Program Semantics and Verification Technique for AI-centred Programs

Fortunat Rajaona[1], Ioana Boureanu[1], Vadim Malvone[2], and Francesco Belardinelli[3]

[1] Surrey Centre for Cyber Security, University of Surrey, United Kindgom
`{s.rajaona,i.boureanu}@surrey.ac.uk`
[2] Télécom Paris, France
`vadim.malvone@telecom-paris.fr`
[3] Imperial College, London, United Kingdom
`francesco.belardinelli@imperial.ac.uk`

Abstract. We give a general-purpose programming language in which programs can reason about their own knowledge. To specify what these intelligent programs know, we define a "program epistemic" logic, akin to a dynamic epistemic logic for programs. Our logic properties are complex, including programs introspecting into future state of affairs, i.e., reasoning now about facts that hold only after they and other threads will execute. To model aspects anchored in privacy, our logic is interpreted over partial observability of variables, thus capturing that each thread can "see" only a part of the global space of variables. We verify program-epistemic properties on such AI-centred programs. To this end, we give a sound translation of the validity of our program-epistemic logic into first-order validity, using a new weakest-precondition semantics and a book-keeping of variable assignment. We implement our translation and fully automate our verification method for well-established examples using SMT solvers.

1 Introduction & Preliminaries

In a digital world governed by strict rules on privacy and access-control [22], some thread A and some thread B will execute concurrently over the same variable space, but A and B will have different, restricted access to global variables. Moreover, both A and B may be decision-making process which take actions based on predictions of future states of their environment [22]. In other words, thread A may need to know now what the state-of-affairs will be after some procedure P runs, albeit as far as A can know modulo its partial observability of the system's variables. More formally, in our framework, we are interested in formulas such as "$K_A \Box_P \varphi$", meaning to reason if "at this current point, thread A knows whether after a procedure P executed, a fact φ expressed over the global domain of variables holds". Or, we may wish to check if agent B knows that agent A knows a fact of this kind, i.e., "$K_B K_A \Box_P \varphi$". Such statements are clearly rich, as they allow threads to reason about the future and moreover about their "perception" of the future, and of one another's perceptions. That is, thread B can check what it "thinks" A will "think" of the global state of the system, after some procedure executes.

A New Verification Method Towards Safer AI. On the one hand, evaluating such knowledge-centric properties considered in a partial observability setting is of paramount importance for AI-based decision making [3]. On the other hand, logics of knowledge, also called *epistemic logics* [15], have been well-explored in computer-science since Hintikka [21], and even in the context of multi-agent systems [30] and under partial observability [36,19,6]. In this space, our innovation focuses in turn on new methods for automatically verifying epistemic properties, but –unlike most of our predecessors– we concentrate on verification methods not for abstract systems, but rather analyses of concrete programs (over an arbitrary first-order domain), as well as requirements richer than what went before us. Notably, we wish to create new formal analyses for the epistemic reasoning of concrete programs, catering for them knowing facts not only after they execute (i.e., $\Box_A K_A \varphi$ or $\Box_B K_A \varphi$), but also before they execute (i.e., $K_A \Box_A \varphi$); the latter allows them as well as a formal-methodist to check local perception of programs on global futures. To this end, we argue that this opens up the area of verification methods for AI-rich programs, their decision-making and thus makes for safer AI.

1.1 Preliminaries & Background

We now introduce a series of logic-related notions that are key to to explaining our contributions in the related field and to setting the scene.

Epistemic Logics. Logics for knowledge or epistemic logics [21] follow a so-called Kripke or "possible-worlds" semantics. Assuming a set of agents, a set of possible worlds are linked by an indistinguishability relation for each agent. Then, an *epistemic formula* $K_a \phi$, stating that "agent a knows that ϕ", holds at a world w, if the statement ϕ is true in all worlds that agent a considers as indistinguishable from world w.

Modelling Imperfect Information in Epistemic Logics. A possible-worlds semantics does not suffice to faithfully capture agents with private information. To this end, *interpreted systems* were introduced [30], whereby agents are associated with private shares of the possible worlds called local states; worlds' indistinguishability is then "sunk" at the level of local states. Alternatively, others looked at how epistemic logic with imperfect information could be expressed via direct notions of visibility (or observability) of propositional variables, e.g., [36,19,6].

Logics of Visibility for Programs. Others [17,29,33] looked at how multi-agent epistemic logics with imperfect information would apply not to generic systems, but specifically to programs. In this setting, the epistemic predicate $K_a(y = 0)$ denotes that agent a knows that the variable y is equal to 0 (in some program). So, such a logic allows for the expression of knowledge properties of program states, using *epistemic predicates.* This is akin to how, in classical program verification, one encodes properties of states using first-order predicates: e.g., Dijkstra's *weakest precondition* [4] [9].

Perfect vs Imperfect recall. For any of the cases aforesaid, an aspect often considered is the amount of knowledge that agents retain, i.e., agents forget all that occur before their current state – *memoryless (or imperfect recall) semantics*, or agents recall all their history of states – *memoryful (or perfect recall) semantics*, or in between the two cases – bounded recall semantics.

"Program-epistemic" Logics. To reason about knowledge change, epistemic logic is usually enriched with *dynamic modalities* from Dynamic Logics [32,20]. Therein, a dynamic formula $\Box_P\phi$ expresses the fact that when the program P's execution terminates, the system reaches a state satisfying ϕ – a statement given in the base logic (propositional/predicate logic); the program P is built from abstract/concrete actions (e.g., assignments), sequential composition, non-deterministic composition, iteration and test.

Gorogiannis *et al.* [17] gave a "program-epistemic" logic, which is a dynamic logic with concrete programs (e.g., programs with assignments on variables over first-order domains such as integer, reals, or strings). Interestingly, à la [36,29,33], the epistemic model in [17] relies on partial observability of the programs' variables by agents. Gorogiannis *et al.* translated program-epistemic validity into a first-order validity, and this outperformed the then state-of-the-art tools in epistemic properties verification. Whilst an interesting breakthrough, Gorogiannis *et al.* present several limitations. Firstly, the verification mechanisation in [17] only supports "classical" programs; this means that [17] cannot support tests on agents' knowledge. Yet, such tests are clearly in AI-centric programs: e.g., in epistemic puzzles [25], in the so-called "knowledge-based" programs in [14], etc. Secondly, the logic in [17] allows only for knowledge reasoning after a program P executed, not before its run (e.g., not $K_{alice}(\Box_P\phi)$, only $\Box_P(K_{alice}\phi)$); this is arguably insufficient for verification of decision-making with "look ahead" into future states-of-affair. Thirdly, the framework in [17] does not allow for reasoning about nested knowledges operators (e.g., $K_{alice}(K_{bob}\phi)$).

1.2 Our Contributions: New & Expressive "Program-epistemic" Logics and Its Generic Verification.

We lift all the limitations of [17] listed above and more. We make the following contributions:

1. We define a *multi-agent, program-epistemic logic* $\mathcal{L}^m_{DK}$, which is a dynamic logic whose base logic is a multi-agent first-order epistemic logic, under an observability-based semantics (Section 2).
   Our logic is *rich*, where the programs modality contains tests on knowledge, and formulas with nested knowledge operators in the multi-agent setting. This is much more expressive than the state-of-the-art.

[4] The weakest precondition $wp(P,\phi)$ is a predicate such that: for any precondition ψ from which the program P terminates and establishes ϕ, ψ implies $wp(P,\phi)$.

2. We give a programming language $\mathcal{PL}$ (programs with tests on knowledge) that concretely defines the dynamic operators in $\mathcal{L}^m_{DK}$. We associate the programming language $\mathcal{PL}$ with a relational semantics and a weakest-precondition semantics, and we show their equivalence (Section 3).
3. We give a sound translation of the validity of a program-epistemic logic into first-order validity (Section 4).
4. We implement the aforesaid translation to allow a fully-automated verification with our program-epistemic logic, via SMT-solving (Section 5).
5. We verify the well-known Dining Cryptographer's protocol [7] and the epistemic puzzle called the "Cheryl's birthday problem" [12]. We report competitive verification results. Collaterally, we are also the first to give SMT-based verification of the "Cheryl's birthday problem" [12] (Section 5).

