, rather than sets of prefixes of runs, we are forced to use $\mathcal{R}^m$ in the construction rather than $\mathit{Pref}_m(\mathcal{R}^m)$. Nevertheless, since Pg depends on the past in $\mathcal{J}$, the "suffixes" in $\mathcal{R}^m$ (i.e., the part of the run after time m) are irrelevant; only the prefixes matter. The purpose of the transfinite steps in the construction is to ensure that the runs that we get are in Ψ, since it is possible that $\mathcal{I}^\omega$ contains runs that are not in Ψ.

Given a knowledge-based program Pg and an interpreted context (γ, π), let $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$ be the set of interpreted systems that represent Pg in (γ, π). Theorem 4.7 gives conditions that guarantee that $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$ is nonempty; that is, conditions that guarantee that there is at least one system that represents Pg in (γ, π). We now give a condition that guarantees that $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$ contains at most one system; that is, conditions that guarantee that there is at most one system that represents Pg in (γ, π).

Theorem 4.8 *Let γ be a nonexcluding context. There is at most one system representing the knowledge-based program* Pg *in (γ, π) iff* Pg *depends on the past in* $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$.

Proof The "only if" part is immediate, since if there is at most one system in, then Pg depends on the past in $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$. The proof of the "if" part appears in the appendix (see Claim A.6). ∎

Note that the theorem holds trivially if $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$ is empty.

Putting together Theorems 4.7 and 4.8, we obtain a necessary and sufficient condition for a program to have a unique system representing it (under the assumption that the context is nonexcluding).

Theorem 4.9 *Let γ be a nonexcluding context. There is a unique system representing the knowledge-based program* Pg *in (γ, π) iff there exists a nonempty set $\mathcal{J}$ containing* $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$ *that is* Pg*-closed with respect to (γ, π) and has limits such that* Pg *depends on the past in $\mathcal{J}$.*

Proof Clearly if there is a unique system representing Pg, say $\mathcal{I}$, then the singleton set $\{\mathcal{I}\}$ contains $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$, is Pg-closed with respect to (γ, π), and has limits. Also, Pg depends on the past in $\{\mathcal{I}\}$. For the converse, assume that there is a nonempty set $\mathcal{J}$ containing $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$ that is Pg-closed with respect to (γ, π) and has limits such that Pg depends on the past in $\mathcal{J}$. It follows immediately from Theorem 4.7 that there is at least one system representing Pg in (γ, π). Moreover, since $\mathcal{J}$ contains $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$ and Pg depends on the past in $\mathcal{J}$, it is immediate from the definitions that Pg also depends on the past in $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$. Thus, it follows from Theorem 4.8 that there is at most one system representing Pg in (γ, π). Hence, there is exactly one system representing Pg in (γ, π). ∎

How useful is the characterization given by Theorem 4.9? That depends, of course, on how hard it is to find a class $\mathcal{J}$ of interpreted systems that satisfies the assumptions of the theorem. One could try to take $\mathcal{J}$ to be $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$, but then one has to show that $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$ is nonempty. We now describe one candidate for $\mathcal{J}$ that often does satisfy the conditions of Theorem 4.9.

Definition 4.10 Given a program Pg and an interpreted context (γ, π), let $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ consist of all interpreted systems $\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma, \pi)$, where $\mathcal{I}$ is of the form $(\mathcal{R}, \pi)$. (Notice that the interpretation π in the pair $(\mathcal{R}, \pi)$ is the same as that in the interpreted context (γ, π).) The system $\mathcal{R}$ can be arbitrary, except that it must satisfy one constraint: all the global states that arise in runs of $\mathcal{R}$ must be in the domain of π (that is, they must be among the global states implicitly determined by the context γ). Thus, $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ consists of all the systems that represent protocols of the form $\mathsf{Pg}^{\mathcal{I}}$ in γ. ∎

We can expect some of the systems that represent protocols of the form $\mathsf{Pg}^{\mathcal{I}}$ in γ to be very different from systems that represent Pg in (γ, π). Nevertheless, certain aspects of the structure of Pg will be reflected in all the systems in $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$. For example, standard tests clearly behave in the same way in all these systems (since we are using the same interpretation π), and certain properties of Pg may also be reflected in all these systems. For example, if the structure of Pg guarantees that a non-null action is performed by each process in every round, then this will be reflected in every system that represents a protocol of the form $\mathsf{Pg}^{\mathcal{I}}$ in γ; the exact action performed in a given round may change from one such system to another.

Clearly $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ is Pg-closed with respect to (γ, π); indeed, it is almost immediate that any superset of $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ is as well. It does not in general have limits. Define $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$ to be the limit closure of $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$; that is, $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$ is the smallest set that contains $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ and is limit closed. (We remark that for our purposes, we could just as well take $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$ to be any set that contains $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ and has limits; for definiteness, we take $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$ to be the limit closure.) Since $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi) \subseteq \mathcal{J}(\mathsf{Pg}, \gamma, \pi) \subseteq \mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$, the following result follows immediately from Theorem 4.9.

Corollary 4.11 *If γ is nonexcluding and Pg depends on the past in $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$, then there is a unique interpreted system representing Pg in (γ, π).*

How hard is it to show that Pg depends on the past in $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$? That depends on Pg, of course, but the following results provide some useful sufficient conditions.

Notice that our formal definition of dependence on the past does not capture the intuition stated earlier that if a test $K_i\varphi$ is false at a point (r, m), then there should be a point (r', m') with $m' \leq m$ that is a "witness" to its falsity. The next definitions do formalize this intuition. If $\mathcal{I}$ is an interpreted system and $K_i\varphi$ is a formula, then we say that $\mathcal{I}$ *provides witnesses for* $K_i\varphi$ if whenever (r, m) is a point of $\mathcal{I}$ such that $(\mathcal{I}, r, m) \models \neg K_i\varphi$, then there is some point (r', m') of $\mathcal{I}$ with $m' \leq m$ such that $r'_i(m') = r_i(m)$ and $(\mathcal{I}, r', m') \models \neg\varphi$. We say that $\mathcal{J}$ *provides witnesses for* Pg if $\mathcal{I}$ provides witnesses for $K_i\varphi$ for every interpreted system $\mathcal{I} \in \mathcal{J}$ and for every subformula $K_i\varphi$ of a test in Pg. Finally, we say that Pg is *atemporal* if all its tests are knowledge formulas (and so do not involve temporal operators).

Lemma 4.12 *If Pg is atemporal and $\mathcal{J}$ provides witnesses for Pg, then Pg depends on the past in $\mathcal{J}$.*

Proof A straightforward induction on the structure of formulas shows that all subformulas of tests in Pg depend on the past in $\mathcal{J}$. For primitive propositions this is immediate, since the truth of a primitive proposition is determined by the global state (given a fixed interpretation π). The case of conjunctions and negations follows immediately from the inductive hypothesis, and the case of epistemic formulas is immediate from the fact that there is always a witness. We leave details to the reader. ∎

Lemma 4.13 *Suppose Pg is atemporal. If $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ provides witnesses for Pg, then so does $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$.*

Proof For the purposes of this proof only, we now give some more definitions that simplify notation. If ρ is the m-prefix of a run r, let us define $\rho_i(m)$ to be $r_i(m)$, the state of process i at time m. Let ρ be the m-prefix of the run r of the interpreted system $\mathcal{I}$ and let φ be a knowledge formula. We define $(\mathcal{I}, \rho) \models \varphi$ to hold precisely if $(\mathcal{I}, r, m) \models \varphi$ holds. This is well-defined, since it is easy to show by induction on the structure of φ that if r' is another run of $\mathcal{I}$ with m-prefix ρ and φ is a knowledge formula, then $(\mathcal{I}, r, m) \models \varphi$ iff $(\mathcal{I}, r', m) \models \varphi$.

We prove that for every knowledge formula φ that is a subformula or the negation of a subformula of a test in Pg, the following properties hold:

(a) for every time m, every pair $\mathcal{I}, \mathcal{I}^1$ of systems in $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$ such that $\mathit{Pref}_m(\mathcal{I}) = \mathit{Pref}_m(\mathcal{I}^1)$, and every ρ that is an m-prefix of both a run of $\mathcal{I}$ and a run of $\mathcal{I}^1$, we have that $(\mathcal{I}, \rho) \models \varphi$ iff $(\mathcal{I}^1, \rho) \models \varphi$, and

(b) if φ is of the form $K_i\psi$, then every $\mathcal{I} \in \mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$ provides witnesses for φ.

We proceed by induction on the structure of formulas. The case of primitive propositions, conjunctions, and negations is straightforward. It remains to show the case of formulas of the form $K_i\psi$. By the symmetry of the roles of $\mathcal{I}$ and $\mathcal{I}^1$, to prove part (a) it is sufficient to show that if $(\mathcal{I}, \rho) \models \neg K_i\psi$ then $(\mathcal{I}^1, \rho) \models \neg K_i\psi$. Assume that $(\mathcal{I}, \rho) \models \neg K_i\psi$. So for some m', there is an m'-prefix ρ' of a run of $\mathcal{I}$ such that $\rho'_i(m') = \rho_i(m)$ and $(\mathcal{I}, \rho') \models \neg\psi$. Let $m^* = \max\{m, m'\}$. Find $\mathcal{I}^2 \in \mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ such that $\mathit{Pref}_{m^*}(\mathcal{I}^2) = \mathit{Pref}_{m^*}(\mathcal{I})$. Such a system $\mathcal{I}^2$ is guaranteed to exist, since every interpreted system in $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$ is the limit of a prefix-compatible sequence of members of $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$. In particular, ρ is the m-prefix of a run of $\mathcal{I}^2$ and ρ' is the m'-prefix of a run of $\mathcal{I}^2$. Since $(\mathcal{I}, \rho') \models \neg\psi$, it follows by the inductive hypothesis for ψ that $(\mathcal{I}^2, \rho') \models \neg\psi$. Since $\rho'_i(m') = \rho_i(m)$, it follows that $(\mathcal{I}^2, \rho) \models \neg K_i\psi$. Since $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ provides witnesses for Pg, there is some $m'' \leq m$ and an m''-prefix ρ'' of a run of $\mathcal{I}^2$ such that $\rho''_i(m'') = \rho_i(m)$ and $(\mathcal{I}^2, \rho'') \models \neg\psi$. Now $\mathit{Pref}_{m''}(\mathcal{I}^1) = \mathit{Pref}_{m''}(\mathcal{I}) = \mathit{Pref}_{m''}(\mathcal{I}^2)$. So ρ'' is an m''-prefix of a run of $\mathcal{I}^1$. Therefore, by the inductive hypothesis for ψ, it follows that $(\mathcal{I}^1, \rho'') \models \neg\psi$. Since $\rho''_i(m'') = \rho_i(m)$, it follows that $(\mathcal{I}^1, \rho) \models \neg K_i\psi$, as desired. So part (a) holds for φ. As for part (b), we see that since ρ'' is also an m''-prefix of a run of $\mathcal{I}$, it follows from the inductive hypothesis that $(\mathcal{I}, \rho'') \models \neg\psi$. So $\mathcal{I}$ provides witnesses for φ. Therefore, (b) holds for φ. This concludes the inductive step. The result stated in the lemma now follows from part (b). ∎

We define a system $\mathcal{I}$ to be *synchronous* if for every agent i and points $(r, m), (r', m') \in \mathcal{I}$, we have that $(r, m) \sim_i (r', m')$ implies $m = m'$. Intuitively, a system is synchronous if an agent can determine the time by looking at his local state.

Lemma 4.14 *If every system in $\mathcal{J}$ is synchronous, then $\mathcal{J}$ provides witnesses for* Pg.

Proof This follows directly from the definitions. Suppose $\mathcal{I} \in \mathcal{J}$ and $(\mathcal{I}, r, m) \models \neg K_i\varphi$. By definition of $\models$, there must be a point (r', m') in $\mathcal{I}$ satisfying both $r'_i(m') = r_i(m)$ and $(\mathcal{I}, r', m') \models \neg\varphi$. Since $\mathcal{I}$ is synchronous, $r'_i(m') = r_i(m)$ implies that $m' = m$ and, in particular, we have that $m' \leq m$. It now follows that $\mathcal{J}$ provides witnesses for Pg (the point (r', m) is the desired "witness" to $\neg\varphi$). ∎

For many programs Pg and interpreted contexts (γ, π) of interest, every system in $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ is indeed synchronous. In particular, if Pg prescribes that each agent performs some action in every round (more precisely, if each agent performs an action in every round of $\mathsf{Pg}^{\mathcal{I}}$, regardless of the choice of $\mathcal{I}$) and if the agents keep track of the actions they have performed in their local states (as is the case in message-passing systems), then interpreted systems of the form $\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma, \pi)$ are necessarily synchronous, since an agent can determine the time by looking at his local state. It follows that $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ provides witnesses for Pg.

Putting the results above together, we can define a condition that guarantees a program has a unique representation and that applies to many contexts of interest. We say that an interpreted context (γ, π) *provides witnesses* for a knowledge-based program Pg exactly if $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ provides witnesses for Pg. As a straightforward corollary of Corollary 4.11, Lemma 4.12 and Lemma 4.13, we obtain the following result (which is precisely Theorem 7.2.4 of [FHMV95]).

Corollary 4.15 *Let* Pg *be an atemporal knowledge-based program and let* (γ, π) *be an interpreted context that provides witnesses for* Pg *such that* γ *is nonexcluding. Then there is a unique interpreted system* $\mathbf{I}^{rep}(\mathsf{Pg}, \gamma, \pi)$ *representing* Pg *in* (γ, π).

