An SMT-based Approach to the Verification of Knowledge-Based Programs

F. BELARDINELLI , I. BOUREANU , V. MALVONE , S. F. RAJAONA

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.

ACM Reference Format:

F. Belardinelli, , I. Boureanu, , V. Malvone, and , S. F. Rajaona. 2024. An SMT-based Approach to the Verification of Knowledge-Based Programs. 1, 1 (October 2024), 25 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION & PRELIMINARIES

The verification of knowledge properties, also known as epistemic properties, is becoming increasingly important in the design and analysis of real-life systems (e.g., electronic voting protocols, robots), especially with the rise of privacy concerns on the one side (e.g., anonymity, unlinkability) [8] and AI on the other [20]. This type of analysis of high-level descriptions of systems is most often done via formal methods and model checking [2]. By contrast, if we look across into the field of program verification, one is generally no longer looking at using model checking, but rather at interactive theorem-proving over program logics (such as Hoare logics [28]), or at using predicate transformers (e.g., strongest postconditions [14]) to reduce verification of program-logics statements to first-order queries fed into SMT solvers (e.g., Z3 [13]). In this paper, we will look at this precisely: *translating model checking of epistemic properties in programs into an SMT-solving problem.*

But, in this realm of knowledge-centric verification of programs, what are the important questions being asked? Consider threads A and B within the same program executing concurrently over the same variable space, but with each thread having access to only a subset of the global variables, in such a way that the two threads only have partial observability of the full variable space and this observability is not the same. Then, in our framework, we are interested in epistemic formulas such as "$K_A\Box_P\varphi$", meaning to reason if "at this current point, thread A knows whether after executing program P, 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., "$K_BK_A\Box_P\varphi$". Such rich statements allow us to reason about the threads' "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 program executes.

Author's address: F. Belardinelli; , I. Boureanu; , V. Malvone; , S. F. Rajaona.

Manuscript submitted to ACM

In this same domain, then it becomes important if the program specification/text is publicly known: that is, when interpreting a formula such as "$K_A \Box_P \varphi$", does one consider that the specification/text of program P is known to thread A? Of course, thread A "knows" some of P's variables and observes their values, but if thread A knows P then it can deduce more information from said values than in the case where thread A does not know P. To this end, our endeavour here is the following: *reducing model checking of privacy-centric properties to SMT solving for multi-threaded programs with publicly-known specifications, by first giving a program-epistemic logic which allows the expression of partial observability of program variables.*

Our method is closely related to a series of recent works. In 2017, [23] introduced a "*bespoke*" epistemic logic for programs. Under given conditions (e.g., set of program-instructions, variable domain, mathematical behaviour of program transformer), [23] proved that the model checking problem for their logic can be reduced to SMT-solving. In this work, we extend the line in [23] to overcome its limitations: (a) not being able to verify knowledge over programs which "looks ahead" into future states-of-affair; (b) not being able to reason about nested knowledge operators (e.g., $K_{alice}(K_{bob}\phi)$). Moreover, in 2023, [5] advanced a technique similar to ours also aimed at overcoming [23]'s limitations; however, [5] operates in different settings, including one where the text of the programs is not public. More on these aspects is discussed in our related-work section.

1.1 Our Contributions

By lifting the limitations of [23] listed above, we make the following contributions:

(1) We define a *multi-agent, program-epistemic logic* $\mathcal{L}^m_{DK}$, which is a dynamic logic [26, 38] whose base language is a multi-agent first-order epistemic logic [27], under an observability-based semantics (see 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 more expressive than the state-of-the-art.
(2) We introduce the 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.
(3) We give a sound translation of the truth of a program-epistemic logic into first-order truth (see Section 3).
(4) We implement the aforesaid translation to allow a fully-automated verification of our program-epistemic logic via SMT-solving (see Section 4).
(5) We verify the well-known Dining Cryptographer's protocol [10] and the epistemic puzzle called the "Cheryl's birthday problem" [18]. We report competitive verification results. Collaterally, we are also the first to give SMT-based verification of the "Cheryl's birthday problem" [18] (see Section 4).

1.2 Presenting These Results.

A short description of our approach was presented in [39]. In this manuscript, we present the full technique. This means that we added more discussions about the technical parts and extended the helper examples, we included all the proofs of our theoretical results, we added one example (with three more programs) to our implementation, and we compared in more details with prior works.

1.3 Preliminaries & Background

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

Epistemic Logics. Logics for knowledge or epistemic logics [27] 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. Interpreted systems were introduced [36] to nuance the possible-worlds semantics with agents who can perform private sharing of information; to this end, in interpreted systems, epistemic indistinguishability relations are at the level of agents' local states (as opposed to global 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., [9, 25, 45].

Logics of Visibility for Programs. Others [23, 35, 40] 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* [14].

Perfect vs Imperfect recall. For any of the aforesaid cases, 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 [26, 38]. 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.* [23] 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).