2 Logical Languages $\mathcal{L}_{FO}$ and $\mathcal{L}^m_{DK}$

We introduce the logics $\mathcal{L}_{FO}$, $\mathcal{L}^m_K$, and $\mathcal{L}^m_{DK}$, used to describe states and epistemic properties of states, and program-epistemic properties of states.

2.1 Syntax of $\mathcal{L}_{FO}$, $\mathcal{L}^m_K$, and $\mathcal{L}^m_{DK}$

Agents and variables. We use $a, b, c, ...$ to denote agents, Ag to denote their whole set, and G for a subset therein.

We consider a set Var of variables such that each variable x in Var is "typed" with the group of agents that can observe it. For instance, we write x_G to make explicit the group $G \subseteq Ag$ of observers of x.

For each agent $a \in Ag$, the set Var of variables can be partitioned into the variables that are observable by a, denoted $\mathbf{o}_a$, and the variables that are not observable by a, denoted $\mathbf{n}_a$. Thus, $\mathbf{n}_a = \{x_G \in Var \mid a \notin G\}$.

The base logic $\mathcal{L}_{QF}$. We consider a user defined base language $\mathcal{L}_{QF}$, on top of which the other logics are built. We assume $\mathcal{L}_{QF}$ to be quantifier-free first-order language with variables in Var. The Greek letter π denotes a formula in $\mathcal{L}_{QF}$. We leave $\mathcal{L}_{QF}$ unspecified for the rest of the paper to allow various instantiations.

First-order logic $\mathcal{L}_{FO}$. We define the quantified first-order logic $\mathcal{L}_{FO}$ based on $\mathcal{L}_{QF}$. This logic describes "physical" properties of a program state and also serves as the target language in the translation of our main logic.

Definition 1. *The quantified first-order logic $\mathcal{L}_{FO}$ is defined by:*

$$\phi ::= \pi \mid \phi \wedge \phi \mid \neg\phi \mid \forall x_G \cdot \phi$$

where π is a quantifier-free formula in $\mathcal{L}_{QF}$, and $x_G \in Var$.

Other connectives and the existential quantifier operator $\exists$, can be derived as standard. We use Greek letters ϕ, ψ, χ to denote first-order formulas in $\mathcal{L}_{FO}$. We extend quantifiers over vectors of variables: $\forall \mathbf{x} \cdot \phi$ means $\forall x_1 \cdot \forall x_2 \cdots \forall x_n \cdot \phi$. As usual, $FV(\phi)$ denotes the set of free variables of ϕ.

Epistemic logic $\mathcal{L}_K^m$ and program-epistemic logic $\mathcal{L}_{DK}^m$. We now define two logics at once. The first is the first-order multi-agent epistemic logic $\mathcal{L}_K^m$ enriched with the public announcement operator. The logic $\mathcal{L}_K^m$ is first-order in the sense that its atomic propositions are predicates from the base language $\mathcal{L}_{QF}$. The second is our main logic, $\mathcal{L}_{DK}^m$, which extends $\mathcal{L}_K^m$ with program modalities $\Box_P$.

Definition 2. *Let $\mathcal{L}_{QF}$ be a base first-order language and $Ag = \{a_1, \ldots, a_m\}$ a set of agents. We define the first-order multi-agent program epistemic logic $\mathcal{L}_{DK}^m$ with the following syntax*

$$\alpha ::= \pi \mid \alpha \wedge \alpha' \mid \neg\alpha \mid K_{a_i}\alpha \mid [\alpha']\alpha \mid \Box_P\alpha \mid \forall x_G \cdot \alpha \qquad (\mathcal{L}_{DK}^m)$$

where $\pi \in \mathcal{L}_{QF}$, P is a program, $G \subseteq Ag$, and $x_G \in Var$.

Each K_{a_i} is the epistemic operator for agent a_i, the epistemic formula $K_{a_i}\alpha$ reads "agent a_i knows that α". The public announcement formula $[\alpha']\alpha$, in the sense of [31,11], means "after every announcement of α', α holds". The dynamic formula $\Box_P\alpha$ reads "at all final states of P, α holds". The program P is taken from a set of programs $\mathcal{PL}$ that we define in Section 3. Other connectives and the existential quantifier $\exists$ can be derived in a standard way as for Definition 1.

The first-order multi-agent epistemic logic $\mathcal{L}_K^m$ is the fragment of $\mathcal{L}_{DK}^m$ without any program operator $\Box_P$.

2.2 Semantics of $\mathcal{L}_{FO}$ and $\mathcal{L}_{DK}^m$

States and the truth of $\mathcal{L}_{QF}$ formulas. We consider a set D, used as the domain for interpreting variables and quantifiers. A *state* s of the system is a valuation of the variables in Var, i.e., a function $s : Var \to \mathsf{D}$. We denote the universe of all possibles states by $\mathcal{U}$.

We assume an interpretation I of constants, functions, and predicates, over D to define the truth of an $\mathcal{L}_{QF}$ formula π at a state s, denoted $s \models_{QF} \pi$.

Truth of an $\mathcal{L}_{FO}$ formula. Let $s[x \mapsto c]$ denote the state s' such that $s'(x) = c$ and $s'(y) = s(y)$ for all $y \in Var$ different from x. This lifts to a set of states, $W[x \mapsto c] = \{s[x \mapsto c] \mid s \in W\}$.

Definition 3. *The truth of $\phi \in \mathcal{L}_{FO}$ at a state s, denoted $s \models_{FO} \phi$, is defined inductively on ϕ by*

$$\begin{array}{lll} s \models_{FO} \pi & \textit{iff} & s \models_{QF} \pi \\ s \models_{FO} \phi_1 \wedge \phi_2 & \textit{iff} & s \models_{FO} \phi_1 \textit{ and } s \models_{FO} \phi_2 \\ s \models_{FO} \neg\phi & \textit{iff} & s \not\models_{FO} \phi \\ s \models_{FO} \forall x_G \cdot \phi & \textit{iff} & \textit{for all } c \in \mathsf{D}, s[x_G \mapsto c] \models_{FO} \phi. \end{array}$$

We lift the definition of $\models_{FO}$ to a set W of states, with $W \models_{FO} \phi$ iff for all $s \in W$, $s \models_{FO} \phi$. The satisfaction set $[\![\phi]\!]$ of a formula $\phi \in \mathcal{L}_{FO}$ is defined, as usual, by $[\![\phi]\!] = \{s \in \mathcal{U} \mid s \models_{FO} \phi\}$.

Epistemic models. We model agents' knowledge of the program state with a possible worlds semantics built on the observability of program variables [17]. We define, for each a in Ag, the binary relation $\approx_a$ on $\mathcal{U}$ by: $s \approx_a s'$ if and only if s and s' agree on the part of their domains that is observable by a, i.e.,

$$s \approx_a s' \text{ iff } \mathsf{dom}(s) \cap \mathbf{o}_a = \mathsf{dom}(s') \cap \mathbf{o}_a \quad \text{and} \quad \textstyle\bigwedge_{x \in (\mathsf{dom}(s) \cap \mathbf{o}_a)} (s(x) = s'(x)).$$

One can show that $\approx_a$ is an equivalence relation on $\mathcal{U}$. Each subset W of $\mathcal{U}$ defines a possible worlds model $(W, \{\approx_{a|W}\}_{a \in Ag})$, such that the states of W are the possible worlds and for each $a \in Ag$ the indistinguishability relation is the restriction of $\approx_a$ on W. We shall use the set $W \subseteq \mathcal{U}$ to refer to an epistemic model, omitting the family of equivalence relations $\{\approx_{a|_W}\}_{a \in Ag}$.

Truth of an $\mathcal{L}_{DK}^m$ formula. We give the semantics of an $\mathcal{L}_{DK}^m$ formula at a pointed model (W, s), which consist of an epistemic model W and a state $s \in W$.

