The complexity of approximations for epistemic synthesis (extended abstract)

Xiaowei Huang
UNSW Australia
xiaoweih@cse.unsw.edu.au

Ron van der Meyden
UNSW Australia
meyden@cse.unsw.edu.au

Epistemic protocol specifications allow programs, for settings in which multiple agents act with incomplete information, to be described in terms of how actions are related to what the agents know. They are a variant of the knowledge-based programs of Fagin et al [Distributed Computing, 1997], motivated by the complexity of synthesizing implementations in that framework. The paper proposes an approach to the synthesis of implementations of epistemic protocol specifications, that reduces the problem of finding an implementation to a sequence of model checking problems in approximations of the ultimate system being synthesized. A number of ways to construct such approximations is considered, and these are studied for the complexity of the associated model checking problems. The outcome of the study is the identification of the best approximations with the property of being PTIME implementable.

1 Introduction

Knowledge-based programs [9] are an abstract specification format for concurrent systems, in which the actions of an agent are conditional on formulas of the logic of knowledge [8]. This format allows the agent to be described in terms of what it must know in order to perform its actions, independently of how that knowledge is attained or concretely represented by the agent. This leads to implementations that are optimal in their use of the knowledge implicitly available in an agent's local state. The approach has been applied to problems including reliable message transmission [12], atomic commitment [11], fault-tolerant agreement [6], robot motion planning [4] and cache coherency [1].

The process of going from an abstract knowledge-based program to a concrete implementation is non-trivial, since it requires reasoning about all the ways that knowledge can be obtained, which can be quite subtle. Adding to the complexity, there is a circularity in that knowledge determines actions, which in turn affect the knowledge that an agent has. It is therefore highly desirable to be able to automate the process of implementation. Unfortunately, this is known to be an inherently complex problem: even deciding whether an implementation exists is intractable [9].

Sound local proposition epistemic specifications [7] are a generalization of knowledge-based programs proposed in part due to these complexity problems. These specifications require only *sufficient* conditions for knowledge, where knowledge-based programs require *necessary and sufficient conditions*. By allowing a larger space of potential implementations, this variant ensures that there always exists an implementation. However, some of these implementations are so trivial as to be uninteresting. In practice, one wants implementations in which agents make good use of their knowledge, so that the conditions under which they act closely approximate the necessary and sufficient conditions for knowledge. To date, a systematic approach to the identification of *good* implementations, and of automating the construction of such good implementations, has not been identified. This is the problem we address in the present paper. Ultimately, we seek an automated approach that is implementable in a way that scales

To appear in EPTCS.

to handling realistic examples. In this paper, we use a CTL basis for specifications, and use PTIME complexity of an associated model checking problem in an explicit state representation as a proxy for practical implementability.

The contributions of the paper are two-fold: first, we present a general approach to the identification of good implementations, that extends the notion of sound local proposition epistemic specification by ordering the knowledge conditions to be synthesized, and then defining a way to construct implementations using a sequence of approximations to the final synthesized system, in which implementation choices for earlier knowledge conditions are fed back to improve the quality of approximation used to compute later implementation choices. This gives an intuitive approach to the construction of implementations, which we show by example to address some unintuitive aspects of the original knowledge-based program semantics. The approach is parametric in a choice of approximation scheme.

Second, we consider a range of possibilities for the approximation scheme to be used in the above ordered semantics, and evaluate the complexity of the synthesis computations associated with each approximation. The analysis leads to the identification of two orthogonal approximations that are optimal in their closeness to a knowledge-based program semantics, while remaining PTIME computable. This identifies the best prospects for future work on synthesis implementations.

The paper is structured as follows. Section 2 recalls basic definitions of temporal epistemic logic. Section 3 defines epistemic protocol specifications. In Section 4 we define the ordered semantics approximation approach for identification of good implementations. Section 5 defines a range of possible approximation schemes, which are then analyzed for complexity in Section 6. We discuss related work in Section 7 and conclude with a discussion of future work in Section 8.

2 A Semantic model for Knowledge and Time

In this section we lay out a general logical framework for agent knowledge, and describe how knowledge arises for agents that execute a concrete protocol in the context of some environment.

Let $Prop$ be a finite set of atomic propositions and Ags be a finite set of agents. The language CTL*K($Prop, Ags$) has the syntax:

$$\phi ::= \ p \mid \neg\phi \mid \phi_1 \vee \phi_2 \mid X\phi \mid (\phi_1 U \phi_2) \mid A\phi \mid K_i\phi$$

where $p \in Prop$ and $i \in Ags$. This is CTL* plus the construct $K_i\phi$, which says that agent i knows that ϕ holds. We freely use standard operators that are definable in terms of the above, specifically $F\phi = \mathbf{true}U\phi$, $G\phi = \neg F\neg\phi$, $\phi_1 R\phi_2 = \neg((\neg\phi_1)U(\neg\phi_2))$, $E\phi = \neg A\neg\phi$. Our focus in this paper is on the fragment CTLK, in which the branching operators may occur only as $A\phi$ and $E\phi$, where ϕ is a formula in which the outermost operator is one of the temporal operators X, U, R, F or G. A further subfragment of this language CTLK$^+$, specified by the grammar

$$\phi ::= \ p \mid \neg p \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid AX\phi \mid AF\phi \mid AG\phi \mid A(\phi_1 U \phi_2) \mid A(\phi_1 R \phi_2) \mid K_i\phi$$

where $p \in Prop$ and $i \in Ags$. Intuitively, this is the sublanguage in which all occurrences of the operators A and K_i are in positive position.

To give semantics to all these languages it suffices to give semantics to CTL*K($Prop, Ags$). We do this using a variant of interpreted systems [8]. Let S be a set, which we call the set of global states. A *run* over S is a function $r : \mathbf{N} \to S$. A *point* is a pair (r, m) where r is a run and $m \in \mathbf{N}$. Given a set $\mathcal{R}$ of runs, we define $Points(\mathcal{R})$ to be the set of all points of runs $r \in \mathcal{R}$. An *interpreted system* for n agents

is a tuple $\mathcal{I} = (\mathcal{R}, \sim, \pi)$, where $\mathcal{R}$ is a set of runs over S, the component $\sim$ is a collection $\{\sim_i\}_{i \in Ags}$, where for each $i \in Ags$, $\sim_i$ is an equivalence relation on $Points(\mathcal{R})$ (called agent i's *indistinguishability relation*) and $\pi : S \to \mathcal{P}(Prop)$ is an interpretation function. We say that a run r' is *equivalent to a run r up to time $m \in \mathbf{N}$* if $r'(k) = r(k)$ for $0 \leq k \leq m$.

We can define a general semantics of CTL*K(V, Ags) by means of a relation $\mathcal{I}, (r,m) \models \phi$, where $\mathcal{I}$ is an intepreted system, (r,m) is a point of $\mathcal{I}$ and ϕ is a formula. This relation is defined inductively as follows:

- $\mathcal{I}, (r,m) \models p$ if $p \in \pi(r(m))$, for $p \in Prop$;
- $\mathcal{I}, (r,m) \models \neg\phi$ if not $\mathcal{I}, (r,m) \models \phi$;
- $\mathcal{I}, (r,m) \models \phi_1 \vee \phi_2$ if $\mathcal{I}, (r,m) \models \phi_1$ or $\mathcal{I}, (r,m) \models \phi_2$;
- $\mathcal{I}, (r,m) \models A\phi$ if $\mathcal{I}, (r',m) \models \phi$ for all runs $r' \in \mathcal{R}$ equivalent to r up to time m;
- $\mathcal{I}, (r,m) \models X\phi$ if $\mathcal{I}, (r,m+1) \models \phi$;
- $\mathcal{I}, (r,m) \models \phi_1 U \phi_2$ if there exists $m' \geq m$ such that $\mathcal{I}, (r,m') \models \phi_2$, and $\mathcal{I}, (r,k) \models \phi_1$ for $m \leq k < m'$;
- $\mathcal{I}, (r,m) \models K_i\phi$ if $\mathcal{I}, (r',m') \models \phi$ for all points $(r',m') \sim_i (r,m)$ of $\mathcal{I}$.

For the knowledge operators, this semantics is essentially the same as the usual interpreted systems semantics. For the temporal operators, it corresponds to a semantics for branching time known as the *bundle semantics* [5, 22]. We write $\mathcal{I} \models \phi$ when $\mathcal{I}, (r,0) \models \phi$ for all runs r of $\mathcal{I}$.

We are interested in systems in which each of the agents runs a protocol in which it chooses its actions based on local information, in the context of a larger environment. An *environment* for agents Ags is a tuple $E = \langle S, I, \{Acts_i\}_{i \in Ags}, \longrightarrow, \{O_i\}_{i \in Ags}, \pi \rangle$, where

1. S is a finite set of states,
2. I is a subset of S, representing the initial states,
3. for each agent i, component $Acts_i$ is a finite set of actions that may be performed by agent i; we define $Acts = \Pi_{i \in Ags} Acts_i$ to be the corresponding set of *joint actions*
4. $\longrightarrow \subseteq S \times Acts \times S$ is a transition relation, labelled by joint actions,
5. for each $i \in Ags$, component O_i is a mapping from S to some set O of observations,
6. $\pi : S \to \mathcal{P}(Prop)$ is an interpretation of some set of atomic propositions $Prop$.

