Abstract. We introduce the notion of a Public Environment: a system in which a publicly known program is executed in an environment that is partially observ-
able to agents in the system. Although agents do not directly have access to all variables in the system, they may come to know the values of unobserved vari-
ables because they know how the program is manipulating these variables. We develop a logic for reasoning about Public Environments, and an axiomatization of the logic.

1 Introduction

From philosophy [7], through game theory [1], to computer science [4, 10], the standard logic for reasoning about knowledge properties of a system with m agents is the multi-
modal system $S^5_m$. In $S^5_m$, each agent is associated with a notion of indistinguishability, which imposes an equivalence relation on the domain. In the paradigm of interpreted systems [4] this semantics is sometimes called grounded [16] in the sense that those equivalence relations can be given a natural computational interpretation: the domain consists of global states, and two such states $s$ and $t$ are indistinguishable for agent $i$ if and only if their local parts are the same, i.e., if $s_i = t_i$. A local state $s_i$ is nothing else than the values of the variables that agent $i$ can ‘see’. Work to connect general $S^5_m$ with that of interpreted systems started in the late 1990s with [9].

In this paper we introduce a computational model for epistemic logic known as Public Environments (PEs), and we then investigate the notions of knowledge that arise in PEs. In PEs, as in interpreted systems, agents have access to a local set of variables. But, crucially, their knowledge can be based on a refinement of the indistinguishability relation that derives from these variables: the agents may observe something has happened, so that the agent is able to rule out some states, even though the local information in such states is the same as in the current state. Moreover, the protocol in a PE is a ‘standard’ PDL program, and it is commonly known that the program under execution is visible to all agents in the PE.

Let us illustrate why it makes sense to distinguish between visibility of variables and indistinguishability based on those values. Assume there is some PDL program $\pi$ that is being executed, and that $\pi$ manipulates the values of the variables in the system. The program $\pi$ is assumed to be public knowledge. Suppose agent $i$ sees the variables $y$ and $z$, and this is commonly known. So, it is public knowledge that $i$ knows the value of $y$ and $z$, at any stage of the program. There is a third variable, $x$, which is not visible to $i$. Now, suppose that the program $\pi$ is actually the assignment $x := y$. After this program is executed, it will be common knowledge that $i$ knows the value of (the new)
\[ x, \text{ even though the value of } x \text{ is not visible to } i, \text{ because } i \text{ sees the value of } y \text{ and knows that the program assigned this value to } x. \text{ If the program } y \equiv x \text{ is executed instead, then } i \text{ will come to learn the (old and current) value of } x. \text{ Were the assignment } y \equiv x + z, \text{ then } i \text{ would learn the value of } x \text{ as well. Thus agents can ‘learn’ the values of variables through certain programs (e.g., } x := 1). \text{ Of course, this information may become subsequently lost again, for example by executing a program } x := u. \]

The kind of questions that we want to settle using PE then typically relate to, for example, how the process of executing a publicly known program in a PE will affect the knowledge of agents who are able to see only some subset of the variables that the program is manipulating. Using PE, it is possible to analyse problems like the Russian Cards [15] and the Dining Cryptographers [3], and indeed many game-like scenarios, by incorporating the protocol (the outcome of the tossing of a coin, the passing of a card from one player to another) explicitly in the object language: we are not restricted to modelling every action by an information update, as in DEL. So, PE’s address the problem of Dynamic Epistemic Logic with factual change [13]. [8] also studies Dynamic Epistemic Logic with assignments, but where our assignments are restricted to program variables, assignments in [8] regard the assignment of the truth of an arbitrary formula to a propositional atom.

2 Motivation

Consider the following well known problem first described in [3].

Three cryptographers are sitting down to dinner at their favourite three-star restaurant. Their waiter informs them that arrangements have been made with the maitre d’hotel for the bill to be paid anonymously. One of the cryptographers might be paying for the dinner, or it might have been NSA. The three cryptographers respect each other’s right to make an anonymous payment, but they wonder if NSA is paying. They resolve their uncertainty fairly by carrying out the following protocol:

Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer on his right, so that only the two of them can see the outcome. Each cryptographer then states aloud whether the two coins he can see the one he flipped and the one his left-hand neighbour flipped-fell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number indicates that NSA is paying (assuming that the dinner was paid for only once). Yet if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is.

What are the essential features of the problem and its solution? We must find a protocol such that: (1) A certain level of knowledge, or equivalently - ignorance, must be maintained among a number of participants in some activity (e. g., the identity of the paying cryptographer, if there is one, must remain unknown for the others throughout the execution of the protocol); (2) The protocol may be such that not all of the relevant
features of the activity are observable or known by all the participants (e. g., the first cryptographer cannot see the value of the coin shared between the second and the third cryptographer); (3) The protocol is common knowledge among the participants.