2 PROGRAM-EPISTEMIC LANGUAGES

We introduce the logics $\mathcal{L}_{FO}$, $\mathcal{L}^m_K$, and $\mathcal{L}^m_{DK}$. We note that this logic is not introduced for the purpose of studying its properties or for advancing logics, but for finding a better way to express epistemic properties of programs in a way that can also be verified in a mechanised way.

We start by describing agents, their variables and states, such that then we can formulate epistemic properties of states, and program-epistemic properties of states, respectively, in our logics.

2.1 Logics syntax

Agents and variables. We use $a, b, c, \ldots$ to denote agents, Ag to denote the whole agents' set, and G for a subset of Ag.

We consider a set *Var* of variables. We define formulae α over variables in *Var*. Variables can be evaluated over *domains of values*.

We use $\alpha[x\backslash t]$ for the *substitution* of variable x in α by another variable, formula or a value t.

Each variable x in *Var* is "indexed" 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, $\mathbf{o}_a = \{x_G \in Var \mid a \in G\}$, and $\mathbf{n}_a = Var \setminus \mathbf{o}_a$.

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

Example 2.1. The base language $\mathcal{L}_{\mathbb{N}}$ for integer arithmetic can be given as:

$$e ::= c \mid v \mid e \circ e \qquad \text{(terms)}$$

$$\pi ::= True \mid False \mid e = e \mid e < e \mid \pi \wedge \pi \mid \neg\pi \qquad \text{(formulas)}$$

where c is an integer constant; $v \in Var$; and $\circ ::= +, -, \times, /$.

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 2.2 ($\mathcal{L}_{FO}$). 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 $x_G \in Var$.

The other Boolean connectives $\vee$, $\rightarrow$, $\leftrightarrow$, and the existential quantifier $\exists$, 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 in Definition 2.3.

Definition 2.3 ($\mathcal{L}_{DK}^m$). 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

$$\alpha ::= \pi \mid \alpha \wedge \alpha \mid \neg\alpha \mid K_a\alpha \mid [\beta]\alpha \mid \forall x_G \cdot \alpha \mid \Box_P\alpha$$

where $\pi \in \mathcal{L}_{QF}$, $a \in Ag$, $\beta \in \mathcal{L}_K^m$ the fragment of $\mathcal{L}_{DK}^m$ without any program operator $\Box_P$, $x_G \in Var$, and P is a program.

We now detail on Definition 2.3. Each K_a is the epistemic operator for agent a; the epistemic formula $K_a\alpha$ reads "agent a knows that α". The public announcement formula $[\beta]\alpha$, in the sense of [17, 37], means "after every announcement of β, 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 $\mathcal{PL}$ of programs that we define in Section 2.2. Other connectives and the existential quantifier $\exists$ can be derived in a standard way as for Definition 2.2.

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

2.2 Programs syntax

We overload a subset *PVar* of the logic variables in *Var* to also denote *program variables*.

Definition 2.4 ($\mathcal{PL}$). The program-epistemic language $\mathcal{PL}$ is defined in BNF as follows:

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

where $x_G \in PVar$, $k_G \in PVar$ and does not appear before P, e is a term over $\mathcal{L}_{QF}$, and $\varphi \in \mathcal{L}_K^m$.

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; k_G is assigned arbitrarily before it is first used in P. The program $P; Q$ is the sequential composition of two programs P and Q. Lastly, the $P \sqcup Q$ is the nondeterministic choice between P and Q.

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

2.3 Logics semantics

2.3.1 States and the truth of $\mathcal{L}_{QF}$ formulas. We consider a domain D used for interpreting variables and quantifiers. A *state* s of the system is a valuation of the variables in *Var*, i.e., a partial function $s : Var \to \mathsf{D}$. We denote the universe of all possible 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 $s \models_{QF} \pi$. In particular, we assume that a state s is *adequate* for π, that is, all free variables in π are assigned some value in D by s.

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 2.5 (Truth of $\mathcal{L}_{FO}$-formulas). The truth of $\phi \in \mathcal{L}_{FO}$ at a state s, denoted $s \models_{FO} \phi$, where $FV(\phi) \subseteq \mathsf{dom}(s)$, is defined inductively on ϕ by

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

We lift the definition of $\models_{FO}$ to a set W of states, with $W \models_{FO} \phi$ iff for all $s \in W$, $s \models_{FO} \phi$.

2.3.2 Epistemic models. We model the agents' knowledge of the program state with a possible worlds semantics built on the observability of program variables [23]. We define, for each a in Ag, the binary relation $\approx_a$ on $\mathcal{U}$ by: $s \approx_a s'$ iff s and s' agree on the part of their domains that is observable by a, i.e.:

$$s \approx_a s' \text{ iff } (\mathbf{o}_a \cap \mathsf{dom}(s)) = (\mathbf{o}_a \cap \mathsf{dom}(s')) \text{ and for all } x \text{ in} (\mathbf{o}_a \cap \mathsf{dom}(s)), s(x) = s'(x)$$

Note that the definition above takes the intersection of $\mathbf{o}_a$ and $\mathsf{dom}(s)$, because states are partial functions over the variables.

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 $\{\approx_{a|W}\}_{a\in Ag}$ of equivalence relations.