Intuitively, a joint action $\mathbf{a}$ represents a choice of action $\mathbf{a}_i$ for each agent, performed simultaneously, and the transition relation resolves this into an effect on the state. We assume that $\longrightarrow$ is serial in the sense that for all $s \in S$ and $\mathbf{a} \in Acts$ there exists $t \in S$ such that $s \xrightarrow{\mathbf{a}} t$. We assume that $Acts_i$ always contains at least an action **skip**, and that for the joint action $\mathbf{a}$ with $\mathbf{a}_i =$ **skip** for all agents i, we have $s \xrightarrow{\mathbf{a}} t$ iff $s = t$. The set O of observations is an arbitrary set: for each agent i, we will be interested in the equivalence relation $s \sim_i t$ if $O_i(s) = O_i(t)$ induced by the observation function O_i rather than the actual values of O_i.

A proposition p is *local to agent i* in the enviroment E if it depends only on the agent's observation, in the sense that for all states s, t with $O_i(s) = O_i(t)$, we have $p \in \pi(s)$ iff $p \in \pi(t)$. We write $Prop_i$ for the set of propositions local to agent i. Intuitively, these are the propositions whose values the agent can always determine, based just on its observation. We similarly say that a boolean formula is local to agent i if it contains only propositions that are local to agent i. We assume that the set of local propositions is *complete* with respect to the observations, in that for each observation o there exists a local formula ϕ such that for all states s, we have $O_i(s) = o$ iff $\pi(s) \models \phi$. (This can be ensured by including a proposition

p_o that is true at just states s with $O_i(s) = o$, or by including a proposition $v = c$ for each possible value c of each variable v making up agent i's observation.)

A *concrete protocol* for agent $i \in Ags$ in such an environment E is a Dijkstra style nondeterministic looping statement P_i of the form

$$\textbf{do } \phi_1 \to a_1 \ [] \ \ldots \ [] \ \phi_k \to a_k \ \textbf{ od} \tag{1}$$

where the a_j are actions in $Acts_i$ and the ϕ_j are boolean formulas local to agent i. Intuitively, this is a nonterminating program that is executed by the agent repeatedly checking which of the guards ϕ_j holds, and then nondeterministically performing one of the corresponding actions a_i. If none of the guards holds, then the action **skip** is performed. That is, implicitly, there is an additional clause $\neg\phi_1 \wedge \ldots \neg\phi_n \to \textbf{skip}$. Without loss of generality, we may assume that the a_i are distinct. (We can always amalgamate two cases $\phi_1 \to a$ and $\phi_2 \to a$ with the same action a into a single case $\phi_1 \vee \phi_2 \to a$.) We say that action a_j is enabled in protocol P_i at state s if ϕ_j holds with respect to the assignment $\pi(s)$, and write $en(P_i, s)$ for the set of all actions enabled in protocol P_i at state s.

A *joint protocol* P is a collection $\{P_i\}_{i\in Ags}$ of protocols for the individual agents. A joint action $a \in Acts$ is *enabled by P at a state s* if $a_i \in en(P_i, s)$ for all $i \in Ags$. We write $en(P, s)$ for the set of all joint actions enabled by P at state s.

Given an environment $E = \langle S, I, \{Acts\}_{i\in Ags}, \longrightarrow, \{O_i\}_{i\in Ags}, \pi\rangle$ and a joint protocol P for the agents in E, we may construct an interpreted system $\mathcal{I}(E,P) = (\mathcal{R}(E,P), \sim, \pi)$ over global states S as follows. The set of runs $\mathcal{R}(E,P)$ consists of all runs $r : \mathbf{N} \to S$ such that $r(0) \in I$ and for all $n \in \mathbf{N}$ there exists $\mathbf{a} \in en(P, r(n))$ such that $r(n) \xrightarrow{\mathbf{a}} r(n+1)$. The component $\sim = \{\sim_i\}_{i\in Ags}$ is defined by $(r,m) \sim_i (r',m')$ if $O_i(r(m)) = O_i(r'(m'))$, i.e., two points are indistinguishable to agent i if it makes the same observation at the corresponding global states; this is known in the literature as the *observational* semantics for knowledge. The interpretation π in the interpreted system $\mathcal{I}(E,P)$ is identical to that in the environment E.

Note that in $\mathcal{I} = \mathcal{I}(E,P)$, the satisfaction of formulas of the form $K_i\phi$ in fact depends only on the observation $O_i(r(m))$. We therefore may write $\mathcal{I}, o \models K_i\phi$ for an observation value o to mean $\mathcal{I}, (r,m) \models K_i\phi$ for all points (r,m) of $\mathcal{I}$ with $O_i(r(m)) = o$.

3 Epistemic Protocol Specifications

Protocol templates generalize concrete protocols by introducing some variables that may be instantiated with local boolean formulas in order to obtain a concrete protocol. Formally, a *protocol template* for agent $i \in Ags$ is an expression in the same form as (1), except that the ϕ_j are now boolean expressions, not just over the local atomic propositions $Prop_i$, but may also contain boolean variables from an additional set X of *template variables*. We write $Vars(Prot_i)$ for the set of these additional boolean variables that occur in some ϕ_i.

An *epistemic protocol specification* is a tuple $\mathcal{S} = \langle Ags, E, \{\mathtt{P}_i\}_{i\in Ags}, \Phi\rangle$, consisting of a set of agents Ags, an environment E for Ags, a collection of protocol templates $\{\mathtt{P}_i\}_{i\in Ags}$ for environment E, and a collection of epistemic logic formulas Φ over the agents Ags and atomic propositions $X \cup Prop$. In this paper, we assume $\Phi \subseteq \mathrm{CTLK}(Ags, X \cup Prop)$. We require that $Vars(\mathtt{P}_i)$ and $Vars(\mathtt{P}_j)$ are disjoint when $i \neq j$.

Intuitively, the protocol templates in such a specification lay out the abstract structure of some concrete protocols, and the variables in X are "holes" that need to be filled in order to obtain a concrete protocol. The formulas in Φ state constraints on how the holes may be filled: it is required that these formulas be valid in the model that results from filling the holes.

To implement an epistemic protocol specification with respect to the observational semantics, we need to replace each template variable v in each agent i's protocol template by an expression over the agent's local variables, in such a way that the specification formulas are satisfied in the model resulting from executing the resulting standard program. We now formalize this semantics.

Let θ be a substitution mapping each template variable $x \in \mathit{Vars}(\mathtt{P}_i)$, for $i \in \mathit{Ags}$, to a boolean formula local to agent i. We may apply such a substitution to a protocol template $\mathtt{P}_i$ in the form (1) by applying θ to each of the formulas ϕ_j, yielding

$$\textbf{do } \phi_1\theta \to a_1 \ [] \ \ldots \ [] \ \phi_k\theta \to a_k \ \textbf{ od}$$

which we write as $\mathtt{P}_i\theta$. Since the $\phi_j\theta$ contain only propositions in Prop_i, this is a concrete protocol for agent i. Consequently, we obtain a joint concrete protocol $\mathtt{P}\theta = \{\mathtt{P}_i\theta\}_{i\in\mathit{Ags}}$, which may be executed in the environment E, generating the system $\mathcal{I}(E, \mathtt{P}\theta)$. The substitution θ may also be applied to the specification formulas in Φ. Each $\phi \in \Phi$ is a formula over variables $X \cup \mathit{Prop}$, so $\phi\theta$ is a formula over variables Prop. We write $\Phi\theta$ for $\{\phi\theta \mid \phi \in \Phi\}$. We say that such a substitution θ provides an *implementation* of the epistemic protocol specification $\mathcal{S}$, provided $\mathcal{I}(E, \{\mathtt{P}_i\theta\}_{i\in\mathit{Ags}}) \models \Phi\theta$. The problem we study in this paper is the following: given an environment E and an epistemic protocol specification $\mathcal{S}$, synthesize an implementation θ.

Knowledge-based programs [8, 9] are a special case of epistemic protocol specifications. Essentially, knowledge-based programs are epistemic protocol specifications in which the set Φ is a collection of formulas of the form $AG(x \Leftrightarrow K_i\psi)$, with exactly one such formula for each agent $i \in \mathit{Ags}$ and each template variable $x \in \mathit{Vars}(\mathtt{P}_i)$. That is, each template variable is associated with a formula of the form $K_i\psi$, expressing some property of agent i's knowledge, and we require that the meaning of the template variable be equivalent to this property. The following example, an extension of an example from [4], illustrates the motivations for knowledge-based programs that have been advocated in the literature.