It is well known that finding a suitable protocol for the accomplishment of a certain task is, in general, an extremely difficult mathematical problem. However, once such a protocol has been found, we must prove that it has the required characteristics. In our example, this means that we must be able to prove that, for instance:

- None of the three cryptographers knows whether one of the other two has lied about the coins he could see at the relevant step of the execution of the protocol;
- The identity of the paying cryptographer, if there is one, must be kept secret, i.e. a certain level of ignorance must be maintained;
- If there is a paying cryptographer, this must be known after the execution of the protocol.

Of course, there are many formal approaches to the verification of protocols like the one described in the example above. Here, we are more interested in the following problem: Suppose that a number of agents are engaged in a commonly known algorithmic activity. What can be said about the evolution of their knowledge if they are not able to make complete observations on their environment? Note that answering such a question is often a part of the process of verification of a cryptographic protocol but it is not the whole story. If we go back to the dining cryptographers, it means that apart from verifying that no one can learn the name of the paying cryptographer at any step of the execution of the protocol, we must prove that the protocol cannot be abused.

So, in order to give some intuition about the question we are concerned with, let us first model the solution proposed by David Chaum in the following way. We name the three cryptographers 1, 2 and 3 respectively. Suppose that instead of coins each different pair of cryptographers \((i,j)\), where \(i,j \in \{1,2,3\}\) and \(i \neq j\), is assigned a different variable \(c_{(i,j)}\) that can take the boolean values 0 and 1. These values model the two sides of the coin shared between \(i\) and \(j\) and the current value of the variable \(c_{(i,j)}\) is visible only to \(i\) and \(j\). How could we model the situation where a cryptographer is paying? Let each cryptographer be assigned a private variable \(p_i\) that is not visible to the rest. These three variables can take only the values 0 and 1. If a certain variable is set to 1, then the respective cryptographer is paying. We will assume that at most one of the variables \(p_1, p_2, p_3\) is set to 1. Next we model the announcement made by each cryptographer in the following way. Let us associate a variable \(a_i\) with each cryptographer \(i\). Each \(a_i\) is visible to all the cryptographers and holds the value \(c_{(i,j)} \oplus c_{(i,k)}\) if \(p_i = 0\) or the value \(1 - c_{(i,j)} \oplus c_{(i,k)}\) otherwise, where \(\oplus\) stands for the “exclusive or” or \(\text{xor}\) of the two values and \(j \neq k\). In this way we model the two types of announcement a cryptographer can make depending on the fact whether he is paying or not. Finally, we introduce a variable \(r\) which holds the result of \(a_1 \oplus a_2 \oplus a_3\), i.e., the number of differences uttered at the table. Then \(r = 1\) if and only if the number of differences is odd. Let us summarise. We have a set of variables \(V = \{p_1, p_2, p_3, c_{(1,2)}, c_{(1,3)}, c_{(2,3)}, a_1, a_2, a_3, r\}\). We associate a subset \(V(i) \subseteq V\) with every cryptographer \(i\), where \(i \in \{1,2,3\}\). Each set \(V(i)\) represents the variables visible to the respective cryptographer. Therefore, we have

- \(V(1) = \{p_1, c_{(1,2)}, c_{(1,3)}, a_1, a_2, a_3, r\}\);
standard machinery of Kripke models. The underlying set of such a model consists of:

- \( V(2) = \{ p_2, c_{(1,2)}, c_{(2,3)}, a_1, a_2, a_3, r \} \);
- \( V(3) = \{ p_3, c_{(1,3)}, c_{(2,3)}, a_1, a_2, a_3, r \} \).

All variables range over the set \( B = \{ 0, 1 \} \). Now we can represent the protocol described in [3] as a program, written in some programming language, which changes the values of the variables in the appropriate way. For example,

1. \( a_1 := 0; a_2 := 0; a_3 := 0; r := 0; \)
2. \( \text{if } p_1 = 0 \text{ then } a_1 := c_{(1,2)} \oplus c_{(1,3)}; \)
3. \( \text{else if } p_1 = 1 \text{ then } a_1 := 1 - \left( c_{(1,2)} \oplus c_{(1,3)} \right); \)
4. \( \text{if } p_2 = 0 \text{ then } a_2 := c_{(1,2)} \oplus c_{(2,3)}; \)
5. \( \text{else if } p_2 = 1 \text{ then } a_2 := 1 - \left( c_{(1,2)} \oplus c_{(2,3)} \right); \)
6. \( \text{if } p_3 = 0 \text{ then } a_3 := c_{(1,3)} \oplus c_{(2,3)}; \)
7. \( \text{else if } p_3 = 1 \text{ then } a_3 := 1 - \left( c_{(1,3)} \oplus c_{(2,3)} \right); \)
8. \( r := a_1 \oplus a_2 \oplus a_3. \)