Proof The assumption that (γ, π) provides witnesses for Pg means, by definition, that $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ provides witnesses for Pg. Since, by assumption, Pg is atemporal, we have by Lemma 4.13 that $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$ provides witnesses for Pg. Lemma 4.12 now implies that Pg depends on the past in $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$. This, coupled with the fact that γ is assumed to be nonexcluding, gives us by Corollary 4.11 that there is a unique interpreted system representing Pg in (γ, π). ∎

The next corollary follows from Lemma 4.14 and Corollary 4.15.

Corollary 4.16 *Suppose that* Pg *is an atemporal knowledge-based program, that* γ *is a nonexcluding context, and that every system in* $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ *is synchronous. Then there is a unique interpreted system* $\mathbf{I}^{rep}(\mathsf{Pg}, \gamma, \pi)$ *representing* Pg *in* (γ, π).

Proof Since every system in $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ is synchronous, it follows from Lemma 4.14 that $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ provides witnesses for Pg, that is, (γ, π) provides witnesses for Pg. The result now follows from Corollary 4.15. ∎

The muddy children problem gives an application of Corollary 4.16.

Example 4.17 We now want to take a more careful look at the knowledge-based program MC run by the muddy children. We start by formally describing the context (γ^{mc}, π^{mc}) corresponding to our intuitive description of the muddy children puzzle. The agents here are the children and the father. We can view $\gamma^{mc} = (P_e^{mc}, \mathcal{G}_0, \tau, \mathit{True})$ as a context in which whatever an agent (the father or one of the children) says in a given round is represented as a message that is delivered in the same round to all other agents, and in which all these messages are recorded in the local states of the agents when they are received. The initial states of the agents describe what they see; later states describe everything they have heard. Thus, $\mathcal{G}_0$ consists of all 2^n tuples of the form $(\langle\rangle, X^{-1}, \ldots, X^{-n}, X)$, where $X = (x_1, \ldots, x_n)$ is a tuple of 0's and 1's, with $x_i = 0$ meaning that child i is clean, and $x_i = 1$ meaning that he has a muddy forehead, and

$X^{-i} = (x_1, \ldots, x_{i-1}, *, x_{i+1}, \ldots, x_n)$, i.e., it differs from X only in that it contains a $*$ in the ith component. Intuitively, X^{-i} describes what child i sees given that X describes the true situation, where $*$ means "no information". Only the father sees all the children, so his initial local state is X. The initial local state of the environment is the empty history $\langle\rangle$. The only actions performed by the children and the father are the sending of messages, and these actions have the obvious results of changing their local states and the local state of the environment. The environment's protocol P_e^{mc} is simply to deliver all messages in the same round in which they are sent.

The children run the knowledge-based programs MC_i described at the beginning of Section 3.2. The father runs the following (standard) program:

case of
    if $\mathit{initial} \wedge \bigvee_{i=1}^{n} p_i$ **do**
        say "At least one of you has mud on your forehead; does any
        of you know whether you have mud on your own forehead?"
    if $\mathit{initial} \wedge \neg \bigvee_{i=1}^{n} p_i$ **do**
        say "Does any of you know whether you have mud
        on your own forehead?"
    if *childrenanswered* **do**
        say "Does any of you know whether you have mud
        on your own forehead?"
end case.

Here *initial* is a primitive proposition that is true in the initial state, i.e., before any communication has taken place, and *childrenanswered* is a primitive proposition that is true if the father heard the children's answers in the previous round. Thus, in round 1, if there is at least one muddy child, a message to this effect is sent to all children. In the odd-numbered rounds 1, 3, 5, ..., the father sends to all children the message "Does any of you know whether you have mud on your own forehead?". The children respond "Yes" or "No" in the even-numbered rounds. Finally, π^{mc} interprets the propositions p_i, $\mathit{childheard}_i$, *initial*, and *childrenanswered* in the obvious way.

We now want to apply Corollary 4.16 to show that there is a unique interpreted system representing MC. Since the admissibility condition in γ^{mc} is *True*, it easily follows that γ^{mc} is nonexcluding. Clearly there are no temporal operators in the tests in MC. Moreover, notice that the father and the children each either send a message or receive one in every round, and they keep track of the messages they send and receive in their local states. Since an action is performed by each child at every round of $\mathsf{MC}^{\mathcal{I}}$, regardless of the choice of $\mathcal{I}$, as we observed in the discussion following Lemma 4.14, it follows that every interpreted system in $\mathcal{J}(\mathsf{MC}, \gamma^{mc}, \pi^{mc})$ is synchronous. Thus, by Corollary 4.16, there is a unique system representing MC in (γ^{mc}, π^{mc}).

The same arguments show that the hypotheses of Corollary 4.16 also hold for any subcontext $\gamma' \sqsubseteq \gamma^{mc}$ obtained by restricting the set of initial states, that is, by replacing $\mathcal{G}_0$ by some subset of $\mathcal{G}_0$. Restricting the set of initial states corresponds to changing the puzzle by making certain information common knowledge. For example, eliminating the initial states where child 3's forehead is clean corresponds to making it common knowledge that child 3's forehead is muddy.

As shown in [HM90], if the father initially says that at least one child has a muddy forehead, then a child that sees k muddy children responds "No" to the father's first k questions and "Yes" to the $(k+1)$st question (and to all the questions after that). Let MC_s be the standard program for the muddy children that has them doing this. Finally, let $\mathcal{I}^{mc} = \mathbf{I}^{rep}(\mathsf{MC}_s, \gamma^{mc})$. It is straightforward to show that $\mathcal{I}^{mc}$ represents MC in (γ^{mc}, π^{mc}), and hence, by our previous argument, is the unique such interpreted system. In fact, MC_s implements MC in (γ^{mc}, π^{mc}). There are, however, contexts in which MC_s does not implement MC. For example, consider the context where it is common knowledge that the children all have muddy foreheads. This is the subcontext $\gamma' \sqsubseteq \gamma^{mc}$ in which we replace $\mathcal{G}_0$ by the singleton set $\{(\langle\rangle, X^{-1}, \ldots, X^{-n}, X\})$, where $X = (1, \ldots, 1)$. We leave it to the reader to check that in the unique interpreted system $\mathcal{I}'$ representing MC in (γ', π^{mc}), all the children respond "Yes" to the father's first question. Clearly MC_s does not implement MC in this context. ∎

As is shown in [FHMV95], Corollary 4.11 (or its derivatives, Corollaries 4.15 and 4.16) can be used to show that a number of other knowledge-based programs have unique representations. For example, it applies to the knowledge-based programs used to analyze the sequence transmission problem [HZ92], Byzantine agreement [DM90, MT88], and to a program designed to capture a Teller giving information to a knowledge base [FHMV94]. On the other hand, there are times when we cannot apply Corollary 4.11, since Pg may fail to depend on the past with respect to $\mathcal{J}^+(\mathsf{Pg}, \gamma, \pi)$, although there may be another class $\mathcal{J}$ to which the hypotheses of Theorem 4.9 apply. Is there anything we can say then? That is the subject of the next section.

5 Testing for existence and uniqueness of representations

While the results of the previous section provide necessary and sufficient conditions to determine if a knowledge-based program has a unique representation, they are not always easy to apply. How hard is it to tell in general whether a knowledge-based program has a unique representation, or any representation at all, for that matter? Clearly the answer depends in part on the context in which the program is run, and how it is represented. In this section, we give a partial answer to that question by considering *finite-state interpreted contexts*.

A finite-state interpreted context is one in which the set of global states is finite, the set of possible actions is finite, the set of primitive propositions is finite, and the admissibility condition on runs is given by a temporal formula. We also assume that all the components of such a context are described in a "transparent" way, so that the environment's protocol is described as a set of (local state, action) pairs, the transition function is described as a set of (joint action, global state, global state) tuples, and the interpretation (of the primitive propositions) is described as a set of (primitive proposition, global state, truth value) tuples. The key is that we should be able to check in polynomial time whether, for example, $\pi(g)(p) = \mathbf{true}$ or $\tau(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)(g) = g'$. A *finite* knowledge-based program is one where the case statement involves only finitely many tests (which may include common knowledge operators and temporal operators). A joint knowledge-based program is finite if each of its components is. Thus, it makes sense to talk about the *size* of a finite-state interpreted context and of a finite knowledge-based program; it is the length of a description of the context or the program under any reasonable encoding. We will measure the complexity of testing existence and uniqueness as a function of the size of the given finite-state interpreted context and of the given finite knowledge-based program.

Our goal in this section is to study the complexity of determining whether a given finite knowledge-based program has some (resp., a unique) representation in a given finite-state interpreted context. It is not hard to show that the problem involves both *model checking* [CES86]—checking whether the tests in the program are true at certain states in a system whose global states are among the global states allowed by the context (γ, π)—and testing the satisfiability of the admissibility condition. Both model checking and satisfiability testing are known to be *PSPACE*-complete problems for (linear time) temporal logic [SC85], so our problem is at least *PSPACE*-hard. We show below (Theorem 5.10) that, in fact, it is no harder.

5.1 An Easier Case

Before proving the general result, we prove a simpler version: we consider *nonrestrictive (interpreted) contexts*, where the admissibility condition is *True*, and *atemporal* knowledge-based programs. With these restrictions, our arguments for *PSPACE*-hardness no longer apply: testing the satisfiability of the admissibility condition is now trivial, and the model checking problem for knowledge formulas can be solved in polynomial time. Indeed, as we now show, these restrictions do make the problems simpler: they drop from *PSPACE* to *NP*. We now develop the technical machinery required, and then proceed to state and prove the results.

5.1.1 Knowledge-Based Programs and Kripke Structures

Our first step is to show that atemporal knowledge-based programs can be interpreted with respect to Kripke structures. This will enable us to characterize the existence of representations for atemporal knowledge-based programs with respect to nonrestrictive interpreted contexts in terms of existence of certain Kripke structures. Let $\mathcal{F}$ be a set of global states and π be an interpretation for the propositions in Φ over $\mathcal{F}$. We define a *Kripke structure* $M_{\mathcal{F}} = (\mathcal{F}, \mathcal{K}_1, \ldots, \mathcal{K}_n, \pi)$, where each $\mathcal{K}_i$ is a binary relation on $\mathcal{F}$ such that $(g, g') \in \mathcal{K}_i$ iff $g_i = g'_i$, that is, if g and g' agree on their ith component. Truth of knowledge formulas in $M_{\mathcal{F}}$ can now be defined in the standard way (cf. [HM92]). In particular, we have

$$(M_{\mathcal{F}}, g) \models K_i\varphi \text{ iff } (M_{\mathcal{F}}, g') \models \varphi \text{ for all } g' \text{ such that } (g, g') \in \mathcal{K}_i$$
$$(M_{\mathcal{F}}, g) \models E\varphi \text{ iff } (M_{\mathcal{F}}, g) \models K_i\varphi \text{ for } i = 1, \ldots, n$$
$$(M_{\mathcal{F}}, g) \models C\varphi \text{ iff } (M_{\mathcal{F}}, g) \models E^k\varphi \text{ for } k = 1, 2, \ldots$$

Consider an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$, where $\mathcal{R}$ is a system over a set $\mathcal{G}$ of global states and π is an interpretation for the propositions in Φ over $\mathcal{G}$. We use both $\mathcal{F}_{\mathcal{I}}$ and $\mathcal{F}_{\mathcal{R}}$ to denote the global states that occur in $\mathcal{R}$, i.e., $\mathcal{F}_{\mathcal{I}} = \mathcal{F}_{\mathcal{R}} = \{r(m) \mid r \in \mathcal{R}\}$. It is easy to prove by induction on the structure of knowledge formulas that $M_{\mathcal{F}_{\mathcal{I}}}$ completely captures the semantics of knowledge formulas in $\mathcal{I}$.

Lemma 5.1 *Let φ be a knowledge formula. Then $(\mathcal{I}, r, m) \models \varphi$ iff $(M_{\mathcal{F}_{\mathcal{I}}}, r(m)) \models \varphi$.*

Given a set $\mathcal{F}$ of global states, we can associate with an atemporal knowledge-based program Pg_i for agent i a protocol $\mathsf{Pg}_i^{\mathcal{F}}$ in much the same way we used an interpreted system $\mathcal{I}$ to obtain the protocol $\mathsf{Pg}_i^{\mathcal{I}}$. We start, as in Section 3.2, by defining truth of tests in local states. We do this by overloading notation and defining yet another satisfaction notion, where on the left-hand side of $\models$ we have a pair $(M_{\mathcal{F}}, \ell)$ consisting of a the Kripke structure $M_{\mathcal{F}}$ and a local state ℓ for agent i.

If φ is a standard test in Pg_i, we define

$(M_{\mathcal{F}}, \ell) \models \varphi$ iff $(\pi, \ell) \models \varphi$.

Since φ is a standard test in Pg_i, it must be local to agent i, so this definition makes sense. If φ is a knowledge test $K_i\psi$, we define

$(M_{\mathcal{F}}, \ell) \models K_i\psi$ iff $(M_{\mathcal{F}}, g) \models \psi$ for all global states $g \in \mathcal{F}$ such that $g_i = \ell$.

Finally, for conjunctions and negations, we follow the standard treatment.

We can now define

$$\mathsf{Pg}_i^{\mathcal{F}}(\ell) = \begin{cases} \{\mathsf{a}_j : (M_{\mathcal{F}}, \ell) \models t_j \wedge k_j\} & \text{if } \{j : (M_{\mathcal{F}}, \ell) \models t_j \wedge k_j\} \neq \emptyset \\ \{\Lambda\} & \text{if } \{j : (M_{\mathcal{F}}, \ell) \models t_j \wedge k_j\} = \emptyset. \end{cases}$$

Intuitively, the actions prescribed by i's protocol $\mathsf{Pg}_i^{\mathcal{F}}$ are exactly those prescribed by Pg_i when the tests are evaluated in $M_{\mathcal{F}}$.