Example 1 *Two robots, A and B, sit on linear track with discretized positions* $0 \ldots 10$*. Initially A is at position* 0 *and B is at position* 10*. Their objective is to meet at a position at least* 2*, without colliding. Each robot is equipped with noisy position sensor, that gives at each moment of time a natural number value in the interval* $[0, \ldots, 10]$*. (We consider various different sensor models below, each defined by a relationship between the sensor reading and the actual position.) The robots do not have a sensor for detecting each other's position. Each robot has an action* **Halt** *and an action* **Move***. The* **Halt** *action brings the robot to a stop at its current location, and it will not move again after this action has been performed. The* **Move** *action moves the robot in the direction that it is facing (right, i.e., from 0 to 10 for A, and left for B). However, the effects of this action are unreliable: when performed, the robot either stays at its current position or moves one step in the designated direction.*

Because of the nondeterminism in the sensor readings and the robot motion, it is a non-trivial matter to program the robots to achieve their goal. In particular, the programmer needs to reason about how the sensor readings are related to the actual positions, in view of the assumptions about the possible robot motions. However, there is a natural abstract description of the solution to the problem at the level of agent knowledge, which we may capture as a knowledge-based program as follows: A has the epistemic protocol specification

$$\begin{array}{ll} \mathtt{P}_A = & \textbf{do} \\ & \quad \neg x \to \textbf{Move} \\ & \quad [] \ x \ \to \textbf{Halt} \\ & \textbf{od} \end{array}$$

$$AG(x \Leftrightarrow K_A(\mathit{position}_A \geq 2))$$

and B has the epistemic protocol specification

$$\begin{array}{ll} \mathtt{P}_B = & \textbf{do} \\ & \quad y \rightarrow \textbf{Move} \\ & \quad [] \ \neg y \rightarrow \textbf{Halt} \\ & \textbf{od} \end{array}$$

$$AG(y \Leftrightarrow K_B(\bigwedge_{p\in[0,\ldots 10]} position_B = p \Rightarrow AG(position_A < p-1)))$$

Intuitively, the specification for A says that A should move to the right until it knows that its position is at least 2. The specification for B says that B should move to the left so long as its knows that, if its current position is p, then A's position will always be to the left of the position $p-1$ *that a move might cause B to enter. If this does not hold then there could be a collision.*

One of the benefits of knowledge-based programs is that they can be shown to guarantee correctness properties of solutions for a problem independently of the way that knowledge is acquired and represented. This gives a desirable level of abstraction that enables a single knowledge level description to be used to generate multiple implementations that are tailored to different environments.

In the case of the above knowledge-based program, we note that it guarantees several properties independently of the details of the sensor model. Informally, since A halts only when it knows that its position is at least 2, and $K_A p \Rightarrow p$ *is a tautology of the logic of knowledge, its program ensures that when A halts, its position will be at least 2. Similarly, since B moves at most one position in any step, and moves only when it knows that moving to the position to its left will not cause a collision with A, a move by B will not be the cause of a collision. It remains to show that A does not cause a collision with B — this requires assumptions about A's sensor. (Note that if A is blind it never halts, and could collide with B even if B never moves, so assumptions are needed.) For termination, moreover, we require fairness assumptions about the way that A and B move (e.g., an action* **Move** *performed infinitely often eventually causes the position to change.).*

What implementations exist for the knowledge-based program depend on the assumptions we make about the error in the sensor readings. We assume that for each agent i, and possible sensor value v, there are propositions $sensor_i = v$, $sensor_i \geq v$, *and* $sensor_i \leq v$ *in* $Prop_i$, *with the obvious meaning. Suppose that we take the robots' position sensor to be free of error, i.e. for each agent i, we always have* $sensor_i = position_i$. *Then agent i always knows its exact position from its sensor value. In this case, the knowledge-based program has an implementation with* $\theta(x)$ *is* $sensor_A = 2$ *and* $\theta(y)$ *is* $sensor_B \geq 4$. *In this implementation, A halts at position 2 and B halts at position 3 (assuming that they reach these positions.)*

On the other hand, suppose that the sensor readings may be erroneous, with a maximal error of 1, i.e., when the robot's position is p, the sensor value is in $\{p-1, p, p+1\}$. *In this case, there exists an implementation* θ *in which* $\theta(x)$ *is* $sensor_A = 3 \vee sensor_A = 4 \vee sensor_A = 5$, *and* $\theta(y)$ *is* $sensor_B = 4 \vee sensor_B = 5 \vee sensor_B = 6$. *In this implementation, A moves until it gets a sensor reading in the set* $\{3,4,5\}$, *and then halts. The effect is that A halts at a location in the set* $\{2,3,4\}$*; which one depends on the pattern of sensor readings obtained. For example, the sequence* $(0,0),(1,1),(2,2),(3,2),(4,3)$ *of* $(position, sensor)$ *values leaves A at position 4, whereas the sequence* $(0,0),(1,1),(2,3)$ *leaves A at position* 2. *The effect of the choice of* $\theta(y)$ *is that B moves to the left and halts in one of the positions* $\{5,6,7\}$. *One run in which B halts at position* 5 *has* $(position, sensor)$ *values* $(10,10),(9,9),(8,8),(7,7),(6,7),(5,4)$. *A run in which B halts at position* 7 *is where these values are* $(10,10),(9,9),(8,8),(7,6)$. *Note that here the sensor reading 6 tells B that it is in the interval* $[5,7]$, *so it could be at* 5. *It is therefore not safe to move, since A might be at* 4.

P_i = **do**
  $start \wedge x_i \rightarrow \mathbf{c}$
 [] $start \wedge \neg x_i \rightarrow \mathbf{w}$
 [] $\neg start \rightarrow \mathbf{p}$
od

$AG(x_i \Leftrightarrow K_i AXw)$

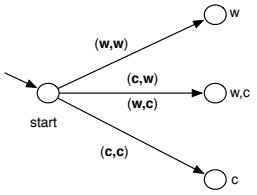

Figure 1: Knowledge-based program and environment

One of the advantages of the knowledge-based programs is that their implementations are optimal in the way that they use the information encoded in the agent's observations. For example, the program for A says that A should halt as soon *as it knows that it is in the goal region. In the case of the sensor with noise at most 1, the putative implementation for A given by* $\theta(x) = sensor_A \geq 4$ *would also ensure that A halts inside the goal region* [2,10]*, but would not implement the knowledge-based program because there are situations (viz.* $sensor_A = 3$*), where A does not halt even though it knows that it is safe to halt.*

The semantics for knowledge-based programs results in implementations that are highly optimized in their use of information. Because knowledge for an implementation θ is computed in the system $\mathcal{I}(E, P\theta)$, agent's may reason with complete information about the implementation they are running in determining what information follows from their observations. This introduces a circularity that makes finding implementations of knowledge-based programs an inherently complex problem. Indeed, it also has the consequence that it is possible for a knowledge-based program to have no implementations. The following provides a simple example where this is the case. It also illustrates a somewhat counterintuitive aspect of knowledge-based programs, that we will argue is improved by our proposed ordered semantics for epistemic specifications below.

Example 2 *Alice and Bob have arranged to meet for a picnic. They are agreed that a picnic should have both wine and cheese, and each should bring one or the other. However, they did not think to coordinate in advance what each is bringing, and they are now not able to communicate, since Alice's phone is in the shop for repairs. They do know that each reasons as follows. Cheese being cheaper than wine, they prefer to bring cheese, and will do so if they know that there is already guaranteed to be wine. Otherwise, they will bring wine. This situation can be captured by the knowledge-based program (for each* $i \in \{A, B\}$*) and environment depicted in Figure 1. Here start is a proposition, local to both agents, that holds before the picnic (at time 0). We use* w, c *as propositions that hold if there is wine (respectively, cheese) in the picnic state (at time 1). Actions* $\mathbf{w}, \mathbf{c}, \mathbf{p}$ *represent bringing wine, bringing cheese, and picnicking, respectively. For any omitted joint actions* $\mathbf{a}$ *from a state s in the diagram, we assume an implicit self-loop* $s \xrightarrow{\mathbf{a}} s$*. We assume that for all states s and* $i \in \{A, B\}$*, we have* $O_i(s) = s$*, i.e., both agents have complete information about the current state.*

This epistemic specification has no implementations. Note that in any implementation, each agent i must choose either $\mathbf{w}$ *or* $\mathbf{c}$ *at the initial state. For each such selection, there is a unique successor state at time 1, so each implementation system* $\mathcal{I}(\mathrm{P}\theta, E)$ *has exactly one state at time 1. If this state satisfies w, then we have* $\mathcal{I}(\mathrm{P}\theta, E) \models K_i(AXw)$*, and this implies that both agents select action* $\mathbf{c}$ *at the start state. But then the state at time 1 does not satisfy w. Conversely, if the unique state at time* 1 *does not satisfy w, then* $\mathcal{I}(\mathrm{P}\theta, E) \models \neg K_i(AXw)$*, and this implies that both agents select action* $\mathbf{w}$ *at the start state, which produces a state at time 1 that satisfies w, also a contradiction. In either case, the assumption that we have an implementation results in a contradiction, so there are no implementations.* □

Testing whether there exists an implementation of a knowledge-based program when the temporal basis of the temporal epistemic logic used is the linear time logic LTL is PSPACE complete [9]. However, the primary source of the hardness here is that model checking LTL is already a PSPACE complete problem.