The first line of the program just sets the value of all the public variables to 0. The meaning of the next six lines of the program is "If the cryptographer \( i \) is not paying, i.e., the variable \( p_i \) is set to 0, then \( i \) truthfully announces whether \( c_{(i,j)} \) and \( c_{(i,k)} \) are different or not, i.e., \( a_i \) is set to \( c_{(i,j)} \oplus c_{(i,k)} \). If \( i \) is paying, i.e., \( p_i \) is set to 1, then \( a_i \) is set to the opposite value of \( c_{(i,j)} \oplus c_{(i,k)} \)." Line 8 sets the value of \( r \) to xor of \( a_1, a_2, a_3 \). Note that we assume that not more than one of the variables \( p_1, p_2, p_3 \) is set to 1. This is common knowledge among the observing agents. We do not assume that the agents have no other knowledge beside the knowledge obtained from the observable variables. This would be unrealistic given the fact that in the original protocol the three cryptographers have a lot of background knowledge which does not depend only on the observations they make during the execution of each step. For example, they must be convinced that an odd number of differences implies that one of them is paying, i.e., they must know some of the properties of the xor function and they implicitly know that not more than one of them is paying, no one is lying etc. In short, in order to model the dynamics of the relevant knowledge (in this case this means the knowledge obtained from observation), we may have to assume a lot of relevant background knowledge. The only requirement we have is:

- The knowledge gained from observation does not contradict the background knowledge of the agents, i.e., in this case no one of the agents can consider it possible (based on the values of the observable variables only) that more than one of the variables \( p_i \) can have the value 1.

Let us address the evolution of the knowledge gained from observation of the three cryptographers during the execution of this program. We assume that before the start of the program all variables different from \( p_1, p_2 \) and \( p_3 \) have some random boolean values. In addition, we assume that at most one of the variables \( p_1, p_2 \) and \( p_3 \) is set to 1. Each step of the program induces some change in the knowledge of each cryptographer by changing the value of the relevant program variables. We can model the knowledge of all the cryptographers at a given point during the execution of the program using the standard machinery of Kripke models. The underlying set of such a model consists of:
– (W) All the assignments of values to the program variables considered possible by at least one of the cryptographers at this particular step of the execution of the program.

The epistemic relation modelling the knowledge of a cryptographer $i$ must be such that

– If $i$ cannot distinguish between two valuations then they must coincide on the values assigned to the variables in $V(i)$;
– If a cryptographer $i$ can distinguish two valuations at a particular step of the computation, then s/he can distinguish their updated images at the next step of the computation.

If $M$ is a model for our description of the protocol, then we would like it to satisfy properties like the following. Let

$$n = (\neg p_1 \land \neg p_2 \land \neg p_3) \quad \text{(i.e., the NSA paid)}$$

Saying that if the NSA paid, all cryptographers will know it afterwards, and if the NSA did not pay but a cryptographer did, this will become known as well. Of course, on top of this we need:

$$M \models (n \rightarrow [\pi] \bigwedge_{i=1,2,3} K_i n) \land (\neg n \rightarrow [\pi] \bigwedge_{i=1,2,3} K_i \neg n)$$

Note that we do not say that if two valuations coincide on the values of the variables visible to $i$, then $i$ cannot differentiate between them. This assumption would lead to counterintuitive scenarios.

Let us summarise. We want to model the following scenario. A group of agents is engaged in an algorithmic activity. These agents can observe only a part of their environment which is affected by this activity. Based on their knowledge of the algorithm and on the values of the observable “variables”, they are able to draw some conclusions and to update their knowledge at each step of the algorithm. We assume that each “variable” can be assigned only finitely many values from some domain. To make things easier, we further assume that the possible values can be just 0 and 1. During its execution, the algorithm can act on only finitely many variables of the agents’ environment. The basic algorithmic steps that can be performed are assignments of the form $x := t$, where $t$ is a term of the language to be defined in the next section, and tests. The agents’ knowledge at each particular step of the algorithm is modelled using Kripke models. The dynamics of the knowledge is modelled using suitably defined updates of the Kripke models.