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

Definition 2.6 (Truth of $\mathcal{L}^m_{DK}$-formulas). Let W be an epistemic model, $s \in W$ a state, α a formula in $\mathcal{L}^m_{DK}$ such that $FV(\alpha) \subseteq \mathsf{dom}(s)$

The truth of an epistemic formula α at the pointed model (W, s) is defined recursively on the structure of α as follows:

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

where $x_G \notin Var(W)$, where $Var(W) = \bigcup_{s\in W} Var(s)$, and $W_{|\beta}$ denotes the submodel of W that consists of the states in which β is true, i.e., $W_{|\beta} = \{s \in W \mid (W, s) \models \beta\}$.

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

Logical connectors, epistemic modality, and the public announcement modality have standard interpretation [6, 17]. In the following subsections, we explain our interpretation of the dynamic modality $\Box_P$ and the universal quantification.

2.3.4 On the semantics of the dynamic modality $\Box_P$. In our interpretation of $\Box_P\alpha$, the context W is also updated by the relation R_P, by taking the post-image of W by R_P[1]. The truth of α is interpreted at a post-state s' under the new context. We use the function $R_P(W, \cdot) : \mathcal{U} \to \mathcal{P}(\mathcal{U})$ to model the program P. We give the function $R_P(W, \cdot)$ concretely for each command P in the next section.

The argument W in $R_P(W, \cdot)$ is a set of states in $\mathcal{U}$. Similarly to relational semantics, $R_P(W, s)$ gives the set of states resulting from executing P at a state s. However, we need the set of states 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.

2.3.5 On the semantics of universal quantification. To evaluate the truth of $\forall x \cdot \alpha$, the epistemic context W is augmented by allowing x_G to be any 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 $s' \approx_a s$.

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

We now discuss this semantics, which may appear non-standard. For that, consider the two following possible definitions for universal quantifier in our program-epistemic logic. We will argue that the first is not appropriate for our case, and that the second is – by means of an example.

$$(W,s) \models \overline{\forall} x_G \cdot \alpha \text{ iff for all } c \in \mathsf{D}, (W[x_G \mapsto c], s[x_G \mapsto c]) \models \alpha \qquad (\overline{\forall}\text{-Definition})$$

$$(W,s) \models \forall x_G \cdot \alpha \text{ iff for all } c \in \mathsf{D}, (\bigcup_{d\in\mathsf{D}}\{s'[x_G \mapsto d] \mid s' \in W\}, s[x_G \mapsto c]) \models \alpha \qquad (\forall\text{-Definition})$$

In fact, $\overline{\forall}$ corresponds to a quantification over rigid objects. Intuitively, the universally quantified variable x_G is given the same value c at s and at all the other possible states in W. In contrast, $\forall$ corresponds to a quantification over non-rigid objects. The variable x_G is allowed to vary from state to state in W.

We need to quantify over non-rigid objects in our context of program-epistemic logic, particularly in our definition of weakest precondition for assignment, i.e., $wp(x_G := e, \alpha) = \forall k_G \cdot [k_G = e](\alpha[x_G \backslash k_G])$. This is illustrated in the following example.

Example 2.7. Let h be a variable of type $\mathbb{B}$ hidden from all agents, and let $P = x_G := h$. After the execution of P, an agent $a \in G$ who can observe x_G (and who knew the execution since programs are executed publicly) learns the value of h.

We abbreviate $K_a(h = 0) \vee K_a(h = 1)$ as $KV_a h$, which reads "a knows the value of h". Intuitively, we expect that

$$wp(x_G := h, KV_a h) \text{ evaluates to } True \text{ for any } a \in G \text{ and } wp(x_G := h, KV_b h) \text{ evaluates to } KV_b h \text{ for any } b \notin G, \qquad (1)$$

i.e., agent $a \in G$ would learn h after $x_G := h$ from any initial conditions. In contrast, $b \notin G$ would know h after executing $x_G := h$, only if they already knew h before the execution of $x_G := h$.

With our definition of wp for assignment, we get

$$wp(x_G := s, KV_a h) = \forall k_G \cdot [k_G = h].KV_a h[x_G \backslash k_G] = \forall k_G \cdot [k_G = n].KV_a h \qquad (2)$$

First, we show that with $\overline{\forall}$-Definition, the weakest precondition (2) does not align with the intuition (1). Consider a set $W = \{h_0, h_1\} = \{h \mapsto 0, h \mapsto 1\}$ of initial states. The truth of $\overline{\forall} k_G \cdot [k_G = h] KV_a h$ at (W, s_0) amounts to establishing $[k_G = h] KV_a h$ first at $(W[k_G \mapsto 0], s_0[k_G \mapsto 0])$, and then at $(W[k_G \mapsto 1], s_0[k_G \mapsto 1])$. Since $s_0[k_G \mapsto 0] \models_{FO} k_G = h$ and $s_0[k_G \mapsto 1] \not\models_{FO} k_G = h$, we are left to establish $(W[k_G \mapsto 0]_{|k_G=h}, s_0[k_G \mapsto 0]) \models KV_a h$, which holds iff $(\{(h, k_G) \mapsto (0,0)\}, (h, k_G) \mapsto (0,0)) \models KV_a h$. The latter always holds whether $a \in G$ or not, since $(\{(h, k_G) \mapsto (0,0)\}, (h, k_G) \mapsto (0,0)) \models K_a(h = 0)$ whether $a \in G$ or not (the only possible world satisfies $h = 0$). Thus, using $\overline{\forall}$-Definition, the weakest precondition (2) does not align with the intuition (1) that only agents in G would learn h from $x_G := h$