In the case of CTL as the temporal basis, where model checking can be done in PTIME, the problem of deciding the existence of an implementation of a given knowledge-based program in a given environment can be shown to be NP-complete. NP hardness follows from Theorem 5.4 in [9], which states that for *atemporal* knowledge-based programs, in which the knowledge formulas $K_i\phi$ used do not contain temporal operators, the complexity of determining the existence of an implementation is NP-complete. However, the construction in the proof in [9] requires both the environment and the knowledge-based program to vary. In practice, the size of the knowledge-based program is likely to be significantly smaller than the size of the environment, inasmuch as it is created by hand and effectively amounts to a form of specification. An alternate approach is to measure complexity as a function of the size of the environment for a fixed knowledge-based program. Even here, it turns out, the problem of deciding the existence of an implementation is NP-hard for very simple knowledge-based programs.

Theorem 1 *There exists a fixed atemporal knowledge-based program* P *for a single agent, such that the problem of deciding, given an environment E, whether* P *has an implementation in E, is NP-hard.*

The upper bound of NP for deciding the existence of implementations of knowledge-based programs is generalized by the following result for our more general notion of epistemic protocol specification.

Theorem 2 *Given an environment E and an epistemic protocol specification $\mathcal{S}$ expressed using CTLK, the complexity of determining the existence of an implementation for $\mathcal{S}$ in E is in NP.*

Theorem 2 assumes that the environment is presented by means of an explicit listing of its states and transitions. In practice, the inputs to the problem will be given in some format that makes their representation succinct, e.g., states will be represented as assignments to some set of variables, and boolean formulas will be used to represent the environment and protocol components. For this alternate input format, the problem of determining the existence of an implementation of a given epistemic protocol specification is NEXPTIME-complete [14].

Under either an implicit or explicit representation of environments, these results suggest that synthesis of implementations of general epistemic protocol specifications, and knowledge-based programs in particular, is unlikely to be practical. An implementation using symbolic techniques is presented in [14], but it works only on small examples and scales poorly (it requires the introduction of exponentially many fresh propositions before using BDD techniques; the number of propositions soon reaches the limit that can be handled efficiently by BDD packages.) In the following section, we consider a restricted class of specifications that weakens the notion of knowledge-based program in such a way that implementations can always be found, and focus on how to efficiently derive implementations that approximate the implementations of corresponding knowledge-based programs as closely as possible.

4 An Ordered Semantics

Sound local proposition epistemic protocol specifications are a generalization of knowledge-based programs, introduced in [7], with one of the motivations being that they provide a larger space of potential implementations, that may overcome the problem of the high complexity of finding an implementation. (There is the further motivation that the implementation of a knowledge-based program, when one exists,

itself may be intractable; e.g., it is shown in [19] that for perfect recall implementations of *atemporal* knowledge-based programs, deciding whether $K_i\phi$ holds at a given point of the implementation may be a PSPACE-complete problem. This specific motivation is less of concern for the observational case that we study in this paper.)

Formally, a *sound local proposition* epistemic protocol specification is one in which Φ is given by means of a function κ with domain $Vars(\mathsf{P})$, such that for each agent i and each template variable $x \in Vars(\mathsf{P}_i)$, the formula $\kappa(x)$ is of the form $K_i\psi$. The corresponding set of formulas for the epistemic protocol specification is $\Phi = \Phi_\kappa = \{AG(x \Rightarrow \kappa(x)) \mid x \in Vars(\mathsf{P})\}$.

As usual for epistemic protocol specifications, an implementation associates to each template variable a boolean formula local to the corresponding agent, such that the resulting system satisfies the specification Φ.[1] Thus, whereas a knowledge-based program requires that each knowledge formula in the program be implemented by a *necessary and sufficient* local formula, a sound local proposition specification requires only that the implementing local formula be *sufficient*.

It is argued in [7] that examples of knowledge-based programs can typically be weakened to sound local proposition specifications without loss of the desired *correctness* properties that hold of all implementations. However, implementations of knowledge-based programs may guarantee *optimality* properties that are not guaranteed by the corresponding sound local proposition specifications. For example, an implementation of a knowledge-based program that states "if $K_i\phi$ then do a" will be optimal in the sense that it ensures that the agent will do a *as soon as* it knows that ϕ holds. By contrast, an implementation that replaces $K_i\phi$ by a sufficient condition for this formula may perform a only much later, or even fail to do so, even if the knowledge necessary to do a is deducible from the agent's local state. (An example of such a situation is given in [1], which identifies a situation where a cache coherency protocol fails to act on knowledge that it has.)

Note that the substitution $\theta_\perp$, defined by $\theta_\perp(x) = \mathbf{false}$ for all template variables x, is *always* an implementation for a sound local proposition specification $\mathcal{S}$ in an environment E. It is therefore trivial to decide the existence of an implementation, and it is also trivial to produce a succinct representation of an implementation. Of course, an implementation of a program "if x then do a" that sets x to be **false** will never perform a, so this trivial implementation is generally not of much interest. What is more interesting is to find *good* implementations, that approximate the corresponding knowledge-based program implementations as closely as possible in order to behave as close to optimally as possible, while remaining tractable.

Consider the order on substitutions defined by $\theta \leq \theta'$ if for all variables x and states $s \in S$ of the environment we have $\pi(s) \models \theta(x) \Rightarrow \theta'(x)$. If both are implementations of $\mathcal{S}$ in E, we may find θ' preferable in that it provides weaker sufficient conditions (i.e., ones more often true) for the knowledge formulas $K_i\phi$ of interest. Pragmatically, if ϕ is a condition that an agent must know to be true before it can safely perform a certain action, the more often the sufficient condition $\theta(x)$ for $K_i\phi$ holds, the more often will the agent perform the action in the implementation. It is therefore reasonable to seek implementations that maximize θ with respect to the order $\leq$. The maximal sufficient condition for $K_i\phi$ is $K_i\phi$ itself, in the system $\mathcal{I}(E, P\theta)$ corresponding to an implementation θ, expressed as an equivalent local formula.[2]

The following result makes this statement precise:

[1] By the assumption of locality of $\theta(x)$, validity of $AG(\theta(x) \Rightarrow K_i\psi)$ in a system is equivalent to validity of $AG(\theta(x) \Rightarrow \psi)$, but we retain the epistemic form for emphasis and to maintain the connection to knowledge-based programs.

[2] The existence of such a formula follows from completeness of the set of local propositions. If we extend the propositions in an environment to include for each agent i and possible observation o of the agent, a proposition $p_{i,o}$ that holds at a state s iff $O_i(s) = o$, then the formula $\theta(x)$ such that $\mathcal{I} \models AG(\theta(x) \Leftrightarrow \kappa(x))$, where $\kappa(x) = K_i\phi$, can be constructed as $\bigvee\{p_{i,o} \mid o \in O_i(S), \mathcal{I}, o \models \kappa(x)\}$, and has size of order the number of observations.

Theorem 3 *Suppose that $\mathcal{S}$ is a sound local proposition epistemic protocol specification, and let $\mathcal{S}'$ be the knowledge-based program resulting from replacing each formula $AG(x \Rightarrow \kappa(x))$ in Φ by the formula $AG(x \Leftrightarrow \kappa(x))$. Then every implementation θ of $\mathcal{S}'$ is an implementation of $\mathcal{S}$.*

However, to have $\theta(x)$ equivalent to $K_i\phi$ in $\mathcal{I}(E,P\theta)$ would mean that θ implements a knowledge-based program. The complexity results of the previous section indicate that this is too strong a requirement, for practical purposes, since it is unlikely to be efficiently implementable. The compromise we explore in this paper is to require $\theta(x)$ to be equivalent to $K_i\phi$ not in the system $\mathcal{I}(E,P\theta)$ itself, but in another system that approximates $\mathcal{I}(E,P\theta)$. The basis for the correctness of this idea is the following lemma.

Lemma 1 *Suppose that $\mathcal{I} \subseteq \mathcal{I}'$, that r is a run of $\mathcal{I}$ and that ϕ is a formula in which knowledge operators and the branching operator A occur only in positive position. Then $\mathcal{I}', (r,m) \models \phi$ implies $\mathcal{I}, (r,m) \models \phi$.*

In particular, if, for a sound local proposition epistemic protocol specification $\mathcal{S}$, the formula $\kappa(x)$ associated to a template variable x is in CTLK$^+$, then this result applies to the formula $AG(x \Rightarrow \kappa(x))$ in Φ_κ, since this is also in CTLK$^+$. Suppose the system $\mathcal{I}'$ approximates the ultimate implementation $\mathcal{I}(E,P\theta)$ in the sense that $\mathcal{I}' \supseteq \mathcal{I}(E,P\theta)$. Let $\theta(x)$ be a local formula such that $\mathcal{I}' \models AG(\theta(x) \Leftrightarrow \kappa(x))$. Then also $\mathcal{I}' \models AG(\theta(x) \Rightarrow \kappa(x))$, hence, by Lemma 1, $\theta(x)$ will also satisfy the correctness condition $\mathcal{I}(E,P\theta) \models AG(\theta(x) \Rightarrow \kappa(x))$ necessary for θ to be an implementation of $\mathcal{S}$.