3 Language and semantics

Let $Ag = \{a_1, \ldots, a_m\}$ be a set of agents and $Var = \{x_1, \ldots, x_n\}$ a set of variables. We define $\varphi \in L$:

$$\varphi := \varphi_0 \mid V_i x \mid \neg \varphi \mid \varphi \land \varphi \mid [\pi] \varphi \mid K_i \varphi \mid O_i \varphi \mid \Box \varphi$$

where $i \in Ag$, $V_i x$ says that agent $i$ sees the value of $x$, $O_i \varphi$ denotes that $i$ observes that $\varphi$ holds, $K_i$ is the knowledge operator, and $\Box$ will be a universal modal operator: it
facilitates us to express properties like \( \Box \tau \) \((K_i(x = 0) \land \neg K_j(x = 0))\): ‘no matter what the actual valuation is, after execution of \( \tau \), agent \( i \) knows that the value of \( x \) is 0, while \( j \) does not know this.

To define a Boolean expression \( \varphi_0 \), we first define terms \( t \). In this paper, terms will have values over \( \{0, 1\} \), this is a rather arbitrary choice, but what matters here is that the domain is finite. Terms are defined as

\[
t := 0 \mid 1 \mid x \mid t + t \mid t \times t \mid -t
\]

\( \text{VAR}(t) \) denotes the set of variables occurring in \( t \). Boolean expressions over terms are:

\[
\varphi_0 := t \mid t < t \mid \neg \varphi_0 \mid \varphi_0 \land \varphi_0
\]

Finally, we define programs:

\[
\tau := \varphi_0? \mid x := t \mid \tau \lor \tau \mid \tau; \tau
\]

where \( x \in \text{VAR} \) and \( \varphi_0 \) is a Boolean expression.

A valuation \( \theta : \text{VAR} \to \{0, 1\} \) assigns a value to each variable. Let the set of valuations be \( \Theta \). We assume that \( v \) is extended to a function \( v : \text{TER} \to \{0, 1\} \) in a standard way, i.e., we assume a standard semantics \( \models_\text{B} \) for which \( v \models_\text{B} \varphi_0 \) is defined in terms of giving a meaning to \( +, -, \times, < \). We extend this semantics \( \models_\text{B} \) to establish the truth of algebraic claims \( \varphi_0 \), in particular that enables us to reason about such claims: \( \{\varphi_1, \varphi_2, \ldots\} \models_\text{B} \varphi_0 \) simply means that any valuation satisfying \( \varphi_1, \ldots, \varphi_0 \) also satisfies \( \varphi_0 \). Each agent \( i \) is able to read the value of some variables \( V(i) \subseteq \text{VAR} \). For two valuations \( \theta_1, \theta_2, \) we write \( \theta_1 \sim_i \theta_2 \) if for all \( x \in V(i) \), \( \theta_1(x) = \theta_2(x) \).

**Definition 1 (Epistemic Models).** Given \( \text{VAR} \) and \( \text{Ag} \), we first define an epistemic \( S5 \)-model for \( \text{Ag} \) with universal modality, or epistemic model, for short, as

\[
M = \langle W, R, V, f \rangle \text{ where}
\]

1. \( W \) is a non-empty set of states;
2. \( f : W \to \Theta \) assigns a valuation \( \theta \) to each state;
3. \( R : \text{Ag} \to 2^{(W \times W)} \) assigns an accessibility relation to each agent: we write \( Ruv \) rather than \( (u, v) \in R(i) \). Each \( R(i) \) is supposed to be an equivalence relation, and moreover, we assume that \( Ruv \) implies that \( f(u) \sim_i f(v) \);
4. \( V : \text{Ag} \to 2^\text{VAR} \) keeps track of the set of variables \( U_i \) that agent \( i \) ‘can see’.

If \( M = \langle W, R, V, f \rangle \), we write \( w \in M \) for \( w \in W \). For \( u, v \in M \), we write \( u \sim_i v \) for \( f(u) \sim_i f(v) \). A pair \( M, v \) (with \( v \in M \)) is called a pointed model. Let \( \mathcal{EM} \) denote the class of pointed epistemic models.