Now, with the $\overline{\forall}$-Definition, the truth of $\forall k_G \cdot [k_G = h] KV_a h$ at (W, s_0) amounts to establishing $[k_G = h].KV_a h$ first at $(W[k_G \mapsto 0] \cup W[k_G \mapsto 1], s_0[k_G \mapsto 0])$, and then at $(W[k_G \mapsto 0] \cup W[k_G \mapsto 1], s_0[k_G \mapsto 1])$. Since $s_0[k_G \mapsto 0] \models_{FO} k_G = h$ and $s_0[k_G \mapsto 1] \not\models_{FO} k_G = h$, we are left to establish $(W[k_G \mapsto 1]_{|k_G=h} \cup W[k_G \mapsto 0]_{|k_G=h}, s_0[k_G \mapsto 0]) \models KV_a h$. This is equivalent to $(\{(h \mapsto 0, k_G \mapsto 0), (h \mapsto 1, k_G \mapsto 1)\}, (h \mapsto 0, k_G \mapsto 0)) \models KV_a h$. The latter holds iff $a \in G$, since then $(h \mapsto 0, k_G \mapsto 0) \not\approx_a (h \mapsto 1, k_G \mapsto 1)$ (the value of k_G allows a to distinguish the two worlds). This satisfies our intuition (1).

2.4 Programs relational semantics

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 $R_P : \mathcal{U} \to \mathcal{P}(\mathcal{U})$, such that R_P maps an initial state s to a set of possible final states.

As per Subsection 2.3.4, 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 2.8 (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 $R_P(W, \cdot) : \mathcal{U} \to \mathcal{P}(\mathcal{U})$ defined inductively on the structure of P by

$$
\begin{aligned}
R_{\beta?}(W,s) &= \begin{cases} \{s\} & \text{if } (W,s) \models \beta; \\ \varnothing & \text{otherwise.} \end{cases} \\
R_{x_G := e}(W,s) &= \{(s[k_G \mapsto s(x_G)])[x_G \mapsto s(e)]\} \\
R_{\mathbf{new}\ k_G \cdot P}(W,s) &= R^*_P(\textstyle\bigcup_{d \in D} W[k_G \mapsto d], \{s[k_G \mapsto d] \mid d \in \mathsf{D}\}) \\
R_{P;Q}(W,s) &= \textstyle\bigcup_{s' \in R_P(W,s)} \{R_Q(R^*_P(W,W), s')\} \\
R_{P \sqcup Q}(W,s) &= \{s'[c_{Ag} \mapsto l] \mid s' \in R_P(W,s)\} \cup \{s'[c_{Ag} \mapsto r] \mid s' \in R_Q(W,s)\}
\end{aligned}
$$

such that k_G is not in $\mathsf{dom}(W)$, and c_{Ag} is not $\mathsf{dom}(R_P(W,s)) \cup \mathsf{dom}(R_Q(W,s))$.

We model nondeterministic choice $P \sqcup Q$ as a disjoint union [7], 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 $R_P(W,s)$, and a value r (for right) for every state in $R_Q(W,s)$. 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. It is known that disjoint-union of models preserves the truth of epistemic formulas, whilst simple union of epistemic models may not [7]. Our modelling of nondeterministic choice as disjoint union corresponds to allowing agents to see how nondeterministic choice are resolved when a program executes.

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. But, in our semantics, variables may be renamed (e.g., via assignment); that is to say, an agent has an implicit but not explicit form of 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 $R_P(W, \cdot)$ 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 2.9 (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

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

such that k_G is not in $\mathsf{dom}(W)$ and c_{Ag} is not in $\mathsf{dom}(F(P,W)) \cup \mathsf{dom}(F(Q,W))$.

The two relational semantics (Def. 2.8 and Def. 2.9) are equivalent (see Appendix B). However, we use both to simplify the presentation. On one hand, the relation on states given by $R_P(W,\cdot)$ is more standard for defining a dynamic formula $\Box_P\alpha$ (see e.g. [23]). 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.

We specifically note that the semantics of **new** $k_G \cdot P$ changes the domain of interpretation; this has an impact, of course, when this dynamic/program operator is mixed in with the epistemic connectives inside a formula, as we will be discussing more later in the manuscript.

2.5 Programs weakest precondition semantics

We now give another semantics for our programs, by lifting the Dijkstra's classical weakest precondition predicate transformer [2] [14] to epistemic predicates.

Definition 2.10. We define the weakest precondition of a program $P \in \mathcal{PL}$ as the epistemic predicate transformer $wp(P,\cdot) : \mathcal{L}^m_K \to \mathcal{L}^m_K$ with

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

for $\alpha \in \mathcal{L}^m_K$ such that $FV(\alpha) \subseteq PVar$, and k_G is not free in the expresion e.