Lemma 5.2 *Let Pg be an atemporal knowledge-based program, let (γ, π) be an interpreted context, and let $\mathcal{I} = (\mathcal{R}, \pi)$ be an interpreted system. Then $\mathsf{Pg}^{\mathcal{I}} = \mathsf{Pg}^{\mathcal{F}_{\mathcal{R}}}$.*

Proof Let $\mathcal{F} = \mathcal{F}_{\mathcal{R}}$. We have to show that for every local or knowledge test φ we have that $(\mathcal{I}, \ell) \models \varphi$ iff $(M_{\mathcal{F}}, \ell) \models \varphi$. For a standard test φ this holds, since $(\mathcal{I}, \ell) \models \varphi$ iff $(\pi, \ell) \models \varphi$ iff $(M_{\mathcal{F}}, \ell) \models \varphi$. For a knowledge test $K_i\psi$, we have that $(\mathcal{I}, \ell) \models K_i\psi$ iff $(\mathcal{I}, r, m) \models \psi$ for all points (r, m) of $\mathcal{I}$ such that $r_i(m) = \ell$. By Lemma 5.1, the latter holds iff $(M_{\mathcal{F}}, g) \models \psi$ for all global states g in $\mathcal{F}$ such that $g_i = \ell$. This holds iff $(M_{\mathcal{F}}, \ell) \models K_i\psi$. (Note that Lemma 5.1 applies only to knowledge formulas, which is why we need to assume that Pg is an atemporal knowledge-based program.) ∎

5.1.2 Testing Existence of Representations

We now provide a characterization for when there is a system representing an atemporal knowledge-based program with respect to a nonrestrictive interpreted context.

Proposition 5.3 *Let Pg be an atemporal knowledge-based program and let (γ, π) be a nonrestrictive interpreted context. There is an interpreted system that represents Pg in (γ, π) iff there is a set $\mathcal{F}$ of global states such that $\mathcal{F} = \mathcal{F}_{\mathcal{R}}$, where $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{F}}, \gamma)$, i.e, $\mathcal{F}$ is precisely the set of states that occur in the system that represents $\mathsf{Pg}^{\mathcal{F}}$ in γ. Furthermore, there is a unique interpreted system that represents Pg in (γ, π) iff there is unique such $\mathcal{F}$.*

Proof Suppose first that that there is an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ that represents Pg in (γ, π), i.e., $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma)$. Let $\mathcal{R}'$ be $\mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{F}_{\mathcal{R}}}, \gamma)$. It follows from Lemma 5.2 that $\mathcal{R}' = \mathcal{R}$.

Conversely, suppose that there is a subset $\mathcal{F}$ of global states such that $\mathcal{F} = \mathcal{F}_{\mathcal{R}}$, where $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{F}}, \gamma)$. We claim that $\mathcal{I} = (\mathcal{R}, \pi)$ represents Pg in (γ, π). This holds since, as before, $\mathsf{Pg}^{\mathcal{I}} = \mathsf{Pg}^{\mathcal{F}}$.

Finally, if there are two different such sets, say $\mathcal{F}_1 \neq \mathcal{F}_2$, then we get two systems, $\mathcal{R}_1 = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{F}_1}, \gamma)$ and $\mathcal{R}_2 = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{F}_2}, \gamma)$, that represent Pg in context γ. These systems are different, since $\mathcal{F}_1 = \mathcal{F}_{\mathcal{R}_1}$ and $\mathcal{F}_2 = \mathcal{F}_{\mathcal{R}_2}$. ∎

We can now obtain the desired complexity results for nonrestrictive finite-state interpreted contexts and atemporal finite knowledge-based programs.

Theorem 5.4 *Testing whether there is at least one (resp., more than one) interpreted system representing a given atemporal finite knowledge-based program in a given nonrestrictive finite-state interpreted context is NP-complete.*

Proof We first show that the problem is in *NP*. Let Pg be an atemporal finite knowledge-based program and let (γ, π) be a nonrestrictive finite-state interpreted context, where $\gamma = (P_e, \mathcal{G}_0, \tau, \mathit{True})$. By Proposition 5.3, there is an interpreted system that represents Pg in (γ, π) iff there is a subset $\mathcal{F}$ of global states such that $\mathcal{F} = \mathcal{F}_{\mathcal{R}}$, where $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{F}}, \gamma)$.

To check that there is at least one interpreted system that represents Pg in (γ, π), our algorithm guesses a subset $\mathcal{F}$ of global states and checks that it satisfies the condition of Proposition 5.3. To that end, we need to compute the set $\mathcal{F}' = \mathcal{F}_{\mathcal{R}}$, where $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{F}}, \gamma)$. $\mathcal{F}'$ is easily seen to be the least set containing $\mathcal{G}_0$ that is closed under the operation of the protocol $\mathsf{Pg}^{\mathcal{F}}$. That is, if $g \in \mathcal{F}'$, then so is every global state of the form $\tau(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)(g)$, where $\mathsf{a}_e \in P_e(g)$ and $\mathsf{a}_i \in \mathsf{Pg}_i^{\mathcal{F}}(g_i)$. Thus, to compute it, we start with the set $\mathcal{G}_0$ of initial states and keep applying the operation of the protocol $\mathsf{Pg}^{\mathcal{F}}$ until no new global states are added. To compute $\mathsf{Pg}_i^{\mathcal{F}}(g_i)$ we need to evaluate the truth of knowledge tests of Pg_i in $M_{\mathcal{F}}$, but this can be done in polynomial time in the size of $\mathcal{F}$ and the size of the knowledge tests [HM92]. Thus, checking that $\mathcal{F} = \mathcal{F}'$ can be done in polynomial time.

To check that there is more than one interpreted system that represents Pg in (γ, π), the algorithm simply guesses two sets $\mathcal{F}_1$ and $\mathcal{F}_2$ and checks that they both satisfy the condition and that they are different. This can clearly be done in nondeterministic polynomial time.

It remains to show that testing whether there is at least one (resp., more than one) interpreted system representing a given atemporal finite knowledge-based program in a given nonrestrictive finite-state interpreted context is *NP*-hard. The proof is by reduction from the satisfiability problem [GJ79].

Suppose we are given a propositional formula ξ over the primitive propositions $p_1, \ldots, p_n$. Without loss of generality, we can assume that if ξ is satisfiable then it has more than one satisfying assignment. (This can be ensured by adding one primitive proposition that does not appear in ξ to the language. Since this proposition can be assigned two truth values, if ξ is satisfiable then it has at least two truth assignments.) We now describe a nonrestrictive finite-state interpreted context (γ, π) and an atemporal finite knowledge-based program Pg such that the following are equivalent:

- ξ is satisfiable
- there is at least one interpreted system that represents Pg in (γ, π),
- there is more than one interpreted system that represents Pg in (γ, π),

The environment can be in any of the states $\{0, 1, \ldots, n\}$, where 0 is the initial state. There is only one agent in the context γ, who is always in the same fixed local state. Thus, we can identify the global state with the environment's state. The set of primitive propositions is $\Phi = \{p_0, p_1, \ldots, p_n\}$. Note that Φ contains a primitive proposition p_0 that is not in ξ. For $p_j \in \Phi$, we define $\pi(i)(p_j) = \mathbf{true}$ iff $i = j$, i.e., p_i holds only in the state i. The set of the agent's actions is $\{\mathsf{a}_1, \ldots \mathsf{a}_n\}$, but the environment can perform only a single action. Thus, we can identify a joint action with the agent's action. Finally, we have that $\tau(\mathsf{a}_i)(j) = i$, independent of j, i.e., the action a_i always moves the system to the state i. This concludes the definition of (γ, π).

Let φ be the knowledge formula obtained from ξ by replacing each occurrence of p_i by the formula $\neg K \neg p_i$, for $1 \le i \le n$. (Since there is only one agent, we do not need to index the knowledge modalities.) Note that φ is a knowledge test. Let Pg be the following program:

case of
    if $Kp_0 \wedge \neg\varphi$ **do** a_1
    if φ **do** a_1
    ...
    if φ **do** a_n
end case.

Assume that $\mathcal{I} = (\mathcal{R}, \pi)$ is an interpreted system that represents Pg in (γ, π), that is, $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma)$, and assume that $r \in \mathcal{R}$. We claim that ξ is satisfied by the truth assignment that makes p_i true precisely when $i \in \mathcal{F}_{\mathcal{R}}$, for $1 \le i \le n$. Since 0 is the initial state, we know that $0 \in \mathcal{F}_{\mathcal{R}}$. Suppose first that $\mathcal{F}_{\mathcal{R}} = \{0\}$. That can happen if no action a_i is ever selected by $\mathsf{Pg}^{\mathcal{I}}$, so the only action selected by $\mathsf{Pg}^{\mathcal{I}}$ is Λ. But $(\mathcal{I}, r, 0) \models Kp_0$, since $\mathcal{F}_{\mathcal{R}} = \{0\}$, so we must have that $(\mathcal{I}, 0) \models \varphi$ (otherwise a_1 is selected by the first clause of Pg, which contradicts our earlier point that only Λ is selected). Since, however, $(\mathcal{I}, r, 0) \models K\neg p_i$ for $1 \le i \le n$ (because $\mathcal{F}_{\mathcal{R}} = \{0\}$), this means that ξ is satisfied by the truth assignment that sets $p_1, \ldots, p_n$ to **false**. Now suppose that $\{0\}$ is a proper subset of $\mathcal{F}_{\mathcal{R}}$. This means that some action a_i must be selected by $\mathsf{Pg}^{\mathcal{I}}$. It follows that $(\mathcal{I}, r, 0) \not\models Kp_0$, which means that the first clause of Pg cannot be selected. For any other clause to be selected, we must have $(\mathcal{I}, r, 0) \models \varphi$. Since $(\mathcal{I}, r, 0) \models \neg K \neg p_i$ iff $i \in \mathcal{F}$, it follows that ξ is satisfied by the truth assignment that makes p_i true precisely when $i \in \mathcal{F}_{\mathcal{R}}$, for $1 \le i \le n$.

Now suppose that ξ is satisfied by a truth assignment χ. Let $\mathcal{F} = \{i \mid \chi(p_i) = \mathbf{true}\} \cup \{0\}$. It is easy to see that $(M_{\mathcal{F}}, j) \models \neg K \neg p_i$ iff $i \in \mathcal{F}$, for $1 \le i \le n$. Thus, $\mathsf{Pg}^{\mathcal{F}}(j) = \{\mathsf{a}_i \mid i \in \mathcal{F} - \{0\}\}$, for each state j. Since $\tau(\mathsf{a}_i)(j) = i$, it follows that $\mathcal{F} = \mathcal{F}_{\mathcal{R}}$, where $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{F}}, \gamma)$, so $(\mathcal{R}, \pi)$ represents Pg in (γ, π), by Proposition 5.3. Note that since ξ is satisfied by more than one truth assignment, Pg is represented by more than one interpreted system. ∎

Theorem 5.4 tells us that testing whether there is at least one or more than one interpreted system representing a given atemporal finite knowledge-based program in a given nonrestrictive finite-state interpreted context is *NP*-complete. How can we test whether there is a *unique* interpreted system representing a given atemporal finite knowledge-based program in a given nonrestrictive finite-state interpreted context? We have to test that the program is represented by at least one interpreted system and that it is not represented by more than one interpreted system. Thus, this test is the *difference* between two *NP* tests. Problems that can be decided by the difference between two *NP* tests are classified in the complexity class D^p [PY82]. Formally, D^p is the class P of problems (i.e., languages) such that $P = P_1 - P_2$, where both P_1 and P_2 are in *NP*. All problems in *NP* and *co-NP* are easily seen to be in D^p; thus, unless *NP* = *co-NP*, the class D^p is strictly larger than either *NP* or *co-NP*. The UNIQUE-SAT problem is the problem of deciding whether a given propositional formula has a unique satisfying assignment. It not hard to show that UNIQUE-SAT is in D^p. It is shown in [JVV86] that, in fact, UNIQUE-SAT is complete for D^p under *randomized reductions*. This means that for every problem $A \in D^p$, there is a random polynomial-time function f (that is, the output of f on input x, denoted $f(x)$, may depend on some coin tosses) and a polynomial p such that

- if $x \notin A$, then $f(x) \notin$ UNIQUE-SAT with probability 1 (that is, whatever the output of f on input x is, it is not in UNIQUE-SAT), and
- if $x \in A$, then $f(x) \in$ UNIQUE-SAT with probability at least $1/p(|x|)$.

Theorem 5.5 *Testing whether there is a unique interpreted system representing a given atemporal finite knowledge-based program in a given nonrestrictive finite-state interpreted system is polynomially equivalent to the UNIQUE-SAT problem.*

Proof We first show that UNIQUE-SAT is polynomially reducible to the unique representation problem. The proof is almost identical to the lower-bound proof in Theorem 5.4. (Unlike the proof in Theorem 5.4, we do not force ξ to have at least two satisfying truth assignments when it has at least one satisfying truth assignment.) It is easy to see there that Pg is represented by a unique interpreted system in (γ, π) iff ξ has a unique satisfying truth assignment.