Our approach to constructing good implementations of $\mathcal{S}$ will be to compute local formulas $\theta(x)$ that are equivalent to $\kappa(x)$ in approximations $\mathcal{I}'$ of the ultimate implementation being constructed. We take this idea one step further. Suppose that we have used this technique to determine the value of $\theta(x)$ for some of the template variables x of $\mathcal{S}$. Then we have increased our information about the final implementation θ, so we are able to construct a *better* approximation $\mathcal{I}''$ to the final implementation $\mathcal{I}(E,P\theta)$, in the sense that $\mathcal{I}' \supseteq \mathcal{I}'' \supset \mathcal{I}(E,P\theta)$. Note that if $\mathcal{I}' \models AG(\phi' \Leftrightarrow \kappa(y))$ and $\mathcal{I}'' \models AG(\phi'' \Leftrightarrow \kappa(y))$, then it follows from $\mathcal{I}' \supseteq \mathcal{I}''$ that $\mathcal{I}'' \models AG(\phi' \Rightarrow \phi'')$. That is, ϕ'' is weaker than ϕ', and hence a better approximation to the knowledge condition $\kappa(y)$ in the ultimate implementation $\mathcal{I}(E,P\theta)$. Thus, by proceeding iteratively through the template variables, and improving the approximation as we construct a partial implementation, we are able to obtain better approximations to $\kappa(y)$ in $\mathcal{I}(E,P\theta)$ for later variables.

More precisely, suppose that we have a total pre-order on the set of all template variables $Vars(\mathrm{P}) = \cup_{i \in Ags} Vars(\mathrm{P}_i)$, i.e., a binary relation $\leq$ on this set that is transitive and satisfies $x \leq y \vee y \leq x$ for all $x, y \in Vars(\mathrm{P})$. Let this be represented by the sequence of subsets $X_1, \ldots, X_k$, where for $i \leq j$ and $x \in X_i$ and $y \in X_j$ we have $x < y$ if $i < j$ and $x \leq y \leq x$ if $i = j$. Suppose we have a sequence of interpreted systems $\mathcal{I}_0 \supseteq \ldots \supseteq \mathcal{I}_k$. Define a substitution θ to be *consistent* with this sequence if for all $i = 1 \ldots k$ and $x \in X_i$, we have $\mathcal{I}_{i-1} \models AG(\theta(x) \Leftrightarrow \kappa(x))$. That is, consistent substitutions associate to each template variable x a local formula that is equivalent to (not just sufficient for) $\kappa(x)$, but in an associated approximation system rather than in the final implementation.

Proposition 1 *Suppose that $\mathcal{I}_k$ is isomorphic to $\mathcal{I}(E,P\theta)$, and that for all $x \in Vars(\mathrm{P})$, the formula $\kappa(x)$ contains knowledge operators and the branching operator A only in positive position. Then θ implements the epistemic protocol specification $\langle Ags, E, P, \Phi_\kappa \rangle$.*

We will apply this result as follows: define an *approximation scheme* to be a mapping that, given an epistemic protocol specification $\mathcal{S} = \langle Ags, E, P, \Phi \rangle$ and a partial substitution θ for $\mathcal{S}$, yields a system $\mathcal{I}(\mathcal{S}, \theta)$, satisfying the conditions

1. if $\theta \subseteq \theta'$ then $\mathcal{I}(\mathcal{S}, \theta) \supseteq \mathcal{I}(\mathcal{S}, \theta')$, and
2. if θ is total, then $\mathcal{I}(\mathcal{S}, \theta)$ is isomorphic to $\mathcal{I}(E, P\theta)$.

Assume now that $\mathcal{S}$ is a sound local proposition specification based on the mapping κ. Given the ordering $\leq$ on $Vars(\mathtt{P})$, with the associated sequence of sets $X_1 \ldots X_k$, we define the sequence $\theta_0, \theta_1, \ldots, \theta_k$ inductively by $\theta_0 = \emptyset$ (the partial substitution that is nowhere defined), and θ_{j+1} to be the extension of θ_j obtained by defining, for $x \in X_{j+1}$, the value of $\theta_{j+1}(x)$ to be the local proposition ϕ such that $\mathcal{I}(\mathcal{S}, \theta_j) \models AG(\phi \Leftrightarrow \kappa(x))$. Plainly $\theta_0 \subseteq \theta_1 \subseteq \ldots \subseteq \theta_k$, so we have $\mathcal{I}(\mathcal{S}, \theta_0) \supseteq \mathcal{I}(\mathcal{S}, \theta_1) \supseteq \ldots \supseteq \mathcal{I}(\mathcal{S}, \theta'_k)$. It follows from the properties of the approximation scheme and Proposition 1 that the substitution θ_k is total and is an implementation of $\mathcal{S}$.

This idea leads to an extension of the idea of epistemic protocol specifications: we now consider specifications of the form $(\mathcal{S}, \leq)$, where $\mathcal{S}$ is a sound local proposition epistemic protocol specification, and $\leq$ is a total pre-order on the template variables of $\mathcal{S}$. Given an approximation scheme, the construction of the previous paragraph yields a unique implementation of $\mathcal{S}$. Intuitively, by specifying an order $\leq$, the programmer fixes the order in which implementations are synthesized for the template variables, and the approach guarantees that variables later in the order are synthesized using information about the values of variables earlier in the order.

5 A spectrum of approximations

It remains to determine which approximation scheme to use in the approach to constructing implementations described in the previous section. In this section, we consider a number of possibilities for the choice of approximation scheme. A number of criteria may be applied to the choice of approximation scheme. For example, since the programmer must select the order in which variables are synthesized, the approximation scheme should be simple enough to be comprehensible to the programmer, so that they may understand the consequences of their ordering decisions.

On the other hand, since synthesis is to be automated, we would like the computation of the values $\theta(x)$ to be efficient. This amounts to efficiency of the model checking problem $\mathcal{I}(\mathcal{S}, \theta') \models \kappa(x)$ for partial substitutions θ' and formulas $\kappa(x) \in \mathrm{CTLK}^+$. To analyze this complexity, we work below with a complexity measure that assumes explicit state representations of environments, but we look for cases where the model checking problem in the approximation systems is solvable in PTIME. We assume that the protocol template $\mathtt{P}$ and the formulas Φ in the epistemic protocol specification are fixed, and measure complexity as a function of the size of the environment E. This is because in practice, the size of the environment is likely to be the dominant factor in complexity.

One immediately obvious choice for the approximation scheme is to take the system $\mathcal{I}(\mathcal{S}, \theta)$, for a partial substitution θ, to be the union of all the systems $\mathcal{I}(E, P\theta')$, over all total substitutions θ' that extend the partial substitution θ. This turns out not to be a good choice (it is the intractable case $\mathcal{I}_{ii,ir,sc}$ below), so we consider a number of relaxations of this definition. The following abstract view of the situation provides a convenient format that unifies the definition of these relaxations.

Given an environment E with states S, define a *strategy* for E to be a function $\sigma : S^+ \to \mathcal{P}(S) \setminus \emptyset$ mapping each nonempty sequence of states to a set of possible successors. We require that for each $t \in \sigma(s_0 \ldots s_k)$ we have $s_k \xrightarrow{\mathbf{a}} t$ for some joint action $\mathbf{a}$. Given a set Σ of strategies, we can construct an interpreted system consisting of all runs consistent with some strategy in Σ. We encode the strategy into the run. We use the extended set of global states $S \times \Sigma$. We take $\mathcal{R}_\Sigma$ to be the set of all $r : \mathbf{N} \to S \times \Sigma$ such that there exists a strategy σ such that for all $n \in \mathbf{N}$ we have $r(n) = (s_n, \sigma)$, for some $s_n \in S$, and, we have $s_{n+1} \in \sigma(s_0 s_1 \ldots s_n)$ for all $n \in \mathbf{N}$. Intuitively, this is the set of all infinite runs, each using some fixed strategy in Σ, with the strategy encoded into the state. We define $\mathcal{I}(E, \Sigma) = (\mathcal{R}_\Sigma, \sim, \pi')$ where $\sim = \{\sim_i\}_{i \in Ags}$ is the relation on points of $\mathcal{R}_\Sigma$ defined by $(r, m) \sim_i (r', m')$ if, with $r(m) = (s, \sigma)$ and $r'(m') = (s', \sigma')$, we

have $O_i(s) = O_i(s')$. The interpretation π' on $S \times \Sigma$ is defined so that $\pi'(s,\sigma) = \pi(s)$, where $s \in S$, $\sigma \in \Sigma$ and π is the interpretation from E.