Definition 4. *Let W be an epistemic model, $s \in W$ a state, α a formula in $\mathcal{L}_{DK}^m$ such that $FV(\alpha) \subseteq \mathsf{dom}(W)$. The truth of an epistemic formula α at the pointed model (W, s) is defined recursively on the structure of α as follows:*

$$\begin{array}{ll}
(W,s) \models \pi & \textit{iff } s \models_{QF} \pi \\
(W,s) \models \neg\alpha & \textit{iff } (W,s) \not\models \alpha \\
(W,s) \models \alpha \wedge \alpha' & \textit{iff } (W,s) \models \alpha \textit{ and } (W,s) \models \alpha' \\
(W,s) \models K_a\alpha & \textit{iff for all } s' \in W, s' \approx_a s \textit{ implies } (W,s') \models \alpha \\
(W,s) \models [\beta]\alpha & \textit{iff } (W,s) \models \beta \textit{ implies } (W_{|\beta}, s) \models \alpha \\
(W,s) \models \Box_P\alpha & \textit{iff for all } s' \in R_W(P,s),\ (R_W^*(P,W), s') \models \alpha \\
(W,s) \models \forall x_G \cdot \alpha & \textit{iff for all } c \in \mathsf{D}, (\bigcup_{d \in \mathsf{D}}\{s'[x_G \mapsto d] \mid s' \in W\}, s[x_G \mapsto c]) \models \alpha
\end{array}$$

where $x_G \notin \mathsf{dom}(W)$, $W_{|\beta}$ is the submodel of W that consists of the states in which β is true, i.e., $W_{|\beta} = \{s \in \mathcal{W} \mid (W,s) \models \beta\}$ [4].

This definition extends from a pointed model (W, s) to the entire epistemic model W as follows: $W \models \alpha$ iff for any s in W, $(W, s) \models \alpha$.

Our interpretation of logical connectors, epistemic formulas, and the public announcement formulas are all standard [4,11].

For universal quantification, the epistemic context W is augmented by allowing x_G to be any possible value in the domain. When interpreting $\forall x_G \cdot K_a\alpha'$ where $a \in G$, we have $s \approx_a s'$ iff $s[x_G \mapsto c] \approx_a s'[x_G \mapsto c]$. However, if $a \notin G$, then $s[x_G \mapsto c] \approx_a s'[x_G \mapsto d]$ for any $d \in \mathsf{D}$ and for any $s' \approx_a s$.

In our interpretation of $\Box_P\alpha$, the context W is also updated by the relation R_W, by taking the post-image of W by R_W[5]. The truth of α is interpreted at a post-state s' under the new context. We use the function $R_W(P, \cdot) : \mathcal{U} \to \mathcal{P}(\mathcal{U})$ to model the program P. We give the function $R_W(P, \cdot)$ concretely for each command P, after we define the programming language $\mathcal{PL}$ in the next section.

[5] The post-image of a function f is denoted by f^*, i.e., $f^*(E) = \bigcup\{f(x) | x \in E\}$.

Remark 1. The index W in $R_W(P,\cdot)$ is a set of states in $\mathcal{U}$. Similarly to the classical relational semantics, $R_W(P,s)$ gives the set of states resulting from executing P at a state s. However, we need the index W to represent the epistemic context in which P is executed. Before executing P, an agent may not know that the actual initial state is s, it only knows about the initial state only as far as it can see from its observable variables. The context W contains any state that some agent may consider as the possible initial state.

3 Programming Language $\mathcal{PL}$

Now, we formalise the language for programs inside a program-operator $\Box_P$ of the logic that we introduced in the previous section.

3.1 Syntax of $\mathcal{PL}$

We use the notations from the previous section: $a, b, c, ...$ to denote agents, Ag to denote their whole set, G for a subset therein, etc. We assume that a non-empty subset $PVar$ of Var consists of program variables.

Definition 5. *The programming language $\mathcal{PL}$ is defined in BNF as follows:*

$$P ::= \varphi? \mid x_G := e \mid \textbf{new } k_G \cdot P \mid P;Q \mid P \sqcup Q$$

where $x_G \in Var$, e is a term over $\mathcal{L}_{QF}$, $\varphi \in \mathcal{L}_K^m$, and any variable in P that is not bound by **new** *is in $PVar$.*

The test $\varphi?$ is an assumption-like test, i.e. it blocks the program when φ is refuted and let the program continue when φ holds; $x_G := e$ is a variable assignment as usual. The command **new** $k_G \cdot P$ declares a new variable k_G observable by agents in G before executing P. The operator $P;Q$ is the sequential composition of P and Q. Lastly, $P \sqcup Q$ is the nondeterministic choice between P and Q.

Commands such as **skip** and conditional tests can be defined with $\mathcal{PL}$, e.g., **if** φ **then** P **else** $Q \overset{\text{def}}{=} (\varphi?;\ P) \sqcup (\neg\varphi?;\ Q)$.

3.2 Relational semantics of $\mathcal{PL}$.

Now, we give the semantics of programs in $\mathcal{PL}$. We refer to as classical program semantics, the modelling of a program as an input-output functionality, without managing what agents can learn during an execution. In classical program semantics, a program P is associated with a relation $R_P = \mathcal{U} \times \mathcal{U}$, or equivalently a function $R(P,\cdot) : \mathcal{U} \to \mathcal{P}(\mathcal{U})$, such that $R(P,\cdot)$ maps an initial state s to a set of possible final states.

As per Remark 1, we define the relational semantics of an epistemic program $P \in \mathcal{PL}$ at a state s for a given context W, with $s \in W$. The context $W \subseteq \mathcal{U}$ contains states that some agents may consider as a possible alternative to s.

Definition 6 (Relational semantics of $\mathcal{PL}$ on states). *Let W be a set of states. The relational semantics of a program P given the context W, is a function $R_W(P,\cdot):\mathcal{U}\to\mathcal{P}(\mathcal{U})$ defined inductively on the structure of P by*

$$\begin{array}{lcl}
R_W(P\sqcup Q,s) & = & \{s'[c_{Ag}\mapsto l]\mid s'\in R_W(P,s)\} \\
 & & \cup\,\{s'[c_{Ag}\mapsto r]\mid s'\in R_W(Q,s)\} \\
R_W(P;Q,s) & = & \bigcup_{s'\in R_W(P,s)}\{R_{R^*_W(P,W)}(Q,s')\} \\
R_W(x_G:=e,s) & = & \{s[k_G\mapsto s(x_G),x_G\mapsto s(e)]\} \\
R_W(\mathbf{new}\ k_G\cdot P,s) & = & R^*_W(P,\{s[k_G\mapsto d]\mid d\in\mathsf{D}\}) \\
R_W(\beta?,s) & = & \mathit{if}\ (W,s)\models\beta\ \mathit{then}\ \{s\}\ \mathit{else}\ \varnothing
\end{array}$$

where k_G is not in $\mathsf{dom}(s)$, and c_{Ag} is not in the domain of any state s' in $R_W(P,s)\cup R_W(Q,s)$.

We model nondeterministic choice $P\sqcup Q$ as a disjoint union [5], which is achieved by augmenting every updated state with a new variable c_{Ag}, and assigning it a value l (for left) for every state in $R_W(P,s)$, and a value r (for right) for every state in $R_W(Q,s)$. The semantics for sequential composition is standard. The semantics of the assignment $x_G := e$ stores the past value of x_G into a new variable k_G, and updates the value of x_G into expression e. With this semantics, an agent always remembers the past values of a variable that it observes, i.e., it has perfect recall. The semantics of **new** $k_G\cdot P$ adds the new variable k_G to the domain of s, then combines the images by $R_W(P,\cdot)$ of all states $s[k_G\mapsto d]$ for d in D. A test is modelled as an assumption, i.e., a failed test blocks the program.

In the epistemic context, we can also view a program as transforming epistemic models, rather than states. This view is modelled with the following alternative relational semantics for $\mathcal{PL}$.

Definition 7 (Relational semantics of $\mathcal{PL}$ on epistemic models). *The relational semantics on epistemic models of a program P is a function $F(P,\cdot):\mathcal{P}(\mathcal{U})\to\mathcal{P}(\mathcal{U})$ given by*

$$\begin{array}{lcl}
F(P\sqcup Q,W) & = & \{s[c_{Ag}\mapsto l]\mid s\in F(P,W)\} \\
 & \cup & \{s[c_{Ag}\mapsto r]\mid s\in F(Q,W)\} \\
F(P;Q,W) & = & F(Q,F(P,W)) \\
F(x_G:=e,W) & = & \{s[k_G\mapsto s(x_G),x_G\mapsto s(e)]\mid s\in W\} \\
F(\mathbf{new}\ k_G\cdot P,W) & = & F(P,\bigcup_{d\in\mathsf{D}}W[k_G\mapsto d]) \\
F(\beta?,W) & = & \{s\in W\mid (W,s)\models\beta\}
\end{array}$$

such that k_G and c_{Ag} are variables not in $\mathsf{dom}(s)$.