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 [14]. A similar definition of wp for a new variable declaration is also found in [34]. 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 [35, 40]. Our wp semantics for test follows from the observation that an

[2]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 $wp(P,\phi)$.

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.

Linked to the note right after Definition 2.9, on the semantics of **new** $k_G \cdot P$, note that the wp-based interpretation of this adds a quantification over all variables introduced by **new** . This will later allow us to "keep track" of these variables inside super/sub-formulae.

We now discuss the case of our semantics for wp separately, arguing why our non-standard approach is needed, in two steps. First, we show that the classical semantics for wp would not be suited for us. Second, we give an intuition why we need our approach.

Example 2.11 below shows that the classical wp semantics (which we denote by $wp_{classical}$) does not capture the information flow in the assignment $x_A := h$, where x_A is observable by A and h is a secret.

Example 2.11. Consider a variable $x_A : \mathbb{B}$, observable by A, and a secret h. Assume we lifted the classical semantics $wp_{classical}$ –which is a substitution– to work with epistemic formulas, then we would have:

$$\begin{aligned} wp_{classical}(x_A := h, K_A(h=0)) &= K_A(h=0)[x_A \backslash h] \\ &= K_A(h=0) \qquad \text{(since } x_A \text{ does not appear in } K_A(h=0)\text{)}. \end{aligned}$$

Intuitively, this means that A knows that $h = 0$ after $x_A := h$ only if A already knows that $h = 0$. Thus, $wp_{classical}$ does not capture the leakage of the secret h in $x_A := h$.

However, using our wp semantics, we can deduce that $wp(x_A := h, K_A(h=0)) \;=\; (h=0)$. Indeed,

$$\begin{aligned} & wp(x_A := h, K_A(h=0)) \\ &= \forall u_A \cdot [u_A = s] K_A(h=0) \\ &= \forall u_A \cdot (u_A = h) \Rightarrow K_A(u_A = h \Rightarrow h = 0) && \text{reduction of PAL, see e.g., [17]} \\ &= \begin{cases} (h=0) \wedge (\forall u_A \cdot (u_A = h) \Rightarrow K_A(u_A = h \Rightarrow h = 0)) \\ \vee (h=1) \wedge (\forall u_A \cdot (u_A = h) \Rightarrow K_A(u_A = h \Rightarrow h = 0)) \end{cases} && \text{distribution of } True = (h = 0 \vee h = 1) \\ &= \begin{cases} (h=0) \wedge (\forall u_A \cdot (u_A = h \wedge h = 0) \Rightarrow K_A(u_A = h \Rightarrow h = 0)) \\ \vee (h=1) \wedge (\forall u_A \cdot (u_A = h \wedge h = 1) \Rightarrow K_A(u_A = h \Rightarrow h = 0)) \end{cases} && \text{absorption rule, } \forall \text{ and } \wedge \text{ commutes} \\ &= \begin{cases} (h=0) \wedge (\forall u_A \cdot (u_A = 0) \Rightarrow K_A(u_A = h \Rightarrow h = 0)) \\ \vee (h=1) \wedge (\forall u_A \cdot (u_A = 1) \Rightarrow K_A(u_A = h \Rightarrow h = 0)) \end{cases} && \text{transitivity of } = \\ &= \begin{cases} (h=0) \wedge (\forall u_A \cdot (u_A = 0) \Rightarrow K_A((u_A = h \wedge u_A = 0) \Rightarrow h = 0)) \\ \vee (h=1) \wedge (\forall u_A \cdot (u_A = 1) \Rightarrow K_A((u_A = h \wedge u_A = 1) \Rightarrow h = 0)) \end{cases} && \text{for } u_A \text{ observable by } A \\ &= \begin{cases} (h=0) \wedge (\forall u_A \cdot (u_A = 0) \Rightarrow K_A(True)) \\ \vee (h=1) \wedge (\forall u_A \cdot (u_A = 1) \Rightarrow K_A(False)) \end{cases} && \text{transitivity of } = \\ &= h = 0. \end{aligned}$$

Thus, our weakest precondition captures the intuition that after $x_A := h$, agent A learns that $h = 0$ if it is the case. Note, that the derivation above does not work for an agent B who does not observe x_A, as the second-to-last step would fail.

So, the reason we need a stronger semantics for wp is that in our case the knowledge of agent A comes compoundly: (a) – knowing the "program text"; (b) how the program (e.g., an assignment) affects an observable variable. This is richer than in the standard cases for wp, where knowledge is not of concern and, in essence, where case (b) counts –as

the example above shows. To control A's knowledge more under our setting of public "program texts", we introduced the new semantics for wp.

2.6 Equivalence between program relational semantics and weakest semantics

The following equivalence shows that our weakest precondition semantics is sound with respect to the program relational model.

Proposition 2.12. *For every program P and every formula $\alpha \in \mathcal{L}^m_{DK}$,*

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

Proof. The proof is done by induction on P.