A *memory definition* is a collection of functions $\mu = \{\mu_i\}_{i \in Ags}$ with each μ_i having domain S^+. In particular, we work with the following memory definitions derived using the observation functions in the environment E:

- The *perfect information, perfect recall* definition $\mu^{pi,pr} = \{\mu_i^{pi,pr}\}_{i \in Ags}$ where $\mu_i^{pi,pr}(s_0 \dots s_k) = s_0 \dots s_k$
- The *perfect information, imperfect recall* definition $\mu^{pi,ir} = \{\mu_i^{pi,ir}\}_{i \in Ags}$ where $\mu_i^{pi,ir}(s_0 \dots s_k) = s_k$
- The *imperfect information, perfect recall* definition $\mu^{ii,pr} = \{\mu_i^{ii,pr}\}_{i \in Ags}$ where $\mu_i^{ii,pr}(s_0 \dots s_k) = O_i(s_0) \dots O_i(s_k)$
- The *imperfect information, imperfect recall* definition $\mu^{ii,ir} = \{\mu_i^{ii,ir}\}_{i \in Ags}$ where $\mu_i^{ii,ir}(s_0 \dots s_k) = O_i(s_k)$

A strategy *depends* on memory definition μ if there exist functions $F_i : range(\mu_i) \to \mathcal{P}(Acts_i)$ for $i \in Ags$ such that for all sequences $\rho = s_0 \dots s_k$, we have $t \in \sigma(s_0 \dots s_k)$ iff $s \xrightarrow{\mathbf{a}} t$ for some joint action $\mathbf{a}$ such that for all $i \in Ags$, we have $\mathbf{a}_i \in F_i(\mu_i(s_0 \dots s_k))$.

Let $\mathtt{P}$ be a joint protocol template and let θ be a partial substitution for $\mathtt{P}$. A strategy σ is *substitution consistent* with respect to $\mathtt{P}, \theta$ and a memory definition μ if σ depends on μ and for all sequences $s_0 \dots s_k$ there exists a substitution $\theta' \supseteq \theta$ mapping all the template variables of $\mathtt{P}$ undefined by θ to truth values, such that

$$\sigma(s_0 \dots s_k) = \{t \mid \text{there exists } \mathbf{a} \in en(P\theta', s_k),\ s_k \xrightarrow{\mathbf{a}} t\} \tag{2}$$

Note that since the choice of θ' is allowed to depend on $s_0 \dots s_k$, this does not imply that the set of possible successors states $\sigma(s_0 \dots s_k)$ depends only on the final state s_k; the reference to s_k in the right hand side of equation 2 is included just to allow the enabled actions to be determined in a way consistent with the substitution θ, which already associates some of the variables with predicates on the state s_k.

Example 3 *Consider the maximally nondeterministic, or* top*, strategy $\sigma_\top$, defined by $\sigma_\top(s_0 \dots s_k) = \{t \mid$ there exists $\mathbf{a} \in Acts, s_k \xrightarrow{\mathbf{a}} t\}$ for all $s_0 \dots s_k$. Intuitively, this strategy allows any action to be taken at any time. It is easily seen that $\sigma_\top$ depends on every memory definition μ. However, it is not in general substitution consistent, since there are protocol templates for which the set of enabled actions (and hence the transitions) depend on the substitution.*

Consider the protocol template $\mathtt{P} =$ **do** $x \to a$ [] $\neg x \to b$ **od** *for a single agent, in an environment with states $S = \{s_0, s_1, s_2\}$ and transitions $s_0 \xrightarrow{a} s_1$, $s_0 \xrightarrow{b} s_2$, $s_1 \xrightarrow{a,b} s_1$ and $s_2 \xrightarrow{a,b} s_2$. Let θ be the empty substitution. For all substitutions θ', $en(\mathtt{P}\theta', s_0)$ is either $\{a\}$ or $\{b\}$, so for the sequence s_0, the right hand side of equation (2) is equal to either $\{s_1\}$ or $\{s_2\}$. For the strategy $\sigma_\top$, we have $\sigma_\top(s_0) = \{s_1, s_2\}$. Hence this strategy is not substitution consistent in this environment.* □

We now obtain eight sets of strategies by choosing an information mode $a \in \{pi, ii\}$, a recall mode $b \in \{pr, ir\}$ and a selection $c \in \{sc, nsc\}$ to reflect a choice with respect to the requirement of substitution consistency. Formally, given a joint protocol template $\mathtt{P}$, a partial substitution θ for $\mathtt{P}$, and an environment E, we define $\Sigma^{a,b,c}(\mathtt{P}, \theta, E)$ to be the set of all strategies in E that depend on $\mu^{a,b}$, and that are substitution consistent with respect to P, θ and $\mu^{a,b}$ in the case $c = sc$.

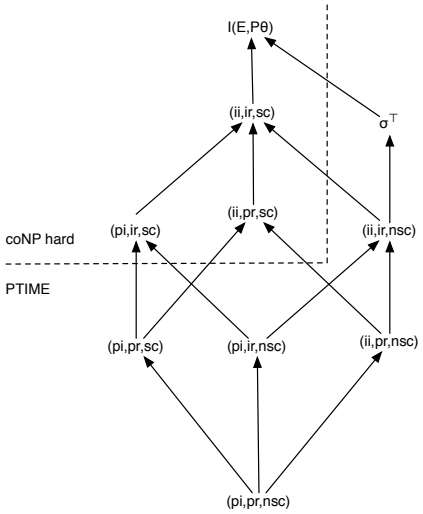

Figure 2: Lattice structure of the approximations

Corresponding to these eight sets of strategies, we obtain eight approximation schemes. Let $\mathcal{S}$ be an epistemic protocol specification with joint protocol template $\mathtt{P}$, and environment E. Given a partial substitution θ for $\mathtt{P}$, and a triple a, b, c, we define the system $\mathcal{I}_{a,b,c}(\mathcal{S}, \theta)$ to be $\mathcal{I}(\Sigma^{a,b,c}(P, \theta, E), E)$.

Proposition 2 *For each information mode $a \in \{pi, ii\}$, a recall mode $b \in \{pr, ir\}$ and selection $c \in \{sc, nsc\}$, the mapping $\mathcal{I}_{a,b,c}$ is an approximation scheme.*

Additionally we have the approximation scheme $\mathcal{I}^{\top}(\mathcal{S}, \theta)$ defined to be $\mathcal{I}(\{\sigma^{\top}_{E,P\theta}\}, E)$, based on the top strategy in E relative to the protocol template $\mathtt{P}\theta$, which is defined by taking $\sigma^{\top}_{E,\mathtt{P}\theta}(s_0 \ldots s_k)$ to be the set of all states $t \in S$ such that there exists a joint action $a \in Acts$ such that for all $i \in Ags$, the protocol template $\mathtt{P}_i\theta$ contains a clause $\phi\theta \rightarrow a_i$ with $\phi\theta$ satisfiable relative to $\pi(s_k)$. (We note that here $\pi(s_k)$ provides the values of propositions *Prop* and we are asking for satisfiability for some assignment to the variables on which θ is undefined. Because we are interested in the case where $\mathtt{P}$, and hence ϕ, is fixed, this satisfiability test can be performed in PTIME as the environment varies.)

For reasons indicated in Example 3, the strategy $\sigma^{\top}_{E,P\theta}$ is not substitution-consistent. However, it is easily seen to depend only on the values $O_i(s_k)$, so we have $\sigma^{\top}_{E,P\theta} \in \Sigma^{ii,ir,nsc}$.

Figure 2 shows the lattice structure of the approximation schemes, with an edge from a scheme $\mathcal{I}$ to a scheme $\mathcal{I}'$ meaning that $\mathcal{I}'$ is a closer approximation to the final system $\mathcal{I}(E, \mathtt{P}\theta)$ synthesized, informally in the sense that $\mathcal{I}$ has more runs and more branches from any point than does $\mathcal{I}'$. (Generally, the relation is one of simple containment of the sets of runs, but in the case of edges involving $\mathcal{I}(E, \mathtt{P}\theta)$ and $\sigma^{\top}$, we need a notion of simulation to make this precise.)

Besides yielding an approach to the construction of implementations of epistemic protocol specifications, we note that our approach also overcomes the counterintuitive aspect of knowledge-based programs illustrated in Example 2.