We assume that every additional c_{Ag}, in the semantics of $P\sqcup Q$, is observable by all agents. The value of c_{Ag} allows every agent to distinguish a state resulting from P from a state resulting from Q. The resulting union is a disjoint-union of epistemic models [5].

The two relational semantics (Def. 6 and Def. 7) are equivalent (see Appendix **??**).

However, we use both to simplify the presentation. On one hand, the relation on states given by $R_W(P,\cdot)$ is more standard for defining a dynamic formula $\Box_P\alpha$ (see e.g. [17]). On the other hand, $F(P,\cdot)$ models a program as transforming states of knowledge (epistemic models) rather than only physical states. Moreover, $F(P,\cdot)$ relates directly with our weakest precondition predicate transformer semantics, which we present next.

3.3 Weakest precondition semantics of $\mathcal{PL}$.

We now give another semantics for our programs, by lifting the Dijkstra's classical weakest precondition predicate transformer [9] to epistemic predicates.

Notation. $\alpha[x\backslash t]$ substitutes x by the term t in α.

Definition 8. *We define the weakest precondition of a program P as the epistemic predicate transformer $wp(P,\cdot) : \mathcal{L}_K^m \to \mathcal{L}_K^m$ with*

$$\begin{array}{lcl} wp(P;Q,\alpha) & = & wp(P, wp(Q,\alpha)) \\ wp(P\sqcup Q,\alpha) & = & wp(P,\alpha)\wedge wp(Q,\alpha) \\ wp(\mathbf{new}\ k_G\cdot P,\alpha) & = & \forall k_G\cdot wp(P,\alpha) \\ wp(\beta?,\alpha) & = & [\beta]\alpha \\ wp(x_G := e,\alpha) & = & \forall k_G\cdot [k_G = e](\alpha[x_G\backslash k_G]) \end{array}$$

for $\alpha\in\mathcal{L}_K^m$ such that $FV(\alpha)\subseteq PVar$.

The definitions of *wp* for nondeterministic choice and sequential composition are similar to their classical versions in the literature, and follows the original definitions in [9]. A similar definition of *wp* for a new variable declaration is also found in [28]. However, our *wp* semantics for assignment and for test differs from their classical counterparts. The classical *wp* for assignment (substitution), and the classical *wp* of tests (implication) are inconsistent in the epistemic context when agents have perfect recall [29,33]. Our *wp* semantics for test follows from the observation that an assumption-test for a program executed publicly corresponds to a public announcement. Similarly, our semantics of assignment involves a public announcement of the assignment being made.

3.4 Equivalence between the two program semantics.

Now, we show that our weakest precondition semantics and our relational semantics are equivalent. For that, we need the following lemma.

Lemma 1. *Consider an epistemic model W, variables x_G and k_G such that k_G is not in the domain of any state in W. Let $W_{x_G\backslash k_G}$ be the model that renames x_G into k_G in the states of W, then*

$$W\models\alpha \ \textit{ iff } \ W_{x_G\backslash k_G}\models\alpha[x_G\backslash k_G].$$

The following equivalence shows that our weakest precondition semantics is sound w.r.t. the program relational model.

Proposition 1. *For every program P and every formula $\alpha \in \mathcal{L}^m_{DK}$,*

$$F(P, W) \models \alpha \quad \textit{iff} \quad W \models wp(P, \alpha).$$

A detailed proof can be found in Appendix B. Below, we sketch the proofs for the cases of nondeterministic choice and assignment.

The equivalence for the case of nondeterministic choice follows from the fact that disjoint union preserves the truth of epistemic formulas (Prop 2.3 in [5]). A formula that is true at both $F(P, W)$ and $F(Q, W)$, remains true at $F(P \sqcup Q, W)$. This allows us to have a standard conjunctive weakest precondition epistemic predicate transformer, i.e. $wp(P \sqcup Q, \alpha) = wp(P, \alpha) \wedge wp(Q, \alpha)$.

We now explain the equivalence for assignment, i.e., how the bookkeeping of variables in our relational semantics of Definition 7 equates to $wp(x_G := e, \alpha) = \forall k_G \cdot [k_G = e](\alpha[x_G \backslash k_G])$. Recall that $F(x_G := e, W)$ renames x_G into k_G in W, then makes a new variable x_G that takes the value e. This translates to the equality $F(x_G := e, W) = F(\textbf{new } x_G \cdot (x_G = e_{x_G \backslash k_G})?, W_{x_G \backslash k_G})$. In the rhs of this equality, x_G is re-introduced as a new variable, W is expanded, by a Cartesian product, into $\bigcup_{d \in D} W[x_G \mapsto d]$ (Definition 7), then restricted to satisfy $x_G = e_{x_G \backslash k_G}$. This restriction corresponds to the semantics of making the assumption test (or public announcement) $(x_G = e_{x_G \backslash k_G})?$. Finally, $F(\textbf{new } x_G \cdot (x_G = e_{x_G \backslash k_G})?, W_{x_G \backslash k_G})$ can be directly to the weakest precondition for assignment via Lemma 1.

The equivalence in Prop 1 serves us in proving that the translation of an $\mathcal{L}^m_{DK}$ formula into a first-order formula, which we present next, is sound w.r.t. the program relational models.

4 Translating $\mathcal{L}^m_{DK}$ to $\mathcal{L}_{FO}$

Our model checking approach relies on the truth-preserving translation between $\mathcal{L}^m_{DK}$ formulas and first-order formulas. We use the following translation function.

Definition 9 (Translation of $\mathcal{L}^m_{DK}$ into $\mathcal{L}_{FO}$). *Let $\pi \in \mathcal{L}_{QF}$ and $\alpha \in \mathcal{L}^m_{DK}$, a be an agent. Let $\mathbf{n} = \mathbf{n}_a \cap (FV(\alpha) \cup FV(\phi))$ be the set of free variables in π and α that are non-observable by a, and $\circ$ be an operator in $\{\wedge, \vee\}$. We define the translation $\tau : \mathcal{L}_{FO} \times \mathcal{L}^m_{DK} \rightarrow \mathcal{L}_{FO}$ as follows:*

$$\begin{aligned}
\tau(\phi, \pi) &= \pi & \tau(\phi, K_a\alpha) &= \forall \mathbf{n} \cdot (\phi \rightarrow \tau(\phi, \alpha)) \\
\tau(\phi, \neg\alpha) &= \neg\tau(\phi, \alpha) & \tau(\phi, [\beta]\alpha) &= \tau(\phi, \beta) \rightarrow \tau(\phi \wedge \tau(\phi, \beta), \alpha) \\
\tau(\phi, \alpha_1 \circ \alpha_2) &= \tau(\phi, \alpha_1) \circ \tau(\phi, \alpha_2) & \tau(\phi, \Box_P\alpha) &= \tau(\phi, wp(P, \alpha)) \\
\tau(\phi, \forall x_G \cdot \alpha) &= \forall x_G \cdot \tau(\phi, \alpha)
\end{aligned}$$

We use the above translation to express the equivalence between the satisfaction of a $\mathcal{L}^m_K$-formula and that of its first-order translation.