We now show that the unique representation problem is polynomially reducible to UNIQUE-SAT. The algorithm in Theorem 5.4 guesses a set $\mathcal{F}$ of global states and then verifies in polynomial time that $\mathcal{F} = \mathcal{F}_{\mathcal{R}}$, where $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{F}}, \gamma)$. Uniqueness of the representation means that there is a unique such $\mathcal{F}$. Clearly, this algorithm can be implemented by a deterministic polynomial time Turing machine M equipped with a "guessing" tape. The standard reduction of M to the satisfiability problem [GJ79] reduces the unique representation problem to UNIQUE-SAT. ∎

Corollary 5.6 *Testing whether there is a unique interpreted system representing a given atemporal finite knowledge-based program in a given nonrestrictive finite-state interpreted system is complete for D^p under randomized reductions.*

5.2 The General Case

The problem is considerably more involved for general contexts and programs, where we allow temporal connectives. To understand the issues involved, we focus attention first on programs that do not mention the knowledge modalities E and C (although they may have temporal modalities and arbitrary nestings of K_i's). The first difficulty stems from the fact that we can no longer collapse an interpreted system $\mathcal{I}$ to the Kripke structure $M_{\mathcal{F}_\mathcal{I}}$, while still preserving the relevant semantic information as in Lemma 5.1. $M_{\mathcal{F}_\mathcal{I}}$ preserves the semantics of knowledge, but does not preserve the temporal semantics. Since the knowledge tests in Pg may involve temporal operators, we cannot simply consider $\mathsf{Pg}^{\mathcal{F}_\mathcal{I}}$ instead of $\mathsf{Pg}^{\mathcal{I}}$.

5.2.1 Knowledge-Based Programs and Knowledge Interpretations

We deal with this problem by considering *knowledge interpretations*, which tell us how to interpret knowledge tests in local states. Given a context γ in which L_i is the set of local states of agent i, for $i = 1, \ldots, n$, let $L_\gamma = L_1 \cup \ldots \cup L_n$. Let Pg be a knowledge-based program. Define $test(\mathsf{Pg})$ to be the set of subformulas of tests in Pg and their negations (we identify a formula $\neg\neg\psi$ with ψ). For each $i = 1, \ldots, n$, a knowledge interpretation κ for Pg in γ assigns to every local state $\ell \in L_i$ and every formula $K_i\psi \in test(\mathsf{Pg})$ a truth value, i.e., $\kappa(\ell, K_i\psi) = \mathbf{true}$ or $\kappa(\ell, K_i\psi) = \mathbf{false}$.

Now consider a knowledge-based program Pg_i for agent i. Instead of using an interpreted system $\mathcal{I}$ to obtain a protocol $\mathsf{Pg}_i^{\mathcal{I}}$, we can associate a protocol $\mathsf{Pg}_i^{\kappa,\pi}$ with Pg_i with a knowledge interpretation κ and an interpretation π that is compatible with Pg_i. If φ is a standard test, we define

$(\kappa, \pi, \ell) \models \varphi$ iff $(\pi, \ell) \models \varphi$.

If φ is a knowledge test $K_i\psi$, we define

$(\kappa, \pi, \ell) \models K_i\psi$ iff $\kappa(\ell, K_i\psi) = \mathbf{true}$.

Finally, for conjunctions and negations, we follow the standard treatment.

We now define

$$\mathsf{Pg}_i^{\kappa,\pi}(\ell) = \begin{cases} \{\mathsf{a}_j : (\kappa, \pi, \ell) \models t_j \wedge k_j\} & \text{if } \{j : (\kappa, \pi, \ell) \models t_j \wedge k_j\} \neq \emptyset \\ \{\Lambda\} & \text{if } \{j : (\kappa, \pi, \ell) \models t_j \wedge k_j\} = \emptyset. \end{cases}$$

In addition to the notion of knowledge interpretation, we also need the notion of *annotated states*, which are global states tagged with sets of formulas. Let g be a global state and let Θ be a subset of $test(\mathsf{Pg})$. The pair (g, Θ) is called an *annotated state.*

A set $\Theta \subseteq test(\mathsf{Pg})$ is *full* if the following holds:

1. For each $\varphi \in test(\mathsf{Pg})$, we have that $\varphi \in \Theta$ iff $\neg\varphi \notin \Theta$.

2. For each $\varphi_1 \wedge \varphi_2 \in test(\mathsf{Pg})$, we have that $\varphi_1 \wedge \varphi_2 \in \Theta$ iff $\varphi_1 \in \Theta$ and $\varphi_2 \in \Theta$.

An annotated state (g, Θ) is *consistent* with a knowledge interpretation κ and an interpretation π if (a) Θ is full, (b) for each proposition $p \in \Phi$ we have that $p \in \Theta$ iff $\pi(g)(p) = \mathbf{true}$, and (c) for each formula $K_i\psi \in test(\mathsf{Pg})$ we have that $K_i\varphi \in \Theta$ iff $\kappa(g_i, K_i\psi) = \mathbf{true}$. These conditions say that the annotations capture the standard semantics of propositions, of Boolean connectives, and of knowledge modalities. On the other hand, no constraint is imposed on the semantics of temporal operators.

To deal with the semantics of temporal operators we have to introduce the notion of *annotated runs*. An annotated run α over a set $\mathcal{F}$ of annotated states is a function from time to annotated states in $\mathcal{F}$ that satisfies the following condition: if $\alpha = (g^0, \Theta^0), (g^1, \Theta^1), \ldots$, then for each formula $\bigcirc\varphi \in \mathit{test}(\mathsf{Pg})$ or $\varphi U \psi \in \mathit{test}(\mathsf{Pg})$ we have:

1. $\bigcirc\varphi \in \Theta^m$ iff $\varphi \in \Theta^{m+1}$

2. $\varphi U \psi \in \Theta^m$ iff $\psi \in \Theta^{m'}$ for some $m' \geq m$ and $\varphi \in \Theta^{m''}$ for all m'' such that $m \leq m'' < m'$.

Thus, annotated runs have to display the "proper" temporal behavior. Given an annoted run $\alpha = (g^0, \Theta^0), (g^1, \Theta^1), \ldots,$ let $\mathit{run}(\alpha)$ be the run $g^0, g^1, \ldots$ that is obtained by deleting the annotations in α. An annotated run α is *consistent* with (κ, π) if every annotated state in α is consistent with (κ, π). An annotated run α is *consistent* with a joint protocol P in a context γ if $\mathit{run}(\alpha)$ is consistent with P in γ.

We can now state a condition for existence for representations. We say that the knowledge interpretation κ is *compatible with* Pg *in interpreted context* (γ, π) if, for each local state $\ell \in L_i$ and each formula $K_i\psi \in \mathit{test}(\mathsf{Pg})$, we have $\kappa(\ell, K_i\psi) = \mathbf{false}$ iff there is an annotated state (g, Θ) such that

- $g_i = \ell$ and $\neg\psi \in \Theta$, and
- (g, Θ) occurs in an annotated run that is consistent both with (κ, π) and with $\mathsf{Pg}^{\kappa,\pi}$ in the context γ.

Proposition 5.7 *Let* Pg *be a knowledge-based program and let* (γ, π) *be an interpreted context. There is an interpreted system that represents* Pg *in* (γ, π) *iff there is a knowledge interpretation* κ *that is compatible with* Pg *in* (γ, π)*. Moreover, there is more than one interpreted system that represents* Pg *in* (γ, π) *iff there are two knowledge interpretations* κ_1 *and* κ_2 *compatible with* Pg *in* (γ, π) *such that* $\mathbf{R}^{rep}(\mathsf{Pg}^{\kappa_1,\pi}, \gamma) \neq \mathbf{R}^{rep}(\mathsf{Pg}^{\kappa_2,\pi}, \gamma)$.

Proof First suppose that that there is an interpreted system $\mathcal{I} = (\mathcal{R}, \pi)$ that represents Pg in (γ, π), i.e., $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma)$. For $\ell \in L_i$, define $\kappa(\ell, K_i\psi) = \mathbf{true}$ iff $(\mathcal{I}, \ell) \models K_i\psi$. By the definition of $\mathsf{Pg}^{\kappa,\pi}$, we have that $\mathsf{Pg}^{\mathcal{I}}$ and $\mathsf{Pg}^{\kappa,\pi}$ are identical. We now show that κ is compatible with Pg in (γ, π):

$\kappa(\ell, K_i\psi) = \mathbf{false}$
iff $(I, \ell) \not\models K_i\psi$
iff $(\mathcal{I}, r, m) \models \neg\psi$ for some point (r, m) of $\mathcal{I}$ such that $r_i(m) = \ell$
iff there is an annotated state (g, Θ) that occurs in an annotated run that is consistent both with (κ, π) and with $\mathsf{Pg}^{\kappa,\pi}$ in context γ such that $g_i = \ell$ and $\neg\psi \in \Theta$.

We have to prove the last equivalence. The direction from left to right is easy: Assume that $r \in \mathcal{R}$. Define an annoted run $\alpha = (g^0, \Theta^0), (g^1, \Theta^1), \ldots$, where $g^m = r(m)$ and $\Theta^m = \{\varphi \in \mathit{test}(\mathsf{Pg}) \mid (\mathcal{I}, r, m) \models \varphi\}$. It is easy to verify that α is consistent both with (κ, π) and with $\mathsf{Pg}^{\mathcal{I}}$ in γ:

- Θ^m is full, since for each $\varphi \in \mathit{test}(\mathsf{Pg})$, we have that $(\mathcal{I}, r, m) \models \varphi$ iff $(\mathcal{I}, r, m) \not\models \neg\varphi$, and for each $\varphi_1 \wedge \varphi_2 \in \mathit{test}(\mathsf{Pg})$, we have that $(\mathcal{I}, r, m) \models \varphi_1 \wedge \varphi_2$ iff $(\mathcal{I}, r, m) \models \varphi_1$ and $(\mathcal{I}, r, m) \models \varphi_2$.
- For each proposition $p \in \Phi$, we have that $(\mathcal{I}, r, m) \models p$ iff $\pi(r(m))(p) = \mathbf{true}$.
- For each formula $K_i\psi \in \mathit{test}(\mathsf{Pg})$, we have that $(\mathcal{I}, r, m) \models K_i\psi$ iff $(\mathcal{I}, r_i(m)) \models K_i\psi$ iff $\kappa(r_i(m), K_i\psi) = \mathbf{true}$.
- $\mathit{run}(\alpha) = r$, $r \in \mathcal{R}$, and $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma)$.

Thus, if $(I, r, m) \models \neg\psi$ and $r_i(m) = \ell$, then $\neg\psi \in \Theta^m$ and $g_i^m = \ell$. For the direction from right to left, let $\alpha = (g^0, \Theta^0), (g^1, \Theta^1), \ldots$ be an annotated run that is consistent both with (κ, π) and with $\mathsf{Pg}^{\kappa,\pi}$ in γ, and let $r = \mathit{run}(\alpha)$. Since $\mathsf{Pg}^{\kappa,\pi} = \mathsf{Pg}^{\mathcal{I}}$, it follows that r is consistent with $\mathsf{Pg}^{\mathcal{I}}$ in γ, and so $r \in \mathcal{R}$. We claim that $\varphi \in \Theta^m$ iff $(\mathcal{I}, r, m) \models \varphi$, for each $m \geq 0$ and $\varphi \in \mathit{test}(\mathsf{Pg})$. The proof is by induction on the structure of formulas in $\mathit{test}(\mathsf{Pg})$.

1. For a proposition $p \in \Phi$, we have that $p \in \Theta^m$ iff $\pi(g^m)(p) = \mathbf{true}$ iff $(\mathcal{I}, r, m) \models p$.
2. For a formula $\neg\varphi \in \mathit{test}(\mathsf{Pg})$, we have that $\neg\varphi \in \Theta^m$ iff $\varphi \notin \Theta^m$ iff $(\mathcal{I}, r, m) \not\models \varphi$ iff $(\mathcal{I}, r, m) \models \neg\varphi$.
3. For a formula $\varphi_1 \wedge \varphi_2 \in \mathit{test}(\mathsf{Pg})$, we have that $\varphi_1 \wedge \varphi_2 \in \Theta^m$ iff $\varphi_1 \in \Theta^m$ and $\varphi_2 \in \Theta^m$ iff $(\mathcal{I}, r, m) \models \varphi_1$ and $(\mathcal{I}, r, m) \models \varphi_2$ iff $(\mathcal{I}, r, m) \models \varphi_1 \wedge \varphi_2$.
4. For a formula $K_i\psi \in \mathit{test}(\mathsf{Pg})$, we have that $K_i\psi \in \Theta^m$ iff $\kappa(\ell, K_i\psi) = \mathbf{true}$ for $\ell = g_i^m$ iff $(\mathcal{I}, \ell) \models K_i\psi$ iff $(\mathcal{I}, r', m') \models \psi$ for every point (r', m') such that $r'_i(m') = \ell$ iff $(I, r, m) \models K_i\psi$.
5. For a formula $\bigcirc\varphi \in \mathit{test}(\mathsf{Pg})$, we have that $\bigcirc\varphi \in \Theta^m$ iff $\varphi \in \Theta^{m+1}$ iff $(\mathcal{I}, r, m+1) \models \varphi$ iff $(\mathcal{I}, r, m) \models \bigcirc\varphi$.
6. For a formula $\varphi U \psi \in \mathit{test}(\mathsf{Pg})$, we have that $\varphi U \psi \in \Theta^m$ iff $\psi \in \Theta^{m'}$ for some $m' \geq m$ and $\varphi \in \Theta^{m''}$ for all m'' such that $m \leq m'' < m'$ iff $(\mathcal{I}, r, m') \models \psi$ for some $m' \geq m$ and $(\mathcal{I}, r, m'') \models \varphi$ for all m'' such that $m \leq m'' < m'$ iff $(\mathcal{I}, r, m) \models \varphi U \psi$.

Thus, if $\neg\psi \in \Theta^m$, then $(\mathcal{I}, r, m) \models \neg\psi$. If we also have $g_i^m = \ell$, then $r_i(m) = \ell$. This proves the desired equivalence.