Example 4 *Suppose that we replace the specification formulas $AG(x_i \Leftrightarrow K_i AXw)$ in Example 2 by the weaker form $AG(x_i \Rightarrow K_i AXw)$, and impose the ordering $x_A < x_B$ on the template variables. We compute the implementation obtained when we use $\mathcal{I}^{\top}$ as the approximation scheme. We take θ_0 to be the empty substitution. $\mathcal{I}(\{\sigma^{\top}_{E,P\theta_0}\}, E)$ has all possible behaviours of the original environment, so at the start*

state, we have $\neg K_A(AXw)$*. It follows that substitution* θ_1*, which has domain* $\{x_A\}$ *assigns to* x_A *a local proposition that evaluates to* **false** *at the initial state. Hence,* $\mathtt{P}_A\theta_1$ *selects action* **w** *at the initial state. The effect of this is to delete the bottom transition from the state transition diagram for the environment in Figure 1. It follows that in* $\mathcal{I}(\{\sigma^{\top}_{E,P\theta_1}\},E)$*, we have* $K_B(AXw)$ *at the initial state, so* $\theta_2(x_B)$ *evaluates to* **true** *at the initial state. This means that the final implementation* $\mathtt{P}\theta_2$ *is the protocol in which Alice brings wine and Bob brings cheese, leading to a successful picnic, by contrast with the knowledge-based program, which does not yield any solutions to their planning problem. (We remark that both Alice and Bob could compute this implementation independently, once given the ordering on the variables. They do not need to communicate during the computation of the implementation.)* □

We noted above in Theorem 3 that a sound local proposition specification that is obtained from a knowledge-based program includes amongst its implementations all the implementations of the knowledge-based program. The knowledge-based program, in effect, imposes additional optimality constraints on these implementations. Our ordered semantics aims to approximate these optimal implementations. It is therefore of interest to determine whether the ordered semantics for sound local proposition specifications can sometimes find such optimal implementations. Although it is not true in general, there are situations where the implementations obtained are indeed optimal. The following provides an example.

Example 5 *Consider the sound local proposition specification obtained from the knowledge-based program of Example 1 by replacing the* $\Leftrightarrow$ *operators in the formulas by* $\Rightarrow$*. That is, we take* Φ *to contain the formulas*

$$AG(x \Rightarrow K_A(position_A \geq 2))$$

and

$$AG(y \Rightarrow K_B(\bigwedge_{p\in[0,\ldots 10]} position_B = p \Rightarrow AG(position_A < p-1)))$$

We consider the setting where sensors readings are within 1 of the actual position. Suppose that we use $\mathcal{I}^{\top}$ *as the approximation scheme, and order the template variables using* $x < y$*, i.e., we synthesize a solution for A before synthesizing a solution for B (knowing what A is doing.) Then, for A, we construct* $\theta(x)$ *as the local proposition for A that satisfies*

$$AG(x \Leftrightarrow K_A(position_A \geq 2))$$

in a system where both A and B may choose either action **Move** *or* **Halt** *at any time. We obtain the substitution* θ_1 *where* $\theta_1(x)$ *is* $sensor_A \geq 3$*, which ensures that always* $position_A \leq 4$*, and in which A may halt at a position in the set* $\{2,3,4\}$*. In the next step, we synthesize* $\theta(y)$ *as the local proposition such that*

$$AG(y \Leftrightarrow K_B(\bigwedge_{p\in[0,\ldots 10]} position_B = p \Rightarrow AG(position_A < p-1)))$$

in the system where A runs $\mathtt{P}_A\theta_1$*, and where B may choose either action* **Move** *or* **Halt** *at any time. In this system, B knows that A's position is always at most* 4*, so it is safe for B to move if* $position_B \geq 6$*. Agent B knows that its position is at least 6 when it gets a sensor reading at least 7. Hence, we obtain the substitution* θ_2 *where* $\theta_2(y)$ *is* $sensor_B \geq 7$ *and* $\theta_2(x)$ *is* $sensor_A \geq 3$*. It can be verified that this substitution is in fact an implementation of the original knowledge-based program.*

6 Complexity of model checking in the approximations

To construct an implementation based on the extended epistemic protocol specification $(\mathcal{S}, \leq)$ using an approximation scheme $\mathcal{I}(\mathcal{S}, \theta)$, we need to perform model checking of formulas in CTLK$^+$ in the systems produced by the approximation scheme. We now consider the complexity of this problem for the approximation schemes introduced in the previous sections. We focus on the complexity of this problem with the protocol template fixed as we vary the size of the environment, for reasons explained above.

We say that the *environment-complexity* of an approximation scheme $\mathcal{I}(\mathcal{S}, \theta)$ is the maximal complexity of the problem of deciding $\mathcal{I}(\mathcal{S}, \theta), o \models \kappa(x)$ with all components fixed and only the environment E in $\mathcal{S}$ varying. More precisely, write $\mathcal{S}^- = \langle Ags, \mathtt{P}, \kappa \rangle$ for a tuple consisting of a set Ags of agents, a collection $\mathtt{P} = \{\mathtt{P}_i\}_{i \in Ags}$ of protocol templates for these agents, and a mapping κ associating, for each agent i, a formula $\kappa(x) = K_i\phi$ of CTLK$^+$ to each template variable x in $\mathtt{P}_i$. Given an environment E, write $\mathcal{S}^-(E)$ for the epistemic protocol specification $\langle Ags, E, \{\mathtt{P}_i\}_{i \in Ags}, \Phi_\kappa \rangle$ obtained from these components. Say that E *fits* a tuple $(\mathcal{S}^-, \theta, o, x)$ consisting of $\mathcal{S}^-$ as above, a substitution θ assigning a boolean formula to a subset of the template variables in $\mathtt{P}$, an observation o and a variable x, if E contains all actions used in $\mathtt{P}$, o is an observation in E of the agent i such that $\mathtt{P}_i$ contains x, and for each x such that $\theta(x)$ is defined, the formula $\theta(x)$ is local in E to the agent i such that $\mathtt{P}_i$ contains x. Given $\mathcal{S}^- = \langle Ags, \mathtt{P}, \kappa \rangle$ and θ, o and x, define $EC_{(\mathcal{S}^-, \theta, o, x)}$ to be the set

$$\{E \mid E \text{ fits } (\mathcal{S}^-, \theta, o, x) \text{ and } \mathcal{I}(\mathcal{S}^-(E), \theta), o \models \kappa(x)\} .$$

Then the environment-complexity of an approximation scheme $\mathcal{I}(\mathcal{S}, \theta)$ is the maximal complexity of the problem of deciding the sets $EC_{(\mathcal{S}^-, \theta, o, x)}$ over all choices of $\mathcal{S}^-$, θ, o and x .

We note that even though we have allowed perfect recall and/or perfect information in the strategy spaces used by the approximation, when we model check in the system generated by the approximation, knowledge operators are handled using the usual observational (imperfect recall, imperfect information) semantics. The stronger capabilities of the strategies are used to increase the size of the strategy space in order to weaken the approximation. (Model checking with respect to perfect recall, in particular, would *increase* the complexity of the model checking problem, whereas we are seeking to decrease its complexity.)

It turns out that several of the approximation schemes, that are closest to the final system synthesized (which would give the knowledge-based program semantics), share with the knowledge-based program semantics the disadvantage of being intractable. These are given in the following result.

Theorem 4 *The approximation schemes $\mathcal{I}_{ii,ir,sc}$, $\mathcal{I}_{ii,pr,sc}$, and $\mathcal{I}_{pi,ir,sc}$ have coNP-hard environment complexity, even for a single agent.*

Each of these intractable cases uses substitution consistent strategies and uses either imperfect recall or imperfect information. The proofs vary, but one of the key reasons for complexity in the imperfect recall cases is that the strategy must behave the same way each time it reaches a state. Intuitively, this means that we can encode existential choices from an NP hard problem using the behaviour of a strategy at a state in this case. In the case of $\mathcal{I}_{ii,pr,sc}$, we use obligations on multiple branches indistinguishable to the agent to force consistency of independent guesses representing the same existential choice. All the remaining approximation schemes, it turns out, are tractable:

Theorem 5 *The approximation schemes $\mathcal{I}^\top$, $\mathcal{I}_{ii,ir,nsc}$, $\mathcal{I}_{pi,pr,sc}$, $\mathcal{I}_{pi,ir,nsc}$, $\mathcal{I}_{ii,pr,nsc}$ and $\mathcal{I}_{pi,pr,nsc}$ have environment complexity in PTIME.*

The reasons are varied, but there are close connections to some known results. The scheme $\mathcal{I}^{\top}$ effectively builds a new finite state environment from the environment and protocol by allowing some transitions that would normally be disabled by the protocol, so its model checking problem reduces to an instance of CLTK model checking, which is in PTIME by a mild extension of the usual CTL model checking approach. It turns out, moreover, by simulation arguments, that for model checking CTLK^{+} formulas, the approximations $\mathcal{I}_{ii,ir,nsc}$ and $\mathcal{I}_{ii,pr,nsc}$ are equivalent to $\mathcal{I}^{\top}$, i.e., satisfy the same formulas at the same states, so the algorithm for $\mathcal{I}^{\top}$ also resolves these cases.