Proposition 2. *For every ϕ in $\mathcal{L}_{FO}$, s in $[\![\phi]\!]$, α in $\mathcal{L}_K^m$ such that $FV(\phi) \cup FV(\alpha) \subseteq PVar$, we have that*

$$([\![\phi]\!], s) \models \alpha \textit{ iff } s \models_{FO} \tau(\phi, \alpha).$$

Proof. The proof for the base epistemic logic without public announcement $\mathcal{L}_K$ $(\pi, \neg, \wedge, K_a)$ is found in [17].
Case of public announcement $[\beta]\alpha$

$$\begin{array}{ll}
([\![\phi]\!], s) \models [\beta]\alpha & \\
\equiv \text{if } ([\![\phi]\!], s) \models \beta \text{ then } ([\![\phi]\!]_{|\beta}, s) \models \alpha & \text{truth of } [\beta]\alpha \\
\equiv \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } ([\![\phi]\!]_{|\beta}, s) \models \alpha & \text{induction hypothesis on } \beta \\
\equiv \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } (\{s' \in \mathcal{U} | s' \models_{FO} \phi \text{ and } ([\![\phi]\!], s') \models \beta\}, s) \models \alpha & \\
 & \text{by definition of } [\![\cdot]\!] \text{ and definition of } _{|\beta} \\
\equiv \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } (\{s' \in \mathcal{U} | s' \models_{FO} \phi \text{ and } s' \models_{FO} \tau(\phi, \beta)\}, s) \models \alpha & \\
 & \text{induction hypothesis on } \beta \\
\equiv \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } (\{s' \in \mathcal{U} | s' \models_{FO} \phi \wedge \tau(\phi, \beta)\}, s) \models \alpha & \text{truth of } \wedge \\
\equiv \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } ([\![\phi \wedge \tau(\phi, \beta)]\!], s) \models \alpha & \text{def of } [\![\cdot]\!] \\
\equiv \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } s \models_{FO} \tau(\phi \wedge \tau(\phi, \beta), \alpha) & \text{induction hypothesis} \\
\equiv \text{if } s \models_{FO} \tau(\phi, \beta) \to \tau(\phi \wedge \tau(\phi, \beta), \alpha) & \text{truth of } \to. \ \blacksquare
\end{array}$$

Now, we can state our main theorem relating the validity of an $\mathcal{L}_{DK}^m$ formula, and that of its first-order translation.

Theorem 1 (Main result). *Let $\phi \in \mathcal{L}_{FO}$, and $\alpha \in \mathcal{L}_{DK}^m$, such that $FV(\phi) \cup FV(\alpha) \subseteq PVar$, then*

$$[\![\phi]\!] \models \alpha \quad \textit{iff} \quad [\![\phi]\!] \models_{FO} \tau(\phi, \alpha).$$

Proof. The proof is done by induction on α. The case where $\alpha \in \mathcal{L}_K^m$ follows directly from Proposition 2.

We are left to prove the case of the program operator $\Box_P \alpha$. Without loss of generality, we can assume that α is program-operator-free, i.e., $\alpha \in \mathcal{L}_K^m$. Indeed, one can show that $\Box_P(\Box_Q \alpha')$ is equivalent to $\Box_{P;Q}\alpha'$. We have

$$\begin{array}{ll}
[\![\phi]\!] \models \Box_P \alpha & \\
\equiv \text{ iff for all } s \text{ in } [\![\phi]\!], ([\![\phi]\!], s) \models \Box_P \alpha & \text{by definition of} \models \text{for a model} \\
\equiv \text{ iff for all } s \text{ in } [\![\phi]\!], \text{ for all } s' \text{ in } R_{[\![\phi]\!]}(P, s), (F(P, [\![\phi]\!]), s') \models \alpha & \models \text{for } \Box_P \\
\equiv \text{ iff for all } s' \text{ in } R^*_{[\![\phi]\!]}(P, [\![\phi]\!]), (F(P, [\![\phi]\!]), s') \models \alpha & \text{post-image} \\
\equiv \text{ iff for all } s' \text{ in } F(P, [\![\phi]\!]), (F(P, [\![\phi]\!]), s') \models \alpha & F(P, W) = R^*_W(P, W) \\
\equiv F(P, [\![\phi]\!]) \models \alpha & \text{by definition of} \models \text{for a model} \\
\equiv [\![\phi]\!] \models wp(P, \alpha) & \text{by Proposition 1} \\
\equiv [\![\phi]\!] \models_{FO} \tau(wp(P, \alpha)) & \text{since } wp(P, \alpha) \in \mathcal{L}_K^m, \text{ the previous case applies. } \blacksquare
\end{array}$$

5 Implementation

Our automated verification framework supports proving/falsifying a logical consequence $\phi \models \alpha$ for α in $\mathcal{L}_{DK}^m$ and ϕ in $\mathcal{L}_{FO}$. By Theorem 1, the problem becomes the unsatisfiability/satisfiability of first-order formula $\phi \wedge \neg\tau(\phi, \alpha)$, which is eventually fed to an SMT solver.

In some cases, notably our second case study, the Cheryl's Birthday puzzle, computing the translation $\tau(\phi, \alpha)$ by hand is tedious and error-prone. For such cases, we implemented a $\mathcal{L}_{DK}^m$-to-$\mathcal{L}_{FO}$ translator to automate the translation.

5.1 Mechanisation of Our $\mathcal{L}_{DK}^m$-to-FO Translation

Our translator implements Definition 9 of our translation τ. It is implemented in `Haskell`, and it is generic, i.e., works for any given example[6]. The resulting first-order formula is exported as a string parsable by an external SMT solver API (e.g., `Z3py` and `CVC5.pythonic` which we use).

Our `Haskell` translator and the implementation of our case studies are at `https://github.com/sfrajaona/program-epistemic-model-checker`.

5.2 Case Study 1: Dining Cryptographers' Protocol [7].

Problem Description. This system is described by n cryptographers dining round a table. One cryptographer may have paid for the dinner, or their employer may have done so. They execute a protocol to reveal whether one of the cryptographers paid, but without revealing which one. Each pair of cryptographers sitting next to each other have an unbiased coin, which can be observed only by that pair. Each pair tosses its coin. Each cryptographer announces the result of XORing three Booleans: the two coins they see and the fact of them having paid for the dinner. The XOR of all announcements is provably equal to the disjunction of whether any agent paid.

Encoding in $\mathcal{L}_{DK}^m$ & Mechanisation. We consider the domain $\mathbb{B} = \{T, F\}$ and the program variables $PVar = \{x_{Ag}\} \cup \{p_i, c_{\{i,i+1\}} \mid 0 \leq i < n\}$ where x is the XOR of announcements; p_i encodes whether agent i has paid; and, $c_{\{i,i+1\}}$ encodes the coin shared between agents i and $i+1$. The observable variables for agent $i \in Ag$ are $\mathbf{o}_i = \{x_{Ag}, p_i, c_{\{i-1,i\}}, c_{\{i,i+1\}}\}$ [7] , and $\mathbf{n}_i = PVar \setminus \mathbf{o}_i$.

We denote ϕ the constraint that at most one agent has paid, and e the XOR of all announcements, i.e.

$$\phi = \bigwedge_{i=0}^{n-1} \left(p_i \Rightarrow \bigwedge_{j=0,j\neq i}^{n-1} \neg p_j\right) \qquad e = \bigoplus_{i=0}^{n-1} p_i \oplus c_{\{i-1,i\}} \oplus c_{\{i,i+1\}}.$$

The Dining Cryptographers' protocol is modelled by the program $\rho = x_{Ag} := e$.

[6] Inputs are `Haskell` files.

[7] When we write $\{i, i+1\}$ and $\{i-1, i\}$, we mean $\{i, i+1 \bmod n\}$ and $\{i-1 \bmod n, i\}$.

Experiments & Results. We report on checking the validity for:

$$\beta_1 = \Box_\rho \left((\neg p_0) \Rightarrow \left(K_0 \left(\bigwedge_{i=1}^{n-1} \neg p_i \right) \vee \bigwedge_{i=1}^{n-1} \neg K_0 p_i \right) \right) \qquad \beta_3 = \Box_\rho (K_0 p_1)$$

$$\beta_2 = \Box_\rho \left(K_0 \left(x \Leftrightarrow \bigvee_{i=0}^{n-1} p_i \right) \right) \qquad \gamma = K_0 \left(\Box_\rho \left(x \Leftrightarrow \bigvee_{i=0}^{n-1} p_i \right) \right).$$

The formula β_1 states that after the program execution, if cryptographer 0 has not paid then she knows that no cryptographer paid, or (in case a cryptographer paid) she does not know which one. The formula β_2 reads that after the program execution, cryptographer 0 knows that x_{Ag} is true iff one of the cryptographers paid. The formula β_3 reads that after the program execution, cryptographer 0 knows that cryptographer 1 has paid, which is expected to be false. Formula γ states cryptographer 0 knows that, at the end of the program execution, x_{Ag} is true iff one of the cryptographers paid.

Formulas β_1, β_2, and β_3 were checked in [17] as well. Importantly, formula γ cannot be expressed or checked by the framework in [17]. We compare the performance of our translation on this case-study with that of [17]. To fairly compare, we reimplemented faithfully the SP-based translation in the same environment as ours. We tested our translation (denoted τ_{wp}) and the reimplementation of the translation in [17] (denoted τ_{SP}) on the same machine.