We now define \( R_c : \mathcal{EM} \times \mathcal{EM} \) This is a simple variation of PDL ((6)), for which we use assignments and tests as atomic programs, and sequential composition and choice as composites. In a setting like ours, where we have knowledge, the test will behave as a public announcement (2). First, given a valuation \( \theta \in \Theta \), define \( (x := t)(\theta) \in \Theta \) as follows:

\[
((x := t)(\theta))(y) = \begin{cases} 
\theta(y) & \text{if } y \neq x \\
\theta(t) & \text{else}
\end{cases}
\]
Definition 2 (Assignment). Given \((M, v) = ((W, R, V, f), v)\), we define \(R(v = 0) (M, v)(M', v')\) iff \((M', v') = (W', R', V', f'), v'\), where

1. \(W' = \{w' \mid w \in W\}\)
2. \(R(v' u') = \sim_1 u'\) and \(R(v u)\)
3. \(V' = V\)
4. \(f'(w') = (x := t(f(w)))\); i.e., if \(f(w) = 0\), then \(f'(w') = (x := t(0))\)

Definition 3 (Test). Given a pointed model \(M, v\), we now define the effect of a test \(\varphi_0\). This time, we first specify the result of applying the test to the whole model \(M = (W, R, V, f) : (\varphi_0)(M) = M' = (W', R', V', f')\) where

1. \(W' = \{v' \mid v \in W \& M, v \models \varphi_0\}\)
2. \(R(v' u') = \sim_1 u'\) and \(R(v u)\)
3. \(V' = V\)
4. \(f'(v') = f(v)\)

We then say that \(R_{\varphi_0}(M, v)(M', v')\) if \((\varphi_0)(M, v) = (M', v')\) and \(M, v \models \varphi_0\).

Finally, we define the relations for sequential composition and choice ([6]):

\(R_{\tau_1 \tau_2}[(M, v)(M', v')\) iff \(\exists M''\), \(v''\) such that \(R_{\tau_1}[(M, v)(M'', v'')]\) and \(R_{\tau_2}[(M'', v'')(M', v')\).

\(R_{\tau_1 \lor \tau_2}[(M, v)(M', v')\) iff either \(R_{\tau_1}[(M, v)(M', v')]\) or \(R_{\tau_2}[(M, v)(M', v')]\).

Definition 4 (Public Environment). Given a set of programs \(T\), a Public Environment \(P = (EM, T)\) is a set of pointed epistemic models \(EM \subseteq EM\) closed under programs \(\tau \in T\). Let \(M = (W, R, V, f) \in EM\). A triple \(P, M, v\) is called a state of the public environment, or a state. We define:

\(- P, M, v \models \varphi_0\) iff \(f(v) \models \varphi_0\)
\(- P, M, v \models V_x i f x \in V(i)\)
\(- P, M, v \models \neg \varphi\) iff \(P, M, v \models \varphi\)
\(- P, M, v \models \varphi \land \psi\) iff \(P, M, v \models \varphi\) and \(P, M, v \models \psi\)
\(- P, M, v \models K_i \varphi\) iff for all \(u \in W\): \(R(vu)\) implies \(P, M, u \models \varphi\)
\(- P, M, v \models O_i \varphi\) iff for all \(w \in W\): \(f(w) \sim f(v) \Rightarrow P, M, w \models \varphi\)
\(- P, M, v \models \Box_i \varphi\) iff for all \(w \in W\): \(P, M, w \models \varphi\)
\(- P, M, v \models [\tau]_i \varphi\) iff \((\tau)(M, v)(M', v')\) implies \(P, M', v' \models \varphi\)

We write \(M_{\varphi}\) for the dual of \(K_i \varphi\), i.e., \(M_{\varphi} = \neg K_i \neg \varphi\). Also, \(O_i \varphi\) will denote the dual of \(O_i \varphi\). Note that the operator \(O_i \varphi\) is definable in terms of valuation descriptions and boxes. Let \(x_1, \ldots, x_m\) be the variables that agent \(i\) can observe (which can be expressed as \(\bigwedge_{1 \leq i \leq m} x_i \land \neg \bigvee_{j \leq m} x_j\)). Then \(O_i \varphi =\)

\[
\bigvee_{e_i \in \{0, 1\}, j \leq m} (e_1 e_2 \ldots e_m \land \Box(e_1 e_2 \ldots e_m \Rightarrow \varphi))
\]
Example 1. Figure 1 (left) shows a public environment for two variables $x$ and $y$ where $V_1 = \{x\}$ and $V_2 = \{y\}$. The program that is executed is $(x = 1 \lor y = 1)?; x := 0; y := 1$. Note that the final epistemic model is equivalent to having just a single point $01$: it has become common knowledge that $x = 0 \land y = 1$. Note however that we cannot identify the two states with valuation $01$: in one of them, 1 knows $x = 0 \land y = 1$ but 2 does not, in the other state, it is the other way around.