The cases $\mathcal{I}_{pi,pr,sc}$ and $\mathcal{I}_{pi,pr,nsc}$ are very close to the problem of *module checking* of universal CTL formulas, which is known to be in PTIME [16]. The proof technique here involves an emptiness check on a tree automaton representing the space of perfect information, perfect recall strategies (either substitution consistent or not required to be so), intersected with an automaton representing the complement of the formula. The cases $\mathcal{I}_{pi,pr,nsc}$ and $\mathcal{I}_{pi,ir,sc}$ can moreover be shown to be equivalent by means of simulation techniques, so the latter also falls into PTIME.

The demarcation between the PTIME and co-NP hard cases is depicted in Figure 2. This shows there are two best candidates for use as the approximation scheme underlying our synthesis approach. We desire an approximation scheme that is as close as possible to the knowledge-based program semantics, while remaining tractable. The diagram shows two orthogonal approximation schemes that are maximal amongst the PTIME cases, namely $\mathcal{I}^{\top}$ and $\mathcal{I}^{pi,pr,sc}$. The former generates a bushy approximation in that it relaxes substitution consistency. The latter remains close to the original protocol by using substitution consistent strategies, but at the cost of allowing perfect information, perfect recall strategies. It is not immediately clear what the impact of these differences will be with respect to the quality of the implementations synthesized using these schemes, and we leave this as a question for future work.

7 Related Work

Relatively little work has been done on automated synthesis of implementations of knowledge-based programs or of sound local proposition specifications, particularly with respect to the observational semantics we have studied in this paper. In addition to the works already cited above, some papers [18, 17, 20, 21, 3] have studied the complexity of synthesis with respect to specifications in temporal epistemic logic using the synchronous perfect recall semantics. A symbolic implementation for knowledge-based programs that run only a finitely bounded number of steps under a clock or perfect recall semantics for knowledge is developed in [13].

There also exists a line of work that is applying knowledge based approaches and model checking techniques to problems in discrete event control, e.g., [2, 10, 15]. In general, the focus of these works is more specific than ours (e.g., in restricting to synthesis for safety properties, rather than our quite general temporal epistemic specifications) but there is a similar use of monotonicity. It would be interesting to apply our techniques in this area and conduct a comparison of the results.

8 Conclusion

In this paper we have proposed an ordered semantics for sound local proposition epistemic protocol specifications, and analyzed the complexity of a model checking problem required to implement the approach, for a number of approximation schemes. This leads to the identification of two optimal approximation schemes, $\mathcal{I}^{\top}$ and $\mathcal{I}^{pi,pr,sc}$ with respect to which the model checking problem has PTIME complexity in an explicit state representation.

A number of further steps are required to obtain a practical framework for synthesis. Ultimately, we would like to be able to implement synthesis using symbolic techniques, so that it can also be practicably carried out for specifications in which the environment is given implicitly using program-like representations, rather than by means of an explicit enumeration of states. The complexity analysis in the present paper develops an initial understanding of the nature of the model checking problems that may be helpful in developing symbolic implementations. In the case of the approximation scheme $\mathcal{I}^{\top}$, in fact, the associated model checking problem amounts essentially to CTLK model checking in a transformed model, for which symbolic model checking techniques are well understood. In work in progress, we have developed an implementation of this case, and we will report on our experimental findings elsewhere.

In the case of the approximation $\mathcal{I}^{pi,pr,sc}$, the model checking problem is more akin to module checking, for which symbolic techniques are less well studied. This case represents an interesting question for future research, as does the question of how the implementations obtained in practice from these tractable approximations differ.

Our examples in this paper give some initial data points that suggest both that the ordered approach is able to construct natural implementations for the sound local proposition weakenings of knowledge-based programs that lack implementations, as well as implementations of such weakenings that are in fact implementations of the original knowledge-based program. More case studies are required to understand how general these phenomena are in practice. It would be interesting to find sufficient conditions under which the ordered approach is guaranteed to generate knowledge-based program implementations.

References

[1] K. Baukus & R. van der Meyden (2004): *A knowledge based analysis of cache coherence.* In: *Proc. 6th Int. Conf. on Formal Engineering Methods*, pp. 99–114, doi:10.1007/978-3-540-30482-1_15.

[2] S. Bensalem, D. Peled & J. Sifakis (2010): *Knowledge Based Scheduling of Distributed Systems*. In: *Time for Verification, Essays in Memory of Amir Pnueli*, Springer LNCS 6200, pp. 26–41, doi:10.1007/978-3-642-13754-9_2.

[3] R. Bozianu, C. Dima & E. Filiot (2014): *Safraless Synthesis for Epistemic Temporal Specifications*. In: *Proc. Int. Conf. on Computer Aided Verification*, pp. 441–456, doi:10.1007/978-3-319-08867-9_29.

[4] R. I. Brafman, J-C. Latombe, Y. Moses & Y. Shoham (1997): *Applications of a logic of knowledge to motion planning under uncertainty*. *JACM* 44(5), doi:10.1145/265910.265912.

[5] J. Burgess (1979): *Logic and time*. *Journal of Symbolic Logic* 44, pp. 556–582, doi:10.2307/2273296.

[6] C. Dwork & Y. Moses (1990): *Knowledge and common knowledge in a Byzantine environment: crash failures*. *Information and Computation* 88(2), pp. 156–186, doi:10.1016/0890-5401(90)90014-9.

[7] K. Engelhardt, R. van der Meyden & Y. Moses (1998): *Knowledge and the Logic of Local Propositions*. In: *Proc. Conf. Theoretical Aspects of Knowledge and Rationality*, pp. 29–41.

[8] R. Fagin, J. Halpern, Y. Moses & M. Vardi (1995): *Reasoning About Knowledge*. MIT Press.

[9] R. Fagin, J. Y. Halpern, Y. Moses & M. Y. Vardi (1997): *Knowledge-Based Programs*. *Distributed Computing* 10(4), pp. 199–225, doi:10.1007/s004460050038.

[10] Susanne Graf, Doron Peled & Sophie Quinton (2012): *Achieving distributed control through model checking*. *Formal Methods in System Design* 40(2), pp. 263–281, doi:10.1007/s10703-011-0138-9.

[11] V. Hadzilacos (1987): *A knowledge-theoretic analysis of atomic commitment protocols*. In: *PODS '87: Proc. 6th ACM Symp. on Principles of Database Systems*, pp. 129–134, doi:10.1145/28659.28672.

[12] J. Y. Halpern & L. D. Zuck (1992): *A little knowledge goes a long way: knowledge-based derivations and correctness proofs for a family of protocols*. *Journal of the ACM* 39(3), pp. 449–478, doi:10.1145/146637.146638.

[13] X. Huang & R. van der Meyden (2013): *Symbolic Synthesis of Knowledge-based Program Implementations with Synchronous Semantics*. In: *Proc. TARK*, pp. 121–130.

[14] X. Huang & R. van der Meyden (2014): *Symbolic Synthesis for Epistemic Specifications with Observational Semantics*. In: *Proc. Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pp. 455–469, doi:10.1007/978-3-642-54862-8_39.

[15] Gal Katz, Doron Peled & Sven Schewe (2011): *Synthesis of Distributed Control through Knowledge Accumulation*. In: *Proc. Int. Conf on Computer Aided Verification*, pp. 510–525, doi:10.1007/978-3-642-22110-1_41.

[16] O. Kupferman, M. Y. Vardi & P. Wolper (2001): *Module Checking*. *Information and Computation* 164(2), pp. 322–344, doi:10.1006/inco.2000.2893.

[17] R. van der Meyden (1996): *Constructing Finite State Implementations of Knowledge-Based Programs with Perfect Recall*. In: *Intelligent Agent Systems, Theoretical and Practical Issues*, LNCS, No. 1209, Springer, pp. 135–151, doi:10.1007/3-540-62686-7_33.

[18] R. van der Meyden (1996): *Finite State Implementations of Knowledge-Based Programs*. In: *Proc. Conf. on Foundations of Software Technology and Theoretical Computer Science*, pp. 262–273, doi:10.1007/3-540-62034-6_55.

[19] R. van der Meyden (1996): *Knowledge Based Programs: On the Complexity of Perfect Recall in Finite Environments*. In: *Proc. Conf. on Theoretical Aspects of Rationality and Knowledge*, pp. 31–49.

[20] R. van der Meyden & M. Y. Vardi (1998): *Synthesis from Knowledge-Based Specifications.* In: *Proc. CONCUR'98*, Springer LNCS 1466, pp. 34–49, doi:10.1007/BFb0055614. Extended version at `http://arxiv.org/abs/1307.6333`.

[21] R. van der Meyden & T. Wilke (2005): *Synthesis of Distributed Systems from Knowledge-Based Specifications*. In: *Proc. Int. Conf. on Concurrency Theory, CONCUR*, pp. 562–576, doi:10.1007/11539452_42.

[22] R. van der Meyden & K. Wong (2003): *Complete Axiomatizations for Reasoning about Knowledge and Branching Time*. *Studia Logica* 75(1), pp. 93–123, doi:10.1023/A:1026181001368.