Note that the performance we got for τ_{SP} differs from what is reported in [17]. This is especially the case for the most complicated formula β_1. This may be due to the machine specifications, or because we used binary versions of `Z3` and `CVC5`, rather than building them from source, like in [17].

The results of the experiments, using the `Z3` solver, are shown in Table 1. `CVC5` was less performant than `Z3` for this example, as shown (only) for β_2. Generally, the difference in performance between the two translations were small. The *SP*-based translation slightly outperforms our translation for β_2 and β_3, but only for some cases. Our translation outperforms the *SP*-based translation for β_1 in these experiments. Again, we note that the performance of the *SP*-based translation reported here is different from the performance reported in [17]. Experiments that took more than 600 seconds were timed out

	Formula β_1		Formula β_2			Formula β_3		Formula γ	
n	τ_{wp}+Z3	τ_{SP}+Z3	τ_{wp}+CVC5	τ_{wp}+Z3	τ_{SP}+Z3	τ_{wp}+Z3	τ_{SP}+Z3	τ_{wp}+Z3	τ_{SP}+Z3
10	0.05 s	4.86 s	0.01 s	0.01 s	0.01 s	0.01 s	0.01 s	0.01 s	N/A
50	31 s	t.o.	0.41 s	0.05 s	0.06 s	0.03 s	0.02 s	0.03 s	N/A
100	t.o.	t.o.	3.59 s	0.15 s	0.16 s	0.07 s	0.06 s	0.07 s	N/A
200	t.o.	t.o.	41.90 s	1.27 s	0.71 s	0.30 s	0.20 s	0.30 s	N/A

Table 1. Performance of our *wp*-based translation vs. our reimplementation of the [17] *SP*-based translation for the Dining Cryptographers. Formula γ is not supported by the *SP*-based translation in [17].

5.3 Case Study 2: Cheryl's Birthday Puzzle [12].

This case study involves the nesting of knowledge operators K of different agents.

Problem Description. Albert and Bernard just became friends with Cheryl, and they want to know when her birthday is. Cheryl gives them a list of 10 possible dates: May 15, May 16, May 19, June 17, June 18, July 14, July 16, August 14, August 15, August 17. Then, Cheryl whispers in Albert's ear the month and only the month of her birthday. To Bernard, she whispers the day only. "Can you figure it out now?", she asks Albert. The next dialogue follows:
- Albert: I don't know when it is, but I know Bernard doesn't know either.
- Bernard: I didn't know originally, but now I do.
- Albert: Well, now I know too!

When is Cheryl's birthday?

Encoding and Mechanisation. To solve this puzzle, we consider two agents a (Albert) and b (Bernard) and two integer program variables $PVar = \{m_a, d_b\}$. Then, we constrain the initial states to satisfy the conjunction of all possible dates announced by Cheryl, i.e., the formula ϕ below:

$$\phi(m_a, d_b) = (m_a = 5 \wedge d_b = 15) \vee (m_a = 5 \wedge d_b = 16) \vee \cdots$$

The puzzle is modelled via public announcements, with the added assumption that participants tell the truth. However, modelling a satisfiability problem with the public announcement operator $[\beta]\alpha$ would return states where β cannot be truthfully announced. Indeed, if β is false at s, (i.e., $(\phi, s) \models \neg\beta$), then the announcement $[\beta]\alpha$ is true. For that, we use the dual of the public announcement operator denoted $\langle\cdot\rangle$ [8]. We use the translation to first-order formula:

$$\tau(\phi, \langle\beta\rangle\alpha) \;=\; \tau(\phi, \beta) \wedge \tau(\phi \wedge \tau(\phi, \beta), \alpha).$$

In both its definition and our translation to first-order, $\langle\cdot\rangle$ uses a conjunction where $[\cdot]$ uses an implication.

We denote the statement "agent a knows the value of x" by the formula $\mathrm{Kv}_a x$ which is common in the literature. We define it with our logic $\mathcal{L}_{DK}^m$ making use of existential quantification: $\mathrm{Kv}_a x \;=\; \exists v_a \cdot K_a(v_a = x)$.

Now, to model the communication between Albert and Bernard, let α_a be Albert's first announcement, i.e., $\alpha_a = \neg\mathrm{Kv}_a(d_b) \wedge K_a(\neg\mathrm{Kv}_b(m_a))$. Then, the succession of announcements by the two participants corresponds to the formula

$$\alpha = \langle(\neg\mathrm{Kv}_b(m_a) \wedge \langle\alpha_a\rangle\mathrm{Kv}_b(m_a))?\rangle\mathrm{Kv}_a d_b.$$

Cheryl's birthday is the state s that satisfies $(\phi, s) \models \alpha$.

[8] The formula $\langle\beta\rangle\alpha$ reads "after some announcement of β, α is the case", i.e., β can be truthfully announced and its announcement makes α true. Formally, $(W, s) \models \langle\beta\rangle\alpha$ iff $(W, s) \models \beta$ and $(W_{|\beta}, s) \models \alpha$.

Experiments & Results. We computed $\tau(\phi, \alpha)$ in 0.10 seconds. The SMT solvers `Z3` and `CVC5` returned the solution to the puzzle when fed with $\tau(\phi, \alpha)$. `CVC5` solved it, in 0.60 seconds, which is twice better than `Z3` (1.28 seconds).

All the experiments were run on a 6-core 2.6 GHz Intel Core i7 MacBook Pro with 16 GB of RAM running OS X 11.6. For `Haskell`, we used GHC 8.8.4. The SMT solvers were `Z3` version 4.8.17 and `CVC5` version 1.0.0.

6 Related Work

SMT-Based Verification of Epistemic Properties of Programs. We start with the work of Gorogiannis *et al.* [17] which is the closest to ours. We already compared with this in the introduction, for instance explaining therein exactly how our logic is much more expressive than theirs. Now, we cover other points.

Program Models. The program models in [17] follow a classical program semantics (e.g., modelling nondeterministic choice as union, overwriting a variable in reassignment). This has been shown [29,33] to correspond to systems where agents have no memory, and cannot see how nondeterministic choices are resolved. Our program models assume perfect recall, and that agents can see how nondeterministic choices are resolved.

Program Expressiveness. Gorogiannis *et al.* [17] have results of approximations for programs with loops, although there were no use cases of that. Here we focused on a loop-free programming language, but we believe our approach can be extended similarly. The main advantage of our programs is the support for tests on knowledge which allows us to model public communication of knowledge.

Mechanisation & Efficiency. We implemented the translation which include an automated computation of weakest preconditions (and strongest postconditions as well). The implementation in [17] requires the strongest postcondition be computed manually. Like [17], we test for the satisfiability of the resulting first-order formula with `Z3`. The performance is generally similar, although sometimes it depends on the form of the formulas (see Table 1).

Verification of information flow with program algebra. Verifying epistemic properties of programs with program algebra was done in [29,27,33]. Instead of using a dynamic logic, they reason about epistemic properties of programs with an ignorance-preserving refinement. Like here, their notion of knowledge is based on observability of arbitrary domain program variables. The work in [33] also consider a multi-agent logics and nested K operators and their program also allows for knowledge tests. Finally, our model for epistemic programs can be seen as inspired by [33]. That said, all these works have no relation with first-order satisfaction nor translations of validity of program-epistemic logics to that, nor their implementation.

Dynamic Epistemic Logics *Dynamic epistemic logic* (DEL, [31,2,11]) is a family of logics that extend epistemic logic with dynamic operators.

Logics' Expressivity. On the one hand, DEL logics are mostly propositional, and their extensions with assignment only considered propositional assignment (e.g., [10]); contrarily, we support assignment on variables on arbitrary domains. Also, we have a denotational semantics of programs (via weakest preconditions), whereas DEL operates on more abstract semantics. On the other hand, action models in DEL can describe complex private communications that cannot be encoded with our current programming language.

Verification. Current DEL model checkers include `DEMO` [13] and `SMCDEL` [34]. We are not aware of the verification of DEL fragments being reduced to satisfiability problems. In this space, an online report [35] discusses –at some high level– the translation `SMCDEL` knowledge structures into QBF and the use of `YICES`.