Example 2. Consider the model $M$ from Figure 1 (right). Assume $V_1 = \{x\}$, and $V_2 = V_3 = \emptyset$. The following table summarises the change in knowledge from $M, 00$ (first row) to $M', 00$ (second row) while executing the program $x := y$.

\[
\begin{array}{c|c|c|c|c}
\neg K_1(y = 0) & \neg K_2(x = 0) & \neg K_3(x = y) & K_3(x = 0) \\
K_1(y = 0) & K_2(x = 0) & K_3(x = y) & \neg K_3(x = 0)
\end{array}
\]

Through the assignment $x := y$ agent 1 learns the value of $y$ (because he reads $x$), agent 2 learns the value of $x$ (because he know the value of $y$) and agent 3, like all other agents, comes to know that $x = y$.

Example 3. We show under which conditions an agent $i$ can learn the value of a variable. Consider the following program, where $x$ and $y$ are different variables:

\[
\alpha = (y = 0?; x := t) \cup (y = 1?; x := u)
\]

The following are valid:
1. \( (K_i(y = 0) \rightarrow [\alpha]K_i(x = t)) \land (K_i(y = 1) \rightarrow [\alpha]K_i(x = u)) \)

   Knowing the condition for branching implies knowing which program is executed;

2. \( (V \land K_i(t \neq u)) \rightarrow ([\alpha]K_i \land [\alpha]K_i = 1) \)

   If an agent can read a variable \( x \), and the value of the variable depends on a variable \( y \), then the agent knows retrospectively what \( y \) was.

3. \( (\neg V \land \neg V \land \neg K_i(y = 0) \land \neg K_i(y = 1)) \rightarrow [\alpha](\neg K_i(y = 0) \land \neg K_i(y = 1)) \)

   An agent who cannot see \( x \) nor \( y \) cannot deduce \( y \)'s value from \( \alpha \).

**Example 4.** We want to swap the value of two variables in a public environment, where the only spare variable is visible to an observer. Can we swap the variables without revealing the variables we want to swap? Formally, let \( \text{Var} = \{x_1, x_2, x_3\}, \text{Ag} = \{1\}, \) and \( V(1) = \{x_2\} \). Informally, the designer of the program wants to ensure that agent 1 never learns the value of \( x_1 \) or \( x_2 \). Formally, if \( i \in \{1, 2\} \), we can capture this in the following scheme:

\[
\chi = \Box [\pi](\neg K_i(x_i = 1) \land \neg K_i(x_i = 0))
\]

Consider the following program \( \pi: x_3 := x_1; x_1 := x_2; x_2 := x_1 \)

Clearly, if \( M \) is the epistemic model that formalises this, we have \( M \models \neg \chi \). But of course, \( \pi \) above is not the only solution to the problem of swapping variables. Now consider the following program \( \pi: x_3 := x_1 + x_2; x_2 := x_1 - x_2; x_1 := x_3 - x_2; \)

In this case, with \( M' \) the epistemic model, we have \( M \models \chi \), as desired.

**Learning and Recall** The principle of recall (wrt. a program \( \tau \)) is

\[
K_i[\tau] \varphi \rightarrow [\tau]K_i \varphi
\]  

(2)

It is straightforward to verify that assignment and test satisfy (2), and moreover, that it is preserved by sequential composition and choice. However, now consider the converse of (2), which is often referred to as no-learning:

\[
[\tau]K_i \varphi \rightarrow K_i[\tau] \varphi
\]  

(3)

This principle is *not* valid, as we can see from Example 2. In that example, we have \( P, M, 00 \models M_1(x := y)(x = 1) \), but not \( P, M, 00 \models (x := y)M_1(x = 1) \) (and hence \( [x := t]\langle \tau \rangle K_i \varphi \rightarrow K_i[x := t] \varphi \) is not valid). Semantically, no learning is

\[
\forall w, t[R \lor \langle w, \tau \rangle \Rightarrow \exists s (R \lor \langle s, t \rangle)]
\]  

(4)

We can phrase failure of (4) in Example 2 as follows: In 00, agent 1 considers the state 01 possible, which with the assignment \( x := y \) would map to 11, however, after the assignment takes the state 00 to 00, since 1 sees variable \( x \), he does not consider the state 11 as an alternative any longer. Loosely formulated: through the assignment \( x := y \) in 00, agent 1 learns that 01 was not the current state.

From the definition of assignment, it is easy to see that the following *is* valid:

\[
(x := t)M_i \varphi \leftrightarrow (x := t)\hat{O}_i \varphi \land M_i(x := t) \varphi
\]  

(5)

Also, the test operator fails no learning: again, in Example 2, we have \( M, 00 \models M_1(y = 1?) \), but we do not have \( P, M, 00 \models (y = 1?)M_1 \uparrow \); by the fact that 00 does not ‘survive’ the test \( y = 1 \), agent 1 ‘learns’ that it was not the real state.
Before revisiting the dining cryptographers, we mention some validities.