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 $P \sqcup Q$. 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 [7]). A formula that is true at both $F(P, W)$ and $F(Q, W)$, remains true at $F(P \sqcup Q, W)$. Formally, we have

$$\begin{aligned}
& F(P \sqcup Q, W) \models \alpha \\
\equiv\ & \{s[c_{Ag} \mapsto l] | s \in F(P, W)\} \cup \{s[c_{Ag} \mapsto l] | s \in F(Q, W)\} \models \alpha && \text{the definition of } F(P \sqcup Q, \cdot) \\
\equiv\ & \{s[c_{Ag} \mapsto l] | s \in F(P, W)\} \models \alpha \text{ and } \{s[c_{Ag} \mapsto l] | s \in F(Q, W)\} \models \alpha \\
& && \text{by Prop 2.3 in [7], this is a disjoint union since } c_{Ag} \text{ observable by all} \\
\equiv\ & F(P, W) \models \alpha \text{ and } F(Q, W) \models \alpha && c_{Ag} \text{ is not in } \alpha \\
\equiv\ & W \models wp(P, \alpha) \text{ and } W \models wp(Q, \alpha) && \text{by induction hypothesis on } P \text{ and } Q.
\end{aligned}$$

Case $P; Q$.

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

Case **new** $k_G \cdot P$.

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

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. This is captured by the following equality $F(x_G := e, W) = F(\mathbf{new}\ x_G \cdot (x_G = e_{x_G \backslash k_G})?, W_{x_G \backslash k_G})$. In the right-hand side of this equality, x_G is re-introduced as a new variable. This new variable expands the model W by a Cartesian product, into $\bigcup_{d \in D} W[x_G \mapsto d]$ (Definition 2.9). The model W is then restricted to satisfy $x_G = e_{x_G \backslash k_G}$. This restriction corresponds to the semantics of an assumption (or public announcement) $(x_G = e_{x_G \backslash k_G})?$. Finally, $F(\mathbf{new}\ x_G \cdot (x_G = e_{x_G \backslash k_G})?, W_{x_G \backslash k_G})$ can be directly to the weakest precondition for assignment via Lemma A.1.

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

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

$$
\begin{aligned}
& F(x_G := e, W) \models \alpha \\
\equiv\ & F(\mathbf{new}\ x_G \cdot (x_G = e_{x_G \backslash k_G})?, W_{x_G \backslash k_G}) \models \alpha && \text{from the previous equality} \\
\equiv\ & F(\mathbf{new}\ k_G \cdot (k_G = e)?, W) \models \alpha_{x_G \backslash k_G} && \text{after swapping } x_G \text{ and } k_G \text{ (Lemma A.1)} \\
\equiv\ & W \models wp(\mathbf{new}\ k_G \cdot (k_G = e)?, \alpha_{x_G \backslash k_G}) && \text{by induction hypothesis on } \mathbf{new}\ k_G \\
\equiv\ & W \models \forall k_G \cdot [k_G = e] \alpha_{x_G \backslash k_G} && \text{by the definition of } \boldsymbol{wp} \text{ for assignment}
\end{aligned}
$$

■

The equivalence in Prop 2.12 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 with respect to the program relational models.

3 REDUCTION TO FIRST-ORDER VALIDITY

Our verification approach relies on the truth-preserving translation between program-epistemic formulas and first-order formulas. The translation of an $\mathcal{L}^m_{DK}$ formula is defined at a given epistemic context. Recall that the epistemic context is a set of reachable or epistemically relevant states. This context is now given as the satisfaction set of a first-order formula ϕ, in which the free variables are program variables in *PVar*. The satisfaction set of ϕ, denoted by $[[\phi]]_{PVar}$, or simply $[[\phi]]$ when *PVar* is clear from the context, is defined by $\{s : PVar \to \mathsf{D} \mid s \models_{FO} \phi\}$.

Definition 3.1 (Translation of $\mathcal{L}^m_{DK}$ into $\mathcal{L}_{FO}$). We define the translation τ of an $\mathcal{L}^m_{DK}$ formula α, at a given context ϕ inductively on the structure of α, as follows

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

where $\pi \in \mathcal{L}_{QF}$ and $\alpha, \alpha' \in \mathcal{L}^m_{DK}$, $\circ$ be an operator in $\{\wedge, \vee\}$, a an agent, $\mathbf{n} = \mathbf{n}_a \cap (FV(\alpha) \cup FV(\phi))$ is the set of free variables in ϕ and α that are non-observable by a, P is a program in $\mathcal{PL}$, and x_G is a variable not free in ϕ.

Let us pause on some cases of this translation. First, the epistemic modality K_a is translated using quantification over the non-observable variables in $\mathbf{n}$ as the latter encode the indistinguishability relation $\approx_a$. Note that if α contains **new** that would have first introduced some variables k (by the definition and semantics of **new** in Def. 2.9 and Def. 2.10). That would have augmented the domain of interpretation also. And, this $\forall$ will (recursively) quantify potentially also over the variables k introduced by α earlier. Similarly, if α is of the form $\Box_P\alpha'$ and P contains assignment, then –by our treatment of assignment – this reduces to same case as **new** . In turn, this means that even if we use SSA (Single Static Assignment) [41] and we store in a given variable x only its latest value, we have all its previous values in **new** -introduced variables k and those are "book-kept" (and quantified over if needed) via our semantics and translation.