A line of research in DEL, the so called *semi-public environments*, also builds agents' indistinguishability relations from the observability of propositional variables [36,6,19]. The work of Grossi [18] explores the interaction between knowledge dynamics and non-deterministic choice/sequential composition. They note that PDLs assumes memory-less agents and totally private nondeterministic choice, whilst DELs' epistemic actions assume agents with perfect recall and publicly made nondeterministic choice. This is the same duality that we observed earlier between the program epistemic logic in [17] and ours.

Other Works. Gorogiannis *et al.* [17] discussed more tenuously related work, such as on general verification of temporal-epistemic properties of systems which are not programs in tools like `MCMAS` [26], `MCK` [16], `VERICS` [24], or one line of epistemic verification of models specifically of `JAVA` programs [1]. [17] also discussed some incomplete method of SMT-based epistemic model checking [8], or even bounded model checking techniques, e.g., [23]. All of those are loosely related to us too, but there is little reason to reiterate.

7 Conclusions

We advanced a multi-agent epistemic logics for programs $\mathcal{L}^m_{DK}$, in which each agent has visibility over some program variables but not others. This logic allows to reason on agents' knowledge of a program after its run, as well as before its execution. Assuming agents' perfect recall, we provided a weakest-precondition epistemic predicate transformer semantics that is sound w.r.t. to its relational counterpart. Leveraging the natural correspondence between the weakest precondition $wp(P, \alpha)$ and the dynamic formula $\Box_P \alpha$, we were able to give a sound reduction of the validity of $\mathcal{L}^m_{DK}$ formulas to first-order satisfaction.

Based on this reduction an $\mathcal{L}^m_{DK}$ formula into a first-order, we implemented a tool that fully mechanise the verification, calling an SMT solver for the final decision procedure. Our method is inspired from [17], but applies to a significantly larger class of program-epistemic formulas in the multi-agent setting.

The multi-agent nature of the logic, the expressiveness of it w.r.t. knowledge evaluation before and after program execution, as well as a complete verification method for this are all novelties in the field. In future work, we will look at a meet-in-the-middle between the memoryless semantics in [17] and the memoryful semantics here, and methods of verifying logics like $\mathcal{L}^m_{DK}$ but with such less "absolutist" semantics.

Acknowledgments. S. Rajaona and I. Boureanu were partly supported by the EPSRC project "AutoPaSS", EP/S024565/1.

References

1. Balliu, M., Dam, M., Guernic, G.L.: ENCoVer: Symbolic exploration for information flow security. In: 25th IEEE Computer Security Foundations Symposium (CSF 2012),. pp. 30–44. IEEE Computer Society (2012). https://doi.org/10.1109/CSF.2012.24
2. Baltag, A., Moss, L.S., Solecki, S.: The logic of public announcements, common knowledge, and private suspicions. In: Proceedings of the 7th Conference on Theoretical Aspects of Rationality and Knowledge (TARK 98). p. 43–56. Morgan Kaufmann Publishers Inc. (1998)
3. Barfuss, W., Mann, R.P.: Modeling the effects of environmental and perceptual uncertainty using deterministic reinforcement learning dynamics with partial observability. Physical Review E **105**(3), 034409 (2022)
4. Blackburn, P., van Benthem, J.F., Wolter, F.: Handbook of modal logic. Elsevier (2006)
5. Blackburn, P., de Rijke, M., Venema, Y.: Modal logic. Cambridge University Press, New York (2001)
6. Charrier, T., Herzig, A., Lorini, E., Maffre, F., Schwarzentruber, F.: Building epistemic logic from observations and public announcements. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016. pp. 268–277. AAAI Press (2016)
7. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology **1**(1), 65–75 (1988)
8. Cimatti, A., Gario, M., Tonetta, S.: A lazy approach to temporal epistemic logic model checking. In: Proc. of AAMAS-38. pp. 1218–1226. IFAAMAS (2016)
9. Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)
10. van Ditmarsch, H.P., van der Hoek, W., Kooi, B.P.: Dynamic epistemic logic with assignment. In: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems. p. 141–148. AAMAS '05, Association for Computing Machinery (2005)
11. van Ditmarsch, H.P., Hoek, W.v.d., Kooi, B.: Dynamic Epistemic Logic. Synthese Library, Springer (2007)
12. van Ditmarsch, H.P., Hartley, M.I., Kooi, B., Welton, J., Yeo, J.B.: Cheryl's birthday. arXiv preprint arXiv:1708.02654 (2017)
13. van Eijck, J.: A demo of epistemic modelling. Interactive Logic p. 303 (2007)
14. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Knowledge-Based Programs. In: Symposium on Principles of Distributed Computing. pp. 153–163 (1995)
15. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. MIT Press (1995)
16. Gammie, P., van der Meyden, R.: MCK: model checking the logic of knowledge. In: Computer Aided Verification. Lecture Notes in Computer Science, vol. 3114, pp. 479–483. Springer (2004). https://doi.org/10.1007/978-3-540-27813-9_41
17. Gorogiannis, N., Raimondi, F., Boureanu, I.: A Novel Symbolic Approach to Verifying Epistemic Properties of Programs. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17. pp. 206–212 (2017). https://doi.org/10.24963/ijcai.2017/30

18. Grossi, D., Herzig, A., van der Hoek, W., Moyzes, C.: Non-determinism and the dynamics of knowledge. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (2017)
19. Grossi, D., van der Hoek, W., Moyzes, C., Wooldridge, M.: Program models and semi-public environments. Journal of Logic and Computation **29**(7), 1071–1097 (01 2016). `https://doi.org/10.1093/logcom/exv086`
20. Harel, D.: Dynamic Logic, pp. 497–604. Springer Netherlands, Dordrecht (1984). `https://doi.org/10.1007/978-94-009-6259-0_10`
21. Hintikka, J.: Knowledge and Belief. Cornell University Press (1962)
22. Jena, M.D., Singhar, S.S., Mohanta, B.K., Ramasubbareddy, S.: Ensuring data privacy using machine learning for responsible data science. In: Intelligent Data Engineering and Analytics, pp. 507–514. Springer (2021)
23. Kacprzak, M., Lomuscio, A., Niewiadomski, A., Penczek, W., Raimondi, F., Szreter, M.: Comparing BDD and SAT based techniques for model checking Chaum's dining cryptographers protocol. Fundamenta Informaticae **72**(1-3), 215–234 (2006)
24. Kacprzak, M., Nabiałek, W., Niewiadomski, A., Penczek, W., Półrola, A., Szreter, M., Woźna, B., Zbrzezny, A.: VerICS 2007 – a model checker for knowledge and real-time. Fundamenta Informaticae **85**(1-4), 313–328 (2008)
25. Lehman, D.: Knowledge, common knowledge, and related puzzles. In: Proc. of the 3rd ACM Symposium on Principles of Distributed Computing. pp. 62–67 (1984)
26. Lomuscio, A., Qu, H., Raimondi, F.: MCMAS: an open-source model checker for the verification of multi-agent systems. Int. Journal on Software Tools for Technology Transfer **19**(1), 9–30 (2015). `https://doi.org/10.1007/s10009-015-0378-x`
27. McIver, A.K.: The secret art of computer programming. In: Theoretical Aspects of Computing. Lecture Notes in Computer Science, vol. 5684, pp. 61–78. Springer (2009)
28. Morgan, C.: Programming from Specifications. Prentice Hall International Series in Computer Science, Prentice Hall, 2 edn. (1994)
29. Morgan, C.: The Shadow Knows: Refinement of ignorance in sequential programs. In: Mathematics of Program Construction, Lecture Notes in Computer Science, vol. 4014, pp. 359–378. Springer (2006)
30. Parikh, R., Ramanujam, R.: Distributed processing and the logic of knowledge. Lecture Notes in Computer Science **193**, 256–268 (1985)
31. Plaza, J.A.: Logics of public communications. Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems (1989)
32. Pratt, V.R.: Semantical considerations on Floyd-Hoare logic. In: 17th Annual Symposium on Foundations of Computer Science. pp. 109–121. IEEE (1976)
33. Rajaona, S.F.: An algebraic framework for reasoning about privacy. Ph.D. thesis, Stellenbosch: University of Stellenbosch (2016)
34. Van Benthem, J., Van Eijck, J., Gattinger, M., Su, K.: Symbolic model checking for dynamic epistemic logic. In: International Workshop on Logic, Rationality and Interaction. pp. 366–378. Springer (2015)
35. Wang, S.: Dynamic epistemic model checking with Yices. `https://github.com/airobert/DEL/blob/master/report.pdf` (2016), accessed 28/06/2022
36. Wooldridge, M., Lomuscio, A.: A computationally grounded logic of visibility, perception, and knowledge. Logic Journal of IGPL **9**(2), 257–272 (2001)