- \( \models [x := t] \mathcal{K}_i(x = t) \land [x := t?] \mathcal{K}_i(x = t) \)

Agents see the programs and are aware of their effects. An assignment \( x := t \) can always be executed, which is not true for a test \( x = t? \).

- \( \models \mathcal{K}_i(t = u) \rightarrow [x := t] \mathcal{K}_i(x = u) \)

Knowing the value of a term implies knowing the value of a variable if it gets assigned that term.

**Dining cryptographers revisited** We will now indicate how to model this scenario in such a way that it is possible to reason about lying participants, a situation where more than one of the cryptographers is paying etc.

Assume \( \mathcal{V} = \{ a_1, a_2, a_3, a_4, h_1, h_2, k_1, k_2, k_3, p_1, p_2, p_3 \} \). This will be used as follows. As before, \( p_i \) indicates that \( i \) has paid, and \( h_i \) means that \( i \) is honest. The variables \( k_1, k_2 \) and \( k_3 \) hold the secret bits shared by 1 and 3, 1 and 2, and 2 and 3, respectively and therefore are visible to the relevant pair of agents. Variables \( a_1, a_2, \) and \( a_3 \) are used to model the announcements of the respective agents and are publicly visible; \( a_4 \) (visible to everybody) stores the xor of \( a_1, a_2 \) and \( a_3 \). Let the set of publicly visible variables be \( P = \{ a_1, a_2, a_3, a_4 \} \). Let \( \mathcal{V} = \mathcal{P} \cup \{ h_1, p_1, k_1, k_2 \} \), \( \mathcal{V} = \mathcal{P} \cup \{ h_2, p_2, k_2, k_3 \} \) and \( \mathcal{V} = \mathcal{P} \cup \{ h_3, p_3, k_3 \} \) be the sets of variables observed by the respective agent.

If we start with some initial boolean values of all the variables then the program \( \pi \) (where \( \oplus \) stands for the “exclusive or”) is as follows.

1. if \( h_1 = 1 \) then \( p_1 := p_1 \)
2. if \( h_2 = 1 \) then \( p_2 := p_2 \)
3. if \( h_3 = 1 \) then \( p_3 := p_3 \)
4. if \( p_1 = 0 \) then \( a_3 := k_3 \oplus k_2 \)
5. if \( p_2 = 0 \) then \( a_2 := k_2 \oplus k_3 \)
6. if \( p_3 = 0 \) then \( a_3 := k_3 \oplus k_1 \)
7. \( a_4 := a_1 \oplus a_2 \oplus a_3 \)

Notice the way we have modelled lying in the first three lines of the program. If \( h_1 = 0 \) then the agent \( A_i \) will behave contrary to what his paying variable indicates, i.e. if \( p_i \) was zero then \( A_i \) behaves as if \( p_i \) was 1 and vice versa. In the original treatment of the problem, it is implicitly assumed that all the cryptographers are honest and at most one of them is paying. This (extremely) strong assumption can be made explicit in our framework.

Let \( \varphi \) stand for the formula:

\[
(h_1 \land h_2 \land h_3) \land \{ (p_1 \land \neg p_2 \land \neg p_3) \lor (\neg p_1 \land p_2 \land \neg p_3) \lor (\neg p_1 \land \neg p_2 \land p_3) \lor (\neg p_1 \land \neg p_2 \land \neg p_3) \}
\]

If we stipulate that our starting epistemic models satisfy \( \varphi \) then the original assumptions of [3] will become common knowledge among the agents.

And for the epistemic model \( M \) associated with our assumptions, we have the properties we wanted to confirm in Example 1, namely:

\[
M \models \left( a_4 \rightarrow [\pi] \bigwedge_{i=1,2,3} \mathcal{K}_i \neg \pi \right) \land \left( \neg a_4 \rightarrow [\pi] \bigwedge_{i=1,2,3} \mathcal{K}_i \pi \right)
\]
Intuitively, the truth of the second conjunct is easily seen. If no one is lying and at most one cryptographer is paying then we can only have an even number (0 or 2) of different couples \((k_i, k_j)\) among \(k_1, k_2, k_3\), so if \(a_4\) (which is a publicly visible variable) is 0 then it is common knowledge that no one of the agents is paying. For the first conjunct, consider the case where we have an odd number (1 or 3) of differences reported (one of the cryptographers has paid). Let us see how a non-paying cryptographer can reason in this case. If the bits he sees are different, namely 0 and 1, and if both other cryptographers reported differences then one of them has lied but, since the hidden bit is unknown to the non-paying agent, he cannot make any conclusion about the identity of the paying cryptographer. Similar reasoning applies in the case when both other agents report that the bits they see are the same. If the bits seen by the non-paying agent are the same then analysis in the same line shows that the identity of the paying agent cannot be discovered, i.e., \(M \models p_i \rightarrow \bigwedge_{j \neq i} (\neg K_j p_j)\)