Now suppose that we have a knowledge interpretation κ that is compatible with Pg in (γ, π). Let $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\kappa,\pi}, \gamma)$ and $\mathcal{I} = (\mathcal{R}, \pi)$. We claim that $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma)$, so $\mathcal{I}$ represents Pg in (γ, π). To prove that, it suffices to show that $\mathsf{Pg}^{\kappa,\pi}$ and $\mathsf{Pg}^{\mathcal{I}}$ coincide. Thus, we have to show that for every test φ of Pg and local state ℓ, we have that $(\kappa, \pi, \ell) \models \varphi$ iff $(\mathcal{I}, \ell) \models \varphi$. Satisfaction of standard tests depend only on π, so all we have to show is that for every formula $K_i\psi \in \mathit{test}(\mathsf{Pg})$ we have that $(\kappa, \pi, \ell) \models K_i\psi$ iff $(\mathcal{I}, \ell) \models K_i\psi$. By definition, $(\mathcal{I}, \ell) \not\models K_i\psi$ iff $(\mathcal{I}, r, m) \not\models \psi$ for some point (r, m) of $\mathcal{I}$ such that $r_i(m) = \ell$. By assumption, $(\kappa, \pi, \ell) \not\models K_i\psi$ iff there is an annotated state (g, Θ) that occurs in an annotated run that is consistent both with (κ, π) and with $\mathsf{Pg}^{\kappa,\pi}$ in context γ such that $g_i = \ell$ and $\neg\psi \in \Theta$. Thus, much like above, it suffices to show that if $\alpha = (g^0, \Theta^0), (g^1, \Theta^1), \ldots$ is an annotated run that is consistent both with (κ, π) and with $\mathsf{Pg}^{\kappa,\pi}$ in γ, if $m \geq 0$ and if $\varphi \in \mathit{test}(\mathsf{Pg})$, then $\varphi \in \Theta^m$ iff $(\mathcal{I}, r, m) \models \varphi$ for $r = \mathit{run}(\alpha)$. The proof is by induction on the structure of formulas in $\mathit{test}(\mathsf{Pg})$. The argument for the various cases of the induction are identical to (1)–(6) above, except for case (4), for a formula $K_i\psi \in \mathit{test}(\mathsf{Pg})$. In this case we proceed as follows:

4′. For a formula $K_i\psi \in \mathit{test}(\mathsf{Pg})$ we have that
$K_i\psi \in \Theta^m$
iff $\kappa(\ell, K_i\psi) = \mathbf{true}$ for $\ell = g_i^m$
iff for every annotated state (g, Θ) that occurs in an annotated run that is consistent both with (κ, π) and with $\mathsf{Pg}^{\kappa,\pi}$ in γ such that $g_i = g_i^m$, we have that $\psi \in \Theta$ iff $(\mathcal{I}, r', m') \models \psi$ for every point (r', m') such that $r'_i(m') = g_i^m$
iff $(\mathcal{I}, r, m) \models K_i\psi$.

The condition about existence of more than one interpreted systems that represents Pg in (γ, π) follows immediately from the correspondence between interpreted systems that represent Pg in (γ, π) and knowledge interpretations that meet the condition of the proposition. ∎

5.2.2 Testing Existence of Representations

We can now obtain the desired complexity results for finite-state interpreted contexts and finite knowledge-based programs. The algorithm is based on Proposition 5.7. The difficult part is in checking that a knowledge interpretation is compatible with Pg in (γ, π). For this we use the automata-theoretic techniques of [VW86].

A Büchi automaton A is a tuple $(\Sigma, S, S^0, \rho, F)$, where Σ is a finite nonempty *alphabet*, S is a finite nonempty set of *states*, $S^0 \subseteq S$ is a nonempty set of *initial* states, $F \subseteq S$ is the set of *accepting* states, and $\rho : S \times \Sigma \rightarrow 2^S$ is a *transition function*. Now suppose that A is given as input an infinite word $w = a_0, a_1, \ldots \in \Sigma^\omega$. A *run* r of A on w is a sequence $s_0, s_1, \ldots$ of states, where $s_0 \in S^0$ and $s_{i+1} \in \rho(s_i, a_i)$, for all $i \geq 0$. Since the run is infinite, we cannot define acceptance in terms of the final state of the run. Instead we have to consider the *limit* behavior of the run. We define $\lim(r)$ to be the set $\{s \mid s = s_i$ for infinitely many i's$\}$, i.e., the set of states that occur in r infinitely often. Since S is finite, $\lim(r)$ is necessarily nonempty. The run r is *accepting* if $\lim(r) \cap F \neq \emptyset$, i.e., if there is some accepting state that repeats in r infinitely often. The infinite word w is *accepted* by A if there is an accepting run of A on w. The infinitary language of A, denoted $L_\omega(A)$, is the set of infinite words accepted by A. An automaton A is *nonempty* if $L_\omega(A) \neq \emptyset$. The *nonemptiness problem* for Büchi automata is to decide, given a Büchi automaton A, whether A is nonempty.

The following result is taken from [VW94]. As we shall need details from the proof, we repeat it here.

Proposition 5.8 [VW94] *The nonemptiness problem for Büchi automata is in NLOGSPACE.*

Proof Let $A = (\Sigma, S, S^0, \rho, F)$ be an automaton and assume that $s, t \in S$. We say that t *is connected to* s if there exists a sequence $s_1, \ldots, s_k$ of states in S and a sequence $a_1, \ldots, a_k$ of symbols in Σ such that $s_1 = s$, $s_k = t$, and $s_i \in \rho(s_{i-1}, a_i)$ for $1 < i \leq k$. If in addition $k = 2$, then we say that t *is directly connected to* s. We claim that $L_\omega(A)$ is nonempty iff there exist states $s_0 \in S^0$ and $t \in F$ such that t is connected to s_0 and t is connected to itself. To see this, suppose first that $L_\omega(A)$ is nonempty. Then there is an accepting run $r = s_0, s_1, \ldots$ of A on some input word. Clearly, s_{i+1} is directly connected to s_i for all $i \geq 0$. Thus, s_j is connected to s_i whenever $i < j$. Since r is accepting, some $t \in F$ occurs in r infinitely. In particular, there exist i, j, where $0 < i < j$, such that $t = s_i = s_j$. Thus, t is connected to $s_0 \in S^0$ and t is also connected to itself.

Conversely, suppose that there exist states $s_0 \in S^0$ and $t \in F$ such that t is connected to s_0 and t is connected to itself. Since t is connected to s_0, there exists a sequence $s_1, \ldots, s_k$ of states and a sequence $a_1, \ldots, a_k$ of symbols such that $s_k = t$ and $s_i \in \rho(s_{i-1}, a_i)$ for $1 < i \leq k$. Similarly, since t is connected to itself, there exists a sequence $t_0, t_1, \ldots, t_l$ of states and a sequence $b_1, \ldots, b_l$ of symbols such that $t_0 = t_k = t$ and $t_i \in \rho(t_{i-1}, b_i)$ for $1 < i \leq l$. Thus, $(s_0, s_1, \ldots, s_{k-1})(t_0, t_1, \ldots, t_{l-1})^\omega$ is an accepting run of A on input $(a_1, \ldots, a_k)(b_1, \ldots, b_l)^\omega$, so $L_\omega(A)$ is nonempty.

Thus, automata nonemptiness is reducible to *graph reachability*, and graph reachability can be tested in *nondeterministic* logarithmic space. The algorithm simply guesses a state $s_0 \in S^0$, then guesses a state s_1 that is directly connected to s_0, then guesses a state s_2 that is directly connected to s_1, etc., until it reaches a state $t \in F$. At that point the algorithm remembers t and it continues to move nondeterministically from a state s to a state s' that is directly connected to s until it reaches t again. Clearly, the algorithm needs only logarithmic memory, since it needs to remember at most a description of three states at each step. ∎

Recall that one can define the truth of temporal formulas in a run r with respect to an interpretation π. In fact, the truth can be defined with respect to the *interpreted run* $\pi(r)$, where $\pi(r)$ is the sequence $\pi(r(0)), \pi(r(1)), \ldots$ of truth assignments on Φ, where Φ is the set of primitive propositions in the underlying language. This sequence can be viewed as an infinite word on the alphabet 2^Φ. The next proposition is from [VW94]. We denote the cardinality of a set S by $|S|$ and the size of a formula φ (the number of symbols in φ) by $|\varphi|$.

Proposition 5.9 [VW94] *There is an exponential-time algorithm that takes as input a temporal formula φ, and constructs a Büchi automaton $A_\varphi = (\Sigma, S, S^0, \rho, F)$, where $\Sigma = 2^\Phi$, Φ is the set of primitive propositions in φ, and $|S| = 2^{O(|\varphi|)}$, such that $L_\omega(A_\varphi)$ is exactly the set of interpreted runs satisfying the formula φ.*

We can now prove our complexity results for general programs and contexts.

Theorem 5.10 *Testing whether there is at least one (resp., precisely one) interpreted system representing a given finite knowledge-based program in a given finite-state interpreted context is PSPACE-complete.*

Proof Let Pg be a knowledge-based program and let (γ, π) be an interpreted context. Note that $|\mathit{test}(\mathsf{Pg})|$ is linear in the size of Pg. By Proposition 5.7, there is an interpreted system that represents Pg in (γ, π) iff there is a knowledge interpretation κ that is compatible with Pg in (γ, π). The algorithm simply guesses a knowledge interpretation κ and checks that it is indeed compatible. We now show that this can be done in nondeterministic polynomial space. Thus, the problem is in *PSPACE* by [Sav70].

Given a knowledge interpretation κ, a local state $\ell \in L_\gamma$, and a formula $K_i\psi \in test(\mathsf{Pg})$ such that $\kappa(\ell, K_i\psi) = \mathbf{false}$, we have to check that there is an annotated state (g, Θ) that occurs in an annotated run that is consistent both with (κ, π) and with $\mathsf{Pg}^{\kappa,\pi}$ in γ such that $g_i = \ell$ and $\neg\psi \in \Theta$. Let $\mathcal{F}$ be the set of annotated states that are consistent with (κ, π). Consider an annotated run $\alpha = (g^0, \Theta^0), (g^1, \Theta^1), \ldots$; it can be viewed as an infinite word over $\mathcal{F}$. We construct a Büchi automaton A that accepts precisely the set of interpreted runs over $\mathcal{F}$ that are consistent with $\mathsf{Pg}^{\kappa,\pi}$ in γ and that contain an annotated state (g, Θ) such that $g_i = \ell$ and $\neg\psi \in \Theta$. All we then have to check is that A is nonempty. The automaton A is of size polynomial in the number of global states in γ (although it may be exponential in the admissibility condition Ψ) and exponential in the size of Pg. By Proposition 5.8, nonemptiness of Büchi automata can be tested in nondeterministic logarithmic space, so nonemptiness of A can be tested in nondeterministic space that is polynomial in the size of the input.

The latter argument requires some care. We cannot simply construct A and then test it for nonemptiness, since A is exponentially big. Instead, we construct A "on-the-fly". First, the algorithm guesses an initial state of A. Then whenever the nonemptiness algorithm wants to move from a state t_1 of A to a state t_2, the algorithm guesses t_2 and checks that it is directly connected to t_1. The description of A is such that checking whether a state t is initial or checking whether a state t_1 is directly connected to a state t_2 can be done using polynomial space. Once this has been verified, the algorithm can discard t_1. Thus, at each step the algorithm needs to keep in memory at most three states of A and there is no need to generate all of A at any single step of the algorithm. In other words, the algorithm is essentially the nondeterministic algorithm described in the proof of Proposition 5.8, except that it uses polynomial space rather than logarithmic space, due to the exponential size of the automaton under consideration.

It remains to describe the construction of A. We take A to be the composition of four Büchi automata A_1, A_2, A_3, and A_4. On input $\alpha = (g^0, \Theta^0), (g^1, \Theta^1), \ldots$, the automaton A_1 checks that $run(\alpha)$ satisfies the admissibility condition Ψ. For this we simply use the automaton A_Ψ of Proposition 5.9. We see from Proposition 5.9 that A_1 has size exponential in Ψ. The automaton A_2 checks that α is weakly consistent with $\mathsf{Pg}^{\kappa,\pi}$ in γ. Take $A_2 = (\mathcal{F}, \mathcal{G}, \mathcal{G}_0, \rho, \mathcal{G})$, where $\rho(g_1, (g_2, \theta)) = \emptyset$ if $g_1 \neq g_2$, and where $g' \in \rho(g, (g, \theta))$ iff $g = (\ell_e, \ell_1, \ldots, \ell_n)$ and there is a joint action $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n) \in P_e(\ell_e) \times \mathsf{Pg}_1^{\kappa,\pi}(\ell_1) \times \cdots \times \mathsf{Pg}_n^{\kappa,\pi}(\ell_n)$ such that $g' = \tau(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)(g)$. That is, A_2 simply simulates $\mathsf{Pg}^{\kappa,\pi}$. Clearly, A_2 can be constructed in polynomial time from γ and Pg. Note that A_1 and A_2 together verify that $run(\alpha)$ is consistent with $\mathsf{Pg}^{\kappa,\pi}$ in γ. The automaton A_3 checks that α is indeed an annotated run, that is, that it satisfies the proper temporal behavior. For a detailed description of a similar construction see [VW86]. The size of A_3 is exponential in the size of Pg. Finally, A_4 is a 2-state automaton that checks that for some (g^m, Θ^m) we have that $g_i^m = \ell$ and $\neg\psi \in \Theta^m$. The automaton A is taken to be the cross product $A_1 \times \cdots \times A_4$; for details on the product construction for Büchi automata see [Cho74]. The important feature of the product construction is that $L_\omega(A_1 \times \cdots \times A_4) = L_\omega(A_1) \cap \ldots \cap L_\omega(A_4)$.