A Equivalence between the relational semantics

For any program $P \in \mathcal{PL}$ and $W \in \mathcal{P}(\mathcal{U})$, we have

$$F(P, W) = R^*_W(P, W).$$

Proof. The proof is done by induction on the structure of P. The difficult case is that of $P; Q$. We have

$$\begin{aligned}
R^*_W(P;Q,W) &= \bigcup_{s\in W}\left\{\bigcup_{s'\in R_W(P,s)}\{R_{R^*_W(P,W)}(Q,s')\}\right\} && \text{def of } R_W(P;Q,\cdot)\\
&= \bigcup_{s\in W}\left\{\bigcup_{s'\in R_W(P,s)}\{R_{F(P,W)}(Q,s')\}\right\} && F(P,W) = R^*_W(P,W)\\
&= \bigcup_{s\in W}\left\{R^*_{F(P,W)}(Q,R_W(P,s))\right\} && \text{by ind hypothesis on } P\\
&= R^*_{F(P,W)}(Q,R^*_W(P,W)) && R^*_W \text{ is the post-image of } R_W\\
&= R^*_{F(P,W)}(Q,F(P,W)) && F(P,W) = R^*_W(P,W)\\
&= F(Q,F(P,W)) && \text{by induction hypothesis on } Q.
\end{aligned}$$

■

B Proof of Proposition 1

Proposition 1 makes the correspondence between our weakest precondition semantics and our relational semantics, as follows.

Proposition 1. *For every program P and every formula $\alpha \in \mathcal{L}^m_{DK}$,*

$$F(P,W) \models \alpha \quad \text{iff} \quad W \models wp(P,\alpha).$$

Proof. Case $\beta?$

$$\begin{aligned}
& W \models wp(\beta?,\alpha)\\
\equiv\ & W \models [\beta]\alpha && \text{the definition of } wp(\beta?,\cdot)\\
\equiv\ & \forall s\in W, (W,s) \models [\beta]\alpha && \text{by the definition of} \models \text{on a model}\\
\equiv\ & \forall s\in W,\ \text{if } (W,s)\models\beta \text{ then } (W_{|\beta},s)\models\alpha && \models \text{for public announcement}\\
\equiv\ & \forall s\in W,\ \text{if } (W,s)\models\beta \text{ then } (\{s'\in W|(W,s')\models\beta\},s)\models\alpha && \text{def of } W_{|\beta}\\
\equiv\ & \forall s\in W,\ \text{if } s\in F(\beta?,W) \text{ then } (F(\beta?,W),s)\models\alpha && \text{by definition of } F(\beta?,\cdot)\\
\equiv\ & F(\beta?,W)\models\alpha && \text{by the definition of} \models \text{on a model}
\end{aligned}$$

Case $P \sqcup Q$

$$\begin{aligned}
& F(P\sqcup Q,W)\models\alpha\\
\equiv\ & \{s[c_{Ag}\mapsto l]|s\in F(P,W)\}\cup\{s[c_{Ag}\mapsto l]|s\in F(Q,W)\}\models\alpha\\
& && \text{the definition of } F(P\sqcup Q,\cdot)\\
\equiv\ & \{s[c_{Ag}\mapsto l]|s\in F(P,W)\}\models\alpha \text{ and } \{s[c_{Ag}\mapsto l]|s\in F(Q,W)\}\models\alpha\\
& && \text{by Prop 2.3 in [5], this is a disjoint union since } c_{Ag} \text{ observable by all}\\
\equiv\ & F(P,W)\models\alpha \text{ and } F(Q,W)\models\alpha && c_{Ag} \text{ is not in } \alpha\\
\equiv\ & W\models wp(P,\alpha) \text{ and } W\models wp(Q,\alpha) && \text{by induction hypothesis on } P \text{ and } Q
\end{aligned}$$

Case $P;Q$ $\quad F(P;Q,W) \models \alpha \equiv F(Q,F(P,W)) \models \alpha$ definition of F for $P;Q$

$$\begin{aligned} &\equiv F(P,W) \models wp(Q,\alpha) && \text{induction hypothesis on } Q\\ &\equiv W \models wp(P,wp(Q,\alpha)) && \text{induction hypothesis on } P \end{aligned}$$

Case **new** $k \cdot P$

$$\begin{aligned} & W \models wp(\mathbf{new}\ k\cdot P,\alpha)\\ \equiv\ & \text{for any } s\in W,\ (W,s)\models \forall k\cdot wp(P,\alpha) && \text{the definition of } wp \text{ for } \mathbf{new}\ k\\ \equiv\ & \text{for any } s\in W \text{ and any } c\in D,\ (\textstyle\bigcup_{d\in\mathsf{D}} W[k\mapsto d], s[k\mapsto c])\models wp(P,\alpha)\\ & && \text{by definition of } \models \text{ for } \forall k\\ \equiv\ & \text{for any } s'\in \textstyle\bigcup_{d\in\mathsf{D}} W[k\mapsto d],\ (\bigcup_{d\in\mathsf{D}} W[k\mapsto d], s')\models wp(P,\alpha)\\ \equiv\ & \textstyle\bigcup_{d\in D} W[k\mapsto d]\models wp(P,\alpha) && \text{by lifting } \models \text{ to the entire model}\\ \equiv\ & F(P,\textstyle\bigcup_{d\in D} W[k\mapsto d])\models\alpha && \text{by induction hypothesis on } P\\ \equiv\ & F(\mathbf{new}\ k\cdot P,W)\models\alpha && \text{the definition of } F(\mathbf{new}\ k,\cdot). \end{aligned}$$

Case $x_G := e$. To understand the proof, observe that the action of $F(x_G := e,\cdot)$ on W, is equivalent to renaming the old x_G into k_G, then making a new variable x_G that takes the value e. We also need to derive the following equality

$$\begin{aligned} & F(x_G := e, W)\\ =\ & \{s[k_G\mapsto s(x_G), x_G\mapsto s(e)] \mid s\in W\} && \text{by definition of } F(x_G := e,\cdot)\\ =\ & \{s[x_G\mapsto s(e_{x_G\backslash k_G})] \mid s\in W_{x_G\backslash k_G}\} && \text{by definition of } W_{x_G\backslash k_G}\\ =\ & (\textstyle\bigcup_{d\in\mathsf{D}} W_{x_G\backslash k_G}[x_G\mapsto d])_{|d=s(e_{x_G\backslash k_G})} && \text{because } x_G \text{ is not in } \mathsf{dom}(W_{x_G\backslash k_G})\\ =\ & F((x_G = e_{x_G\backslash k_G})?, \textstyle\bigcup_{d\in\mathsf{D}} W_{x_G\backslash k_G}[x_G\mapsto d]) && \text{by definition of } F \text{ for tests}\\ =\ & F(\mathbf{new}\ x_G\cdot(x_G = e_{x_G\backslash k_G})?, W_{x_G\backslash k_G}) && \text{by definition of } F \text{ for } \mathbf{new}\ x_G. \end{aligned}$$

where $W_{x_G\backslash k_G}$ renames x_G into k_G in the states of W. Now,

$$\begin{aligned} & F(x_G := e, W)\models\alpha\\ \equiv\ & F(\mathbf{new}\ x_G\cdot(x_G = e_{x_G\backslash k_G})?, W_{x_G\backslash k_G})\models\alpha && \text{from the previous equality}\\ \equiv\ & F(\mathbf{new}\ k_G\cdot(k_G = e)?, W)\models\alpha_{x_G\backslash k_G} && \text{after swapping } x_G \text{ and } k_G \text{ (Lemma ??)}\\ \equiv\ & W\models wp(\mathbf{new}\ k_G\cdot(k_G = e)?, \alpha_{x_G\backslash k_G}) && \text{by induction hypothesis on } \mathbf{new}\ k_G\\ \equiv\ & W\models \forall k_G\cdot[k_G = e]\alpha_{x_G\backslash k_G} && \text{by the definition of } wp \text{ for assignment.} \quad \blacksquare \end{aligned}$$