If we chose, however, to concentrate on the general case where the motivation and honesty of the participants is unknown then many of the shortcomings of this protocol can be easily seen. If two participants behave as if they have paid for the dinner then we will have a collision and the final result displayed in \(a_4\) will be 0 making this case indistinguishable from the case where no one is paying. Notice the words we have chosen, namely 'behave as if they have paid'. This behaviour can result from the fact that they may be lying, or only one of them is lying or both have paid. As long as the variable \(h_i\), is visible only to \(A_i\), these cases cannot be distinguished by the other two agents. Similar observations can be made in the case where only one or all the cryptographers behave as if they have paid.

4 Axiomatisation

Let \(L_0\) be the language without modal operators. The usual ([6]) dynamic axioms apply, although not to the full language: exceptions are the axioms \(\text{Assign}\), and \(\text{test}(\tau)\).

Axiom \(\text{Assign}\) is not valid for arbitrary \(\varphi\), as the following example shows. First of all, we need to exclude formulas of the form \(V^i x\). It is obvious that \([x := y] V^i x\) does not imply \(V^j y\). But even if we made an exception for \(V^i\) formulas, we would leave undesired equivalences: Take \(y\) for \(t\). If \([x := y] K^i y (y = 0)\) holds and \(\text{Assign}\) were valid, this would imply that \(K^j y = 0\), which it should not (it might well be that agent 1 learned the value of \(y\) since he sees that \(x\) becomes 0).

Note that \(\text{VK}\) is only valid for values 0 and 1, and not for arbitrary terms: we do not have \([x := y] (V^i x \land x = t) \rightarrow K^i (x = t)\) (take for instance \(t = z\) for a counterexample).

Our completeness proof uses Theorem 1, of which the proof immediately follows from the following equivalences:

\[
\begin{align*}
[x := t] V^i x & \leftrightarrow V^i x \quad \quad [\varphi_0 ^t] V^i x & \leftrightarrow (\varphi_0 \rightarrow V^i x) \\
[x := t] \varphi & \leftrightarrow \varphi_0 ^{t/x} \quad \quad [\varphi_0 ^t] \psi_0 & \leftrightarrow (\varphi_0 \rightarrow \psi_0) \\
[\alpha] (\varphi \land \psi) & \leftrightarrow ([\alpha] \varphi \land [\alpha] \psi) \quad \quad [x := t] \neg \varphi & \leftrightarrow \neg [x := t] \varphi \\
[\varphi_0 ^t \neg \varphi & \leftrightarrow (\varphi_0 \rightarrow \neg [\varphi_0 ^t] \varphi) \quad \quad [x := t] \varphi & \leftrightarrow \neg [x := t] \varphi \\
[\varphi_0 ^t] \Box \varphi & \leftrightarrow (\varphi_0 \rightarrow \Box [\varphi_0 ^t] \varphi) \quad \quad [x := t] K_v \varphi & \leftrightarrow (K_v [x := t] \varphi \lor [x := t] O_v \varphi) \\
[\varphi_0 ^t] K_v \varphi & \leftrightarrow (\varphi \rightarrow K_v [\varphi_0 ^t] \varphi)
\end{align*}
\]
Theorem 1. Every formula is equivalent to one without dynamic operators.

Theorem 1 implies that every effect of a program is completely determined by the ‘first’ epistemic model. More precisely, it implies that for every formula \( \phi \) there is an equivalent epistemic formula (using \( V_i \) and \( K \)) which is provably equivalent.

Theorem 2. The logic KPPE is sound and complete with respect to public environments.

5 Conclusion

We have introduced a framework for reasoning about programs and valuations where knowledge is based on a hybrid semantics using notions from interpreted systems and general S5 axioms. This is only a first step in doing so. Possible extensions are many-fold. First, we think it is possible to include repetition (\( \? \)) as an operator on programs and still obtain a well-behaved logical system, although the technical details for doing so can become involved. There are several restrictions in our framework that may be worthwhile relaxing, like allowing a richer language for tests, and not assuming that it is common knowledge which variables are seen by whom, or what the program under execution is. Those assumptions seem related, and removing them may well be a way to reason about Knowledge-based Programs ([5]), where the programs are distributed over the agents, and where it would be possible to branch in a program depending on the knowledge of certain agents.
References