To check that there is precisely one interpreted system that represents Pg in (γ, π), we check that there is such an interpreted system, but no more than one. We now show that we can check in polynomial space whether there is more than one interpreted system that represents Pg in (γ, π). By Proposition 5.7, this means that we need to check that there are two knowledge interpretations κ_1 and κ_2 compatible with Pg in (γ, π) and a run $r \in \mathbf{R}^{rep}(\mathsf{Pg}^{\kappa_1,\pi}, \gamma) - \mathbf{R}^{rep}(\mathsf{Pg}^{\kappa_2,\pi}, \gamma)$. The first step is to guess κ_1 and κ_2 and check that each is indeed compatible with Pg in (γ,π). To check that there is a run $r \in \mathbf{R}^{rep}(\mathsf{Pg}^{\kappa_1,\pi},\gamma) - \mathbf{R}^{rep}(\mathsf{Pg}^{\kappa_2,\pi},\gamma)$, we first construct a Büchi automaton A_{κ_1} that accepts precisely the runs in $\mathcal{R}(\mathsf{Pg}^{\kappa_1,\pi},\gamma)$. This automaton is essentially the product of the automata A_1 and A_2 described above, so its size is exponential in the size of Pg and Ψ, but polynomial in the number of global states in γ. We can similarly construct an automaton A_{κ_2} that accepts the runs in $\mathcal{R}(\mathsf{Pg}^{\kappa_2,\pi},\gamma)$. The automaton that accepts the runs in $\mathcal{R}(\mathsf{Pg}^{\kappa_1,\pi},\gamma) - \mathcal{R}(\mathsf{Pg}^{\kappa_2,\pi},\gamma)$ is then $A_{\kappa_1,\kappa_2} = A_{\kappa_1} \times \overline{A_{\kappa_2}}$, where $\overline{A_{\kappa_2}}$ is the complement of A_{κ_2}, and accepts precisely the runs rejected by A_{κ_2}. Notice that a run r is not accepted by A_{κ_2} if it either does not satisfy the admissibility condition Ψ (which can be checked using the automaton $A_{\neg\Psi}$, which has exponential size, of Proposition 5.9) or if it is not weakly consistent with $\mathsf{Pg}^{\kappa_2,\pi}$ in γ (which can easily be checked by an automaton of polynomial size that checks whether r contains a global state $g = (\ell_e, \ell_1, \ldots, \ell_n)$ followed by a global state g', but there is no joint action $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n) \in P_e(\ell_e) \times \mathsf{Pg}_1^{\kappa_2,\pi}(\ell_1) \times \cdots \times \mathsf{Pg}_n^{\kappa_2,\pi}(\ell_n)$ such that $g' = \tau(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n)(g)$). It is clear that $\overline{A_{\kappa_2}}$ has exponential size, just as A_{κ_2} does. It remains to check that A_{κ_1,κ_2} is nonempty. Since this automaton has exponential size, this can be done in polynomial space.

Finally, we must show that testing whether there is at least one (resp., precisely one) interpreted system representing a given finite knowledge-based program in a given finite-state interpreted context is *PSPACE*-hard. We show that this is the case even if either the interpreted context is nonrestrictive or the knowledge-based program is atemporal. The reduction is from the satisfiability problem for temporal formulas [SC85].

Suppose $\Phi = \{p\}$ and φ is a temporal formula over Φ.[2] We now describe a finite-state interpreted context (γ,π) and an atemporal finite knowledge-based program Pg such that φ is satisfiable iff there is an interpreted system that represents Pg in (γ,π). The set of environment states is $\{1,2\}$. There is only one agent in the context γ, who is always in the same fixed local state. Thus, we can identify the global state with the environment's state. We take 1 to be the initial state. Assume that p is true in the state 2 but not in the state 1. The set of the agent's actions is $\{\mathsf{a}_1, \mathsf{a}_2\}$, but the environment can perform only a single action, so that we can identify a joint action with the agent's action. We define $\tau(\mathsf{a}_i)(j) = i$, independent of j, i.e., the action a_i always moves the system to the state i. Finally, we take Ψ to be $\bigcirc\varphi$. This concludes the definition of (γ,π).

Let Pg be the following atemporal program:

case of
  if *true* **do** a_1
  if *true* **do** a_2
end case.

Clearly, if φ is not satisfiable, neither is $\bigcirc\varphi$, so there is no system representing Pg in (γ,π). On the other hand, note that if φ is satisfiable, then, since φ mentions only the primitive

[2]The *PSPACE*-hardness proof in [SC85] uses temporal formulas with an unbounded number of primitive propositions. By using a Turing machine M that accepts a *PSPACE*-complete language, it is possible to bound the number of primitive proposition used to the size of the working alphabet of M. Since it is possible to encode the truth values of m primitive proposition in one state by the truth values of a single primitive proposition along $\log m$ states, it follows that satisfiability of temporal formulas with a single primitive proposition is also *PSPACE*-hard.

proposition p, there is a run of the form $1(1+2)^\omega$ that satisfies $\bigcirc\varphi$. Moreover, if φ is satisfiable, it is not hard to see that Pg is represented in (γ, π) by the unique interpreted system that consists of all runs of the form $1(1+2)^\omega$ that satisfy $\bigcirc\varphi$. This shows *PSPACE*-hardness even when the knowledge-based program is atemporal (indeed, standard—since *true* is the only formula in tests). We now show *PSPACE*-hardness when the interpreted context is nonrestrictive.

Let γ' be the context that results by replacing the admissibility condition $\bigcirc\varphi$ in γ by *True*; this means that γ' is now nonrestrictive. Let Pg' be the program

case of
  if $K\neg p$ **do** a_2
  if $\neg K\neg\varphi$ **do** a_1
  if $\neg K\neg\varphi$ **do** a_2
end case.

As before, it is easy to see that if φ is satisfiable, then there is an interpreted system $\mathcal{I}$ representing Pg' in (γ', π); $\mathcal{I}$ simply consists of all runs of the form $1(1+2)^\omega$. (Note that the first clause in Pg' does not play any role here.) Now suppose that Pg' is represented in (γ, π) by $\mathcal{I}'$. We claim that φ must hold at some point in $\mathcal{I}'$. For suppose not. Clearly the second and third clauses are not selected by $\mathsf{Pg}^{\mathcal{I}'}$. The first clause is selected only if the state 2 does not occur in $\mathcal{I}'$, but then a_2 is selected, which changes the state to 2. On the other hand, if the first clause is not selected, then no test is satisfied. By assumption, this means that the action Λ is performed at all times. This, in turn, means that the system consists of one run, where the global state is always 1. But then $K\neg p$ holds, which means that the first clause has to be selected. Thus, φ must hold at some point of $\mathcal{I}'$. But then both actions a_1 and a_2 are selected by $\mathsf{Pg}^{\mathcal{I}'}$, so $\mathcal{I}'$ consists of all runs of the form $1(1+2)^\omega$, which means that $\mathcal{I}' = \mathcal{I}$. It follows that if φ is satisfiable, there is a unique interpreted system representing Pg' in (γ', π), and if φ is not satisfiable, then there are no systems representing Pg' in (γ', π'). ∎

Remark 5.11 So far we have ignored the modalities E and C. We now show how they can be handled. Dealing with E is easy:

- We enlarge $test(\mathsf{Pg})$ by adding $K_i\psi$ and $\neg K_i\psi$ for each formula $E\psi \in test(\mathsf{Pg})$.
- We modify the definition of being full so that a full set $\Theta \subseteq test(\mathsf{Pg})$ must satisfy, in addition to the previous requirements, the additional requirement that $E\psi \in \Theta$ iff $K_i\psi \in \Theta$ for $1 \le i \le n$.

Dealing with the modality C is somewhat more involved:

- We enlarge $test(\mathsf{Pg})$ by adding $K_iC\psi$ and $\neg K_iC\psi$ for each formula $C\psi \in test(\mathsf{Pg})$.
- We modify the definition of being full so that a full set $\Theta \subseteq test(\mathsf{Pg})$ must satisfy, in addition to the previous requirements, the additional requirement that $C\psi \in \Theta$ iff $K_iC\psi \in \Theta$ for $1 \le i \le n$.

- We modify the definition of compatibility so that for κ to be compatible with Pg in (γ, π), we also require that for each local state $\ell \in L_i$ and each formula $K_i C\psi \in \mathit{test}(\mathsf{Pg})$, we have $\kappa(\ell, K_i C\psi) = \textbf{false}$ iff there is a sequence $(g^1, \Theta^1), (g^2, \Theta^2), \ldots (g^k, \Theta^k)$ of annotated states, each occurring in an annotated run that is consistent both with (κ, π) and with $\mathsf{Pg}^{\kappa,\pi}$ in context γ such that $g_i^1 = \ell$, $\neg\psi \in \Theta^k$, and for each $1 \leq i < k$ there is some j with $1 \leq j \leq n$ such that $g^i \sim_j g^{i+1}$.

The additional condition on κ ensures that $C\psi$ fails precisely when $E^l\psi$ fails for some $l \geq 1$. Note that this condition can be checked in nondeterministic polynomial space. We simply guess the sequence $(g^1, \Theta^1), (g^2, \Theta^2), \ldots (g^k, \Theta^k)$ and use the automata-theoretic technique to check that each (g^i, Θ^i) occurs in an annotated run that is consistent both with (κ, π) and with $\mathsf{Pg}^{\kappa,\pi}$ in context γ. ∎

5.3 Testing Implementations

So far we have dealt with the question of whether a given finite knowledge-based program Pg has a (unique) representation in a finite interpreted context (γ, π). As we observed earlier, there is a representation precisely if Pg is implemented by some protocol P in (γ, π). Suppose, however, that we are also given a protocol P and we want to decide whether P implements Pg in (γ, π). Is this problem easier than deciding whether Pg is implemented by *some* protocol? We now show that this problem is indeed easier (provided $P \neq NP$) for atemporal knowledge-based programs and nonrestrictive interpreted contexts, but is not easier in general.

We first consider the simpler setting, where things are indeed easier.

Proposition 5.12 *Let Pg be an atemporal knowledge-based program, let (γ, π) be a nonrestrictive interpreted context, and let P be a protocol. Then P implements Pg in (γ, π) iff $P = \mathsf{Pg}^{\mathcal{F}_\mathcal{R}}$, where $\mathcal{R} = \mathbf{R}^{rep}(P, \gamma)$.*

Proof First suppose that P implements Pg in (γ, π), i.e., $P = \mathsf{Pg}^{\mathcal{I}}$, where $\mathcal{I} = (\mathcal{R}, \pi)$ for $\mathcal{R} = \mathbf{R}^{rep}(P, \gamma)$. By Lemma 5.2, $\mathsf{Pg}^{\mathcal{I}} = \mathsf{Pg}^{\mathcal{F}_\mathcal{R}}$. It follows that $P = \mathsf{Pg}^{\mathcal{F}_\mathcal{R}}$.

Conversely, suppose that $P = \mathsf{Pg}^{\mathcal{F}_\mathcal{R}}$, where $\mathcal{R} = \mathbf{R}^{rep}(P, \gamma)$. By Lemma 5.2, $\mathsf{Pg}^{\mathcal{I}} = \mathsf{Pg}^{\mathcal{F}_\mathcal{R}}$, where $\mathcal{I} = (\mathcal{R}, \pi)$. It follows that P implements Pg in (γ, π). ∎

Theorem 5.13 *There is a polynomial-time algorithm for testing whether a given protocol implements a given atemporal finite knowledge-based program in a given nonrestrictive finite-state interpreted context.*

Proof Let Pg be an atemporal finite knowledge-based program, let (γ, π) be a nonrestrictive finite-state interpreted context, and let P be a protocol. By Proposition 5.12, P implements Pg in (γ, π) iff $P = \mathsf{Pg}^{\mathcal{F}_\mathcal{R}}$, where $\mathcal{R} = \mathbf{R}^{rep}(P, \gamma)$.

To check that P implements Pg in (γ, π), our algorithm computes the set $\mathcal{F}_\mathcal{R}$ of global states using the technique described in the proof of Theorem 5.4, and checks that it satisfies the condition of Proposition 5.12. To check that $P = \mathsf{Pg}^{\mathcal{F}_\mathcal{R}}$ we have to check that $P_i(\ell) = \mathsf{Pg}_i^{\mathcal{F}_\mathcal{R}}(\ell)$ for each agent i and local state ℓ. As observed in the proof of Theorem 5.4, this can be done in polynomial time. ∎

Thus, in the case of an atemporal knowledge-based program Pg and a nonrestrictive interpreted context (γ, π), deciding whether a given protocol P implements Pg in (γ, π) can be decided in polynomial time (Proposition 5.13), whereas deciding whether this knowledge-based program is implemented by *some* protocol in this interpreted context is *NP*-complete (Theorem 5.4). So the first problem is easier, if $P \neq NP$. We now consider the general case.