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

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

Proof. The proof for the base epistemic logic without public announcement $\mathcal{L}_K$ $(\pi, \neg, \wedge, K_a)$ is found in [23].

Case of public announcement $[\beta]\alpha$.

$$
\begin{aligned}
& ([[\phi]], s) \models [\beta]\alpha \\
\equiv\ & \text{if } ([[\phi]], s) \models \beta \text{ then } ([[\phi]]_{|\beta}, s) \models \alpha && \text{truth of } [\beta]\alpha \\
\equiv\ & \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } ([[\phi]]_{|\beta}, s) \models \alpha && \text{induction hypothesis on } \beta \\
\equiv\ & \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } (\{s' : PVar \to \mathsf{D} \mid s' \models_{FO} \phi \text{ and } ([[\phi]], s') \models \beta\}, s) \models \alpha && \text{by definition of } [[\cdot]] \text{ and definition of } {}_{|\beta} \\
\equiv\ & \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } (\{s' : PVar \to \mathsf{D} \mid s' \models_{FO} \phi \text{ and } s' \models_{FO} \tau(\phi, \beta)\}, s) \models \alpha && \text{induction hypothesis on } \beta \\
\equiv\ & \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } (\{s' : PVar \to \mathsf{D} \mid s' \models_{FO} \phi \wedge \tau(\phi, \beta)\}, s) \models \alpha && \text{truth of } \wedge \\
\equiv\ & \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } ([[\phi \wedge \tau(\phi, \beta)]], s) \models \alpha && \text{def of } [[\cdot]] \\
\equiv\ & \text{if } s \models_{FO} \tau(\phi, \beta) \text{ then } s \models_{FO} \tau(\phi \wedge \tau(\phi, \beta), \alpha) && \text{induction hypothesis} \\
\equiv\ & \text{if } s \models_{FO} \tau(\phi, \beta) \to \tau(\phi \wedge \tau(\phi, \beta), \alpha) && \text{truth of } \to \\
\equiv\ & \text{if } s \models_{FO} \tau(\phi, [\beta]\alpha) && \text{by definition of } \tau
\end{aligned}
$$

Case of quantification $\forall x_G \cdot \alpha$.

$$
\begin{aligned}
& ([[\phi]]_{PVar}, s) \models \forall x_G \cdot \alpha \\
\equiv\ & \text{iff for all } c \in \mathsf{D}, (\textstyle\bigcup_{d \in \mathsf{D}} \{s'[x_G \mapsto d] \mid s' \in [[\phi]]_{PVar}\}, s[x_G \mapsto c]) \models \alpha && \text{Def. 2.6 for } \forall \\
\equiv\ & \text{iff for all } c \in \mathsf{D}, ([[\phi]]_{PVar \cup \{x_G\}}, s[x_G \mapsto c]) \models \alpha && \text{since } \textstyle\bigcup_{d \in \mathsf{D}} \{s'[x_G \mapsto d] \mid s' \in [[\phi]]_{PVar}\} = [[\phi]]_{PVar \cup \{x_G\}} \\
\equiv\ & \text{iff for all } c \in \mathsf{D}, s[x_G \mapsto c] \models_{FO} \tau(\phi, \alpha) && \text{induction hypothesis} \\
\equiv\ & \text{if } s \models_{FO} \forall x_G \cdot \tau(\phi, \alpha) && \text{Def. 2.5 for } \forall \\
\equiv\ & \text{if } s \models_{FO} \tau(\phi, \forall x_G \cdot \alpha) && \text{by definition of } \tau
\end{aligned}
$$

■

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

Theorem 3.3 (Main result). *Let $\phi \in \mathcal{L}_{FO}$, and $\alpha \in \mathcal{L}^m_{DK}$, such that $FV(\phi) \cup FV(\alpha) \subseteq PVar$, then*

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

Proof. The proof is done by induction on α. The case where $\alpha \in \mathcal{L}^m_K$ follows directly from Proposition 3.2.

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., $\alpha \in \mathcal{L}^m_K$. Indeed, one can show that $\Box_P(\Box_Q\alpha')$ is equivalent to $\Box_{P;Q}\alpha'$. We have

$$
\begin{aligned}
& [[\phi]] \models \Box_P\alpha \\
\equiv\ & \text{iff for all } s \text{ in } [[\phi]],\ ([[\phi]], s) \models \Box_P\alpha && \text{by definition of } \models \text{ for a model} \\
\equiv\ & \text{iff for all } s \text{ in } [[\phi]], \text{for all } s' \text{ in } R_{[[\phi]]}(P, s),\ (F(P, [[\phi]]), s') \models \alpha && \models \text{ for } \Box_P \\
\equiv\ & \text{iff for all } s' \text{ in } R^*_{[[\phi]]}(P, [[\phi]]),\ (F(P, [[\phi]]), s') \models \alpha && \text{post-image} \\
\equiv\ & \text{iff for all } s' \text{ in } F(P, [[\phi]]),\ (F(P, [[\phi]]), s') \models \alpha && F(P, W) = R^*_W(P, W) \\
\equiv\ & F(P, [[\phi]]) \models \alpha && \text{by definition of } \models \text{ for a model} \\
\equiv\ & [[\phi]] \models wp(P, \alpha) && \text{by Proposition 2.12} \\
\equiv\ & [[\phi]] \models_{FO} \tau(\phi, wp(P, \alpha)) && \text{since } wp(P, \alpha) \in \mathcal{L}^m_K, \text{ the previous case applies.}
\end{aligned}
$$