Proposition 5.14 *Let* Pg *be a knowledge-based program, let* (γ, π) *be an interpreted context, and let* P *be a protocol. Then* P *implements* Pg *in* (γ, π) *iff there is a knowledge interpretation* κ *that is compatible with* Pg *in* (γ, π) *such that* $P = \mathsf{Pg}^{\kappa,\pi}$*.*

Proof First suppose that P implements Pg in (γ, π), i.e., $P = \mathsf{Pg}^{\mathcal{I}}$, where $\mathcal{I} = (\mathcal{R}, \pi)$ for $\mathcal{R} = \mathbf{R}^{rep}(P, \gamma)$. It follows that $\mathcal{I}$ represents Pg in (γ, π), since $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma)$. Define $\kappa(\ell, K_i\psi) =$ **true** iff $(\mathcal{I}, \ell) \models K_i\psi$. Clearly, $\mathsf{Pg}^{\mathcal{I}}$ and $\mathsf{Pg}^{\kappa,\pi}$ are identical, so $P = \mathsf{Pg}^{\kappa,\pi}$. Furthermore, we showed in the proof of Proposition 5.7 that κ is compatible with Pg in (γ, π).

Now suppose that we have a knowledge interpretation κ that is compatible with Pg in (γ, π) such that $P = \mathsf{Pg}^{\kappa,\pi}$. Let $\mathcal{R} = \mathbf{R}^{rep}(\mathsf{Pg}^{\kappa,\pi}, \gamma)$ and $\mathcal{I} = (\mathcal{R}, \pi)$. We showed in the proof of Proposition 5.7 that $\mathsf{Pg}^{\kappa,\pi} = \mathsf{Pg}^{\mathcal{I}}$. It follows that P implements Pg in (γ, π). ∎

Theorem 5.15 *Testing whether a given protocol implements a given finite knowledge-based program in a given finite-state interpreted context is PSPACE-complete.*

Proof Let Pg be a finite knowledge-based program, and let (γ, π) be a finite interpreted context, and let P be a protocol. By Proposition 5.14, P implements Pg in (γ, π) iff there is a knowledge interpretation κ that is compatible with Pg in (γ, π) such that $P = \mathsf{Pg}^{\kappa,\pi}$. We saw in the proof of Theorem 5.10 how to find in polynomial space a knowledge interpretation κ that is compatible with Pg in (γ, π). Clearly, checking that $P = \mathsf{Pg}^{\kappa,\pi}$ can be done in polynomial time. This proves the uper bound.

It remains to show that the problem is *PSPACE*-hard. The reduction is similar to the reduction in the proof of Theorem 5.10, and applies even if either the interpreted context is nonrestrictive or the knowledge-based program is atemporal.

Suppose $\Phi = \{p\}$ and φ is a temporal formula over Φ. Let (γ, π) and Pg be as described in the proof of Theorem 5.10. We define a protocol P by taking $P(1) = P(2) = \{\mathsf{a}_1, \mathsf{a}_2\}$. It is not hard to see that P implements Pg iff φ is satisfiable. This shows *PSPACE*-hardness even when the knowledge-based program is atemporal. To show *PSPACE*-hardness when the interpreted context is nonrestrictive, let Pg' and γ' be as in the proof of Theorem 5.10. Again, it is not hard to see that P implements Pg' in (γ', π) iff φ is satisfiable. ∎

Thus, in contrast to the case of an atemporal knowledge-based program Pg and a nonrestrictive interpreted context (γ, π), we see from Theorems 5.10 and 5.15 that deciding whether a given protocol P implements a general knowledge-based program Pg in a general interpreted context (γ, π) is no easier than deciding whether this knowledge-based program is implemented by *some* protocol in this interpreted context: both problems are *PSPACE*-complete.

6 Concluding remarks

Standard programs work at the level of runs; by way of contrast, knowledge-based programs work at the knowledge level, which provides a higher level of abstraction. We believe that the approach of designing a knowledge-based program that satisfies some specification, and then compiling it to a standard program, will give us a powerful tool for program development. The examples given in [FHMV95] provide some support for this belief.

In this paper, we focused on ways of determining whether a knowledge-based program is represented by a unique system, no system, or many systems. Such information will be crucial if we are to use knowledge-based programs in a serious way. As pointed out in [San91], it would also be useful to have techniques for reasoning about knowledge-based programs without having to construct the system(s) that represent them. The development in [FHMV95] has already simplified the reasoning by distinguishing between programs and contexts, and allowing us to discuss systems representing a program in a given context without having to describe the runs of the system explicitly. Nevertheless, we still need to find ways of employing the technology that has been developed for proving correctness of programs for the task of reasoning about knowledge-based programs. A first step along these lines was taken by Sanders [San91], who extended UNITY in such a way as to allow the definition of knowledge predicates (although it appears that the resulting knowledge-based programs are somewhat less general than those described here), and then used proof techniques developed for UNITY to prove the correctness of another knowledge-based protocol for the sequence transmission problem. (We remark that techniques for reasoning about knowledge obtained in CSP programs, but not for knowledge-based programs, were given in [KT86].)

One potential problem in starting with a knowledge-based program and then implementing it is that, as we stressed in Section 2.6, our definition of knowledge is an external one. Since we do not assume that agents necessarily compute their knowledge, it may not always be straightforward to implement the tests for knowledge that appear in knowledge-based programs. Indeed, an example in which this problem arises appears in [MT88]. The (provably optimal) knowledge-based program for simultaneous Byzantine agreement presented in [MT88] (based on the one in [DM90]) has tests that are *NP*-hard to compute in a context that allows generalized omission failures. (The same tests are efficiently computable, and hence the optimal program is efficiently implementable, in contexts that allow only sending omission failures or crash failures.) Based on the notion of *resource-bounded knowledge* defined in [Mos88], a notion of *algorithmic knowledge* is introduced in [FHMV95] that is intended to capture knowledge that is computable. In addition, *algorithmic programs*, which use tests for algorithmic knowledge, are considered. Algorithmic programs can be viewed as a halfway point between knowledge-based programs and standard programs, since, although they have tests for knowledge, these tests are, in a precise sense, guaranteed to be implementable.

An extension of the framework of knowledge-based programs is presented in [MK93]. Moses and Kislev argue that actions, as well as a program's internal tests, should be thought of at the knowledge level. The effect of sending a single message in a context with reliable communication can be considered similar to sending many messages in an unreliable context. As a result, they define *knowledge-oriented programs*, which are knowledge-based programs involving high-level actions that are defined in terms of knowledge. They illustrate how knowledge-oriented programs can be used to unify solutions to well-known problems in different contexts, as well as to generate efficient programs in a given context by way of top-down design.

It is clear that there is more work to be done in understanding the knowledge-based approach. We feel that the potential advantages of this approach make the effort worthwhile.

A Appendix: Proofs

In this appendix, we fill in the missing details of some of the proofs. We first establish two useful lemmas.

Lemma A.1 *If P is a protocol and $\gamma = (P_e, \mathcal{G}_0, \tau, \Psi)$ is nonexcluding, then $Pref_0(\mathbf{R}^{rep}(P,\gamma)) = \mathcal{G}_0 \cap Pref_0(\Psi)$.*

Proof Clearly, $Pref_0(\mathbf{R}^{rep}(P,\gamma)) \subseteq \mathcal{G}_0 \cap Pref_0(\Psi)$. We now must show that for each state $g \in \mathcal{G}_0 \cap Pref_0(\Psi)$, there is a run $r \in \mathbf{R}^{rep}(P,\gamma)$ such that $r(0) = g$. It is immediate from the definition of a protocol that there is a run r'' weakly consistent with P in context γ such that $r''(0) = g$. It then follows immediately from part (b) of the definition of nonexcluding that there is a run $r \in \mathbf{R}^{rep}(P,\gamma)$ such that $r(0) = g$, as desired. ∎

The next lemma is the key lemma, which shows that our inductive construction has the right properties. Intuitively, this lemma says that, for each interpreted system $\mathcal{I}' \in \mathcal{J}$, the actions of the protocol $\mathsf{Pg}^{\mathcal{I}'}$ at time m depend only on the prefixes of $\mathcal{I}'$ up to time m. This lemma is the only place in the proof where we use the assumption that Pg depends on the past in $\mathcal{J}$; this and the preceding lemma are the only ones that use the assumption that γ is nonexcluding.

Lemma A.2 *Assume that Pg depends on the past in $\mathcal{J}$ and that γ is nonexcluding. Suppose $\mathcal{I}_1, \mathcal{I}_2 \in \mathcal{J}$ and $Pref_m(\mathcal{I}_1) = Pref_m(\mathcal{I}_2) = Pref_m(\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}_1},\gamma,\pi)) = Pref_m(\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}_2},\gamma,\pi))$. Then $Pref_{m+1}(\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}_1},\gamma,\pi)) = Pref_{m+1}(\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}_2},\gamma,\pi))$.*

Proof Suppose $\rho \in Pref_{m+1}(\mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}_1},\gamma))$. Thus, there must exist a run $r \in \mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}_1},\gamma)$ such that $\rho = Pref_{m+1}(r)$. Suppose $r(m) = (\ell_e, \ell_1, \ldots, \ell_n)$. It follows that there must be a tuple $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n) \in P_e(\ell_e) \times \mathsf{Pg}_1^{\mathcal{I}_1}(\ell_1) \times \cdots \times \mathsf{Pg}_n^{\mathcal{I}_1}(\ell_n)$ such that $r(m+1) = \tau((\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n))(r(m))$. We now show that $\mathsf{a}_i \in \mathsf{Pg}_i^{\mathcal{I}_2}(\ell_i)$ for each agent i. By the assumptions of the lemma, there is a run r^1 of $\mathcal{I}_1$ and a run r^2 of $\mathcal{I}_2$ such that $Pref_m(r) = Pref_m(r^1) = Pref_m(r^2)$. Furthermore, $Pref_m(\mathcal{I}_1) = Pref_m(\mathcal{I}_2)$. We know that $\mathsf{a}_i \in \mathsf{Pg}_i^{\mathcal{I}_1}(\ell_i)$. This means that either

(1) there is a line in the knowledge-based program Pg_i of the form "**if** $t \wedge k$ **do** a_i", where t is a standard test and k is a knowledge test, and $(\mathcal{I}_1, \ell_i) \models t \wedge k$, or

(2) a_i is the null action Λ and for each line "**if** $t \wedge k$ **do** a" of Pg_i, necessarily $(\mathcal{I}_1, \ell_i) \not\models t \wedge k$.

First assume that (1) holds. Then $(\pi, \ell_i) \models t$ and $(\mathcal{I}_1, r^1, m) \models k$ (the latter holds since $r_i^1(m) = r_i(m) = \ell_i$). Since (a) Pg depends on the past in $\mathcal{J}$, (b) $\mathcal{I}_1 \in \mathcal{J}$ and $\mathcal{I}_2 \in \mathcal{J}$, (c) $(\mathcal{I}_1, r^1, m) \models k$, (d) $Pref_m(\mathcal{I}_1) = Pref_m(\mathcal{I}_2)$, and (e) $Pref_m(r^1) = Pref_m(r^2)$, it follows that $(\mathcal{I}_2, r^2, m) \models k$. So $(\mathcal{I}_2, \ell_i) \models k$, since $r_i^2(m) = \ell_i$. Also $(\mathcal{I}_2, \ell_i) \models t$, since $(\pi, \ell_i) \models t$. Hence, $(\mathcal{I}_2, \ell_i) \models$

$t \wedge k$. Therefore, under the assumption that (1) holds, we have shown that $\mathsf{a}_i \in \mathsf{Pg}_i^{\mathcal{I}_2}(\ell_i)$, as desired. A similar argument goes through when (2) holds. Hence, $(\mathsf{a}_e, \mathsf{a}_1, \ldots, \mathsf{a}_n) \in P_e(\ell_e) \times \mathsf{Pg}_1^{\mathcal{I}_2}(\ell_1) \times \cdots \times \mathsf{Pg}_n^{\mathcal{I}_2}(\ell_n)$. It follows that there is a run with prefix ρ that is weakly consistent with $\mathsf{Pg}^{\mathcal{I}_2}$ in context γ. Since ρ is in $\mathit{Pref}_{m+1}(\mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}_1}, \gamma))$, and hence in $\mathit{Pref}_{m+1}(\Psi)$, it follows from the fact that γ is nonexcluding that there is a run with prefix ρ that is consistent with $\mathsf{Pg}^{\mathcal{I}_2}$ in context γ. This shows that $\mathit{Pref}_{m+1}(\mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}_1}, \gamma)) \subseteq \mathit{Pref}_{m+1}(\mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}_2}, \gamma))$. Using symmetric arguments, we get that $\mathit{Pref}_{m+1}(\mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}_2}, \gamma)) \subseteq \mathit{Pref}_{m+1}(\mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}_1}, \gamma))$. Therefore, $\mathit{Pref}_{m+1}(\mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}_1}, \gamma)) = \mathit{Pref}_{m+1}(\mathbf{R}^{rep}(\mathsf{Pg}^{\mathcal{I}_2}, \gamma))$, as desired. ∎

We now complete the proof of Theorem 4.7. Recall that we want to show that the system $\mathcal{I}^{\omega+1}$ defined by our inductive construction represents Pg in (γ, π). To do this, we need to show that $\mathcal{I}^{\omega+1} = \mathcal{I}^{\omega+2}$. As unfinished business, we also need to prove that the sequence $\mathcal{I}^0, \mathcal{I}^1, \ldots$ is prefix-compatible.

Claim A.3 If $0 \leq m \leq m' < \omega$, then $\mathit{Pref}_m(\mathcal{I}^{m'}) = \mathit{Pref}_m(\mathcal{I}^m)$.

Proof We proceed by induction on m. The case $m = 0$ follows immediately from Lemma A.1. Suppose we have proved the result for $m = k$ and wish to prove it for $m = k + 1$. Suppose that $m' \geq k+1$. We want to show that $\mathit{Pref}_{k+1}(\mathcal{I}^{m'}) = \mathit{Pref}_{k+1}(\mathcal{I}^{k+1})$. By the induction hypothesis, $\mathit{Pref}_k(\mathcal{I}^{m'-1}) = \mathit{Pref}_k(\mathcal{I}^k) = \mathit{Pref}_k(\mathcal{I}^{m'}) = \mathit{Pref}_k(\mathcal{I}^{k+1})$. We can now apply Lemma A.2 (where the roles of $\mathcal{I}_1$ and $\mathcal{I}_2$ are played by $\mathcal{I}^{m'-1}$ and $\mathcal{I}^k$), since by definition $\mathbf{I}^{rep}(\mathsf{Pg}^{I^{m'-1}}, \gamma, \pi) = \mathcal{I}^{m'}$ and $\mathbf{I}^{rep}(\mathsf{Pg}^{I^k}, \gamma, \pi) = \mathcal{I}^{k+1}$. We then obtain that $\mathit{Pref}_{k+1}(\mathcal{I}^{m'}) = \mathit{Pref}_{k+1}(\mathcal{I}^{k+1})$, as desired. ∎

Incidentally, the reason we named the system in $\mathcal{J}$ that we began our fixed-point construction with to be $\mathcal{I}^{-1}$ rather than $\mathcal{I}^0$ is that, as Claim A.3 tells us, the m-ary prefix of $\mathcal{I}^m$ is preserved for $m \geq 0$. That is, every $\mathcal{I}^{m'}$ for $m' \geq m \geq 0$ has the same m-ary prefix as $\mathcal{I}^m$. This would not necessarily have been true when $m = 0$ had we started our fixed-point construction by taking $\mathcal{I}^0$ to be an arbitrary member of $\mathcal{J}$.

Since $\mathcal{I}^0, \mathcal{I}^1, \ldots$ is prefix-compatible sequence of elements in $\mathcal{J}$ and $\mathcal{J}$ has limits, it follows that there is a limit $\mathcal{I}^\omega$ of this sequence in $\mathcal{J}$. (This is the only place in the proof where we use the fact that $\mathcal{J}$ has limits.) We now define $\mathcal{I}^{\omega+1}$ and $\mathcal{I}^{\omega+2}$ as before, by letting $\mathcal{I}^{\theta+1} = \mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}^\theta}, \gamma, \pi)$, for $\theta = \omega$ and $\theta = \omega + 1$. Since $\mathcal{I}^\omega \in \mathcal{J}$ and $\mathcal{J}$ is Pg-closed, it follows that $\mathcal{I}^{\omega+1} \in \mathcal{J}$.

The next claim provides a tool for proving that $\mathcal{I}^{\omega+1} = \mathcal{I}^{\omega+2}$. Let Pg be a knowledge-based program and (γ, π) an interpreted context. Then $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$, as defined after Theorem 4.9, consists of all interpreted systems $\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}}, \gamma, \pi)$, where $\mathcal{I}$ is of the form $(\mathcal{R}, \pi)$. Note that $\mathcal{I}^\theta$, for $0 \leq \theta < \omega$ or $\omega < \theta \leq \omega + 2$, is in $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$.

Claim A.4 If $\mathcal{I}_1, \mathcal{I}_2 \in \mathcal{J}(\mathsf{Pg}, \gamma, \pi)$ and $\mathit{Pref}_m(\mathcal{I}_1) = \mathit{Pref}_m(\mathcal{I}_2)$ for all m, then $\mathcal{I}_1 = \mathcal{I}_2$.

Proof Suppose $\mathcal{I}_1 = \mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}'_1}, \gamma, \pi)$ and $\mathcal{I}_2 = \mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}'_2}, \gamma, \pi)$. Let r be a run in $\mathcal{I}_1$; we now show that r is a run in $\mathcal{I}_2$. Since $\mathit{Pref}_m(\mathcal{I}_1) = \mathit{Pref}_m(\mathcal{I}_2)$ for all m, r is weakly consistent with $\mathsf{Pg}^{\mathcal{I}'_2}$ in γ. Clearly every run in $\mathcal{I}_1$ (and, in particular r) is in Ψ. It follows that r is consistent with $\mathsf{Pg}^{\mathcal{I}'_2}$ in γ. Therefore, r is a run in $\mathcal{I}_2$. By a symmetric argument, we can show that every run in $\mathcal{I}_2$ is in $\mathcal{I}_1$. Thus, $\mathcal{I}_1 = \mathcal{I}_2$. ∎

Claim A.5 $\mathit{Pref}_m(\mathcal{I}^{\omega+1}) = \mathit{Pref}_m(\mathcal{I}^{\omega+2})$ for all m.

Proof We first show that $\mathit{Pref}_m(\mathcal{I}^{\omega+1}) = \mathit{Pref}_m(\mathcal{I}^m)$ for each m. The case of $m = 0$ follows directly from Lemma A.1. So assume that $m = k+1$ for some nonnegative integer k. By definition of I^ω, we know that $\mathit{Pref}_k(\mathcal{I}^\omega) = \mathit{Pref}_k(\mathcal{I}^k)$. By the induction hypothesis, we have $\mathit{Pref}_k(\mathcal{I}^{\omega+1}) = \mathit{Pref}_k(\mathcal{I}^k)$. By Claim A.3, we have $\mathit{Pref}_k(\mathcal{I}^k) = \mathit{Pref}_k(\mathcal{I}^{k+1})$, so $\mathit{Pref}_k(\mathcal{I}^\omega) = \mathit{Pref}_k(\mathcal{I}^k) = \mathit{Pref}_k(\mathcal{I}^{\omega+1}) = \mathit{Pref}_k(\mathcal{I}^{k+1})$. We can now apply Lemma A.2, where the roles of $\mathcal{I}_1$ and $\mathcal{I}_2$ are played by $\mathcal{I}^\omega$ and $\mathcal{I}^k$. It follows that $\mathit{Pref}_{k+1}(I^{\omega+1}) = \mathit{Pref}_{k+1}(\mathcal{I}^{k+1})$, as desired.

Using the fact that $\mathit{Pref}_m(\mathcal{I}^{\omega+1}) = \mathit{Pref}_m(\mathcal{I}^m)$ for each m, a similar argument (where the roles of $\mathcal{I}_1$ and $\mathcal{I}_2$ in Lemma A.2 are played by $\mathcal{I}^{\omega+1}$ and $\mathcal{I}^k$) shows that $\mathit{Pref}_m(\mathcal{I}^{\omega+2}) = \mathit{Pref}_m(\mathcal{I}^m)$ for each m. Therefore, $\mathit{Pref}_m(\mathcal{I}^{\omega+1}) = \mathit{Pref}_m(\mathcal{I}^{\omega+2})$ for each m. ∎

By Claims A.4 and A.5, we must have $\mathcal{I}^{\omega+1} = \mathcal{I}^{\omega+2}$, as desired. This completes the proof of Theorem 4.7. (Note that $\mathit{Pref}_m(\mathcal{I}^\omega) = \mathit{Pref}_m(\mathcal{I}^{\omega+1})$ for all m. We cannot, however, apply Claim A.4 to show that $\mathcal{I}^\omega = \mathcal{I}^{\omega+1}$, since $\mathcal{I}^\omega$ is not necessarily in $\mathcal{J}(\mathsf{Pg}, \gamma, \pi)$.)

We now give the rest of the proof of Theorem 4.8 (i.e., the "if" direction). Recall that we want to show that if γ is nonexcluding and Pg depends on the past in $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$, then there is at most one system representing Pg in (γ, π).

Suppose that $\mathcal{I}_1$ and $\mathcal{I}_2$ are two systems in $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$; we want to show that $\mathcal{I}_1 = \mathcal{I}_2$. To do this, we want to apply Claim A.4. Thus, we first show the following claim.

Claim A.6 If $\mathcal{I}_1$ and $\mathcal{I}_2$ are two systems in $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$, then $\mathit{Pref}_m(\mathcal{I}_1) = \mathit{Pref}_m(\mathcal{I}_2)$ for all m.

Proof We prove this by induction on m. Since $\mathcal{I}_1$ and $\mathcal{I}_2$ are in $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi)$, we know that $\mathcal{I}_1 = \mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}_1}, \gamma, \pi)$ and $\mathcal{I}_2 = \mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}_2}, \gamma, \pi)$. The base case $m = 0$ is now immediate from Lemma A.1. For the inductive step, assume that $\mathit{Pref}_m(\mathcal{I}_1) = \mathit{Pref}_m(\mathcal{I}_2)$. By Lemma A.2, $\mathit{Pref}_{m+1}(\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}_1}, \gamma, \pi)) = \mathit{Pref}_{m+1}(\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}_2}, \gamma, \pi))$. Therefore, $\mathit{Pref}_{m+1}(\mathcal{I}_1) = \mathit{Pref}_{m+1}(\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}_1}, \gamma, \pi)) = \mathit{Pref}_{m+1}(\mathbf{I}^{rep}(\mathsf{Pg}^{\mathcal{I}_2}, \gamma, \pi)) = \mathit{Pref}_{m+1}(\mathcal{I}_2)$, as desired. ∎

As we said above, the fact that $\mathcal{I}_1 = \mathcal{I}_2$ now follows from Claim A.4, since $\mathsf{REP}(\mathsf{Pg}, \gamma, \pi) \subseteq \mathcal{J}(\mathsf{Pg}, \gamma, \pi)$. This completes the proof of Theorem 4.8.

Acknowledgments:

We would like to thank the two anonymous referees for their careful reading of the paper.

References

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[Bar81] J. Barwise. Scenes and other situations. *Journal of Philosophy*, 78(7):369–397, 1981.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, 1986. An early version appeared in *Proc. 10th ACM Symposium on Principles of Programming Languages*, 1983.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and System Sciences*, 8:117–141, 1974.

[CM88] K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation.* Addison-Wesley, Reading, Mass., 1988.

[DM90] C. Dwork and Y. Moses. Knowledge and common knowledge in a Byzantine environment: crash failures. *Information and Computation*, 88(2):156–186, 1990.

[FHMV94] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. An operational semantics for knowledge bases. In *Proc. National Conference on Artificial Intelligence (AAAI '94)*, pages 1142–1147, 1994.

[FHMV95] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge.* MIT Press, Cambridge, Mass., 1995.

[FT91] D. Fudenberg and J. Tirole. *Game Theory.* MIT Press, Cambridge, Mass., 1991.

[GJ79] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. Freeman and Co., San Francisco, Calif., 1979.

[HF85] J. Y. Halpern and R. Fagin. A formal model of knowledge, action, and communication in distributed systems: preliminary report. In *Proc. 4th ACM Symp. on Principles of Distributed Computing*, pages 224–236, 1985.

[HF89] J. Y. Halpern and R. Fagin. Modelling knowledge and action in distributed systems. *Distributed Computing*, 3(4):159–179, 1989. A preliminary version appeared in *Proc. 4th ACM Symposium on Principles of Distributed Computing*, 1985, with the title "A formal model of knowledge, action, and communication in distributed systems: preliminary report".

[HM90] J. Y. Halpern and Y. Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM*, 37(3):549–587, 1990. A preliminary version appeared in *Proc. 3rd ACM Symposium on Principles of Distributed Computing*, 1984.

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

[HMW90] J. Y. Halpern, Y. Moses, and O. Waarts. A characterization of eventual Byzantine agreement. In *Proc. 9th ACM Symp. on Principles of Distributed Computing*, pages 333–346, 1990.

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

[JVV86] M.R. Jerrum, L.G. Valiant, and V.V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43:169–188, 1986.

[KT86] S. Katz and G. Taubenfeld. What processes know: definitions and proof methods. In *Proc. 5th ACM Symp. on Principles of Distributed Computing*, pages 249–262, 1986.

[Kur86] R. Kurki-Suonio. Towards programming with knowledge expressions. In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 140–149, 1986.

[Maz91] M. S. Mazer. Implementing distributed knowledge-based protocols. Submitted for publication, 1991.

[MK93] Y. Moses and O. Kislev. Knowledge-oriented programming. In *Proc. 12th ACM Symp. on Principles of Distributed Computing*, pages 261–270, 1993.

[Mos88] Y. Moses. Resource-bounded knowledge. In M. Y. Vardi, editor, *Proc. Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 261–276. Morgan Kaufmann, San Francisco, Calif., 1988.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*, volume 1. Springer-Verlag, Berlin/New York, 1992.

[MT88] Y. Moses and M. R. Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3:121–169, 1988.

[NB92] G. Neiger and R. Bazzi. Using knowledge to optimally achieve coordination in distributed systems. In Y. Moses, editor, *Theoretical Aspects of Reasoning about Knowledge: Proc. Fourth Conference*, pages 43–59. Morgan Kaufmann, San Francisco, Calif., 1992.

[Nei88] G. Neiger. Knowledge consistency: a useful suspension of disbelief. In M. Y. Vardi, editor, *Proc. Second Conference on Theoretical Aspects of Reasoning about Knowledge*, pages 295–308. Morgan Kaufmann, San Francisco, Calif., 1988.

[NT93] G. Neiger and S. Toueg. Simulating real-time clocks and common knowledge in distributed systems. *Journal of the ACM*, 40(2):334–367, 1993.

[OL82] S. Owicki and L. Lamport. Proving liveness properties of concurrent programs. *ACM Trans. on Programming Languages and Systems*, 4(3):455–495, 1982.

[PY82] C. H. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). *Journal of Computer and System Sciences*, 28(2):244–259, 1982.

[San91] B. Sanders. A predicate transformer approach to knowledge and knowledge-based protocols. In *Proc. 10th ACM Symp. on Principles of Distributed Computing*, pages 217–230, 1991. A revised report appears as ETH Informatik Technical Report 181, 1992.

[Sav70] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4:177–192, 1970.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

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

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st IEEE Symp. on Logic in Computer Science*, pages 332–344, 1986.

[VW94] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.