■

4 IMPLEMENTATION

Our automated verification framework supports proving/falsifying a logical consequence $\phi \models \alpha$ for α in $\mathcal{L}^m_{DK}$ and ϕ in $\mathcal{L}_{FO}$. By Theorem 3.3, 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 $\mathcal{L}^m_{DK}$-to-$\mathcal{L}_{FO}$ translator to automate the translation.

4.1 Mechanisation of Our $\mathcal{L}^m_{DK}$-to-FO Translation

Our translator implements Definition 3.1 of our translation τ. It is implemented in `Haskell`, and it is generic, i.e., works for any given example[3]. 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. 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.

4.2 Case Study 1: Dining Cryptographers' Protocol [10]

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.

[3]Inputs are `Haskell` files.

Encoding in $\mathcal{L}^m_{DK}$ & 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 $\mathbf{o}_i = \{x_{Ag}, p_i, c_{\{i-1,i\}}, c_{\{i,i+1\}}\}$ [4] , 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 $\rho = x_{Ag} := e$.

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 [23] as well. Importantly, formula γ cannot be expressed or checked by the framework in [23]. We compare the performance of our translation on this case-study with that of [23]. 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 [23] (denoted τ_{SP}) on the same machine.

Note that the performance we got for τ_{SP} differs from what is reported in [23]. 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 [23].

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 [23]. Experiments that took more than 600 seconds were timed out.

4.3 Case Study 2: Cheryl's Birthday Puzzle [18]

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:

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

	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 [23] *SP*-based translation for the Dining Cryptographers. Formula γ is not supported by the *SP*-based translation in [23].

- 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 $\langle\cdot\rangle$ [5]. 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 $\mathcal{L}^m_{DK}$ 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 $(\phi, s) \models \alpha$.

4.3.1 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).

4.4 Case Study 3: The Pit Card Game

In this we apply our logic and programming language to describe scenarios and actions in card games. We specifically treat a simplified version of the Pit game [1], which was also studied in the setting of Epistemic Logics in [15] and [43].

[5]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 $(W_{|\beta}, s) \models \alpha$.

Consider a deck of two Wheat, two Flax, and two Rye cards (w, x, y). Three players Anne, Bob, and Cath (a, b, and c) draw two cards from the deck. We denote the cards held by the players a, b, and c respectively by (la, ra), (lb, rb), and (lc, rc). Only a can see her cards (la, ra), etc. The setup is common knowledge to all agents.

The goal of each player is to establish a corner in a commodity, i.e., to have cards of the same suits. We assume that nobody achieved a corner from the initial deal.

In our implementation, we represent the cards (w, x, y) by the prime numbers $(2, 3, 5)$. The context ϕ is given by

$$\begin{aligned}\phi \coloneqq & \textstyle\bigwedge_{card \in \{la,ra,lb,rb,lc,rc\}} (card = 2 \vee card = 3 \vee card = 5) \\ & \wedge (la \times ra \times lb \times rb \times lc \times rc = 900) \\ & \wedge (la \neq ra \wedge lb \neq rb \wedge lc \neq rc).\end{aligned}$$

4.4.1 Simple cards swap. For simplicity, we omit the subscript a (which indicates the group of observing agents) from the variable la_a. Note also that in our implementation file `ExamplePit.hs`, variables are labelled with the agents that cannot observe them, rather than the agents that observe them.

First, we consider the action $swap_1$, in which a and b swap the card on their left, i.e., la and lb. To achieve this swap, a and b both put the required card face down on the table. Then a takes b's card from the table, and b's takes a's card.

$$swap_1 \coloneqq \mathbf{new}\ n_{\{\}} \cdot \mathbf{new}\ m_{\{\}} \cdot n \coloneqq la; m \coloneqq lb; la \coloneqq m; lb \coloneqq n$$

Two new variables, unobservable to all, are created to store la and lb. A more accurate representation of this scenario would use simultaneous assignments, $(la, n) \coloneqq (\varnothing, la); (lb, m) \coloneqq (\varnothing, lb); \ldots$ (a would no longer have la by putting it on the table). However, $swap_1$ is enough for our purpose, as the intermediate value of la and lb (nothing) adds no more information.

We performed the model checking $[[\phi]] \models \alpha$ of several formulas using the validity $[[\phi]] \models_{FO} \tau(\phi, \alpha)$ from our Main Theorem. We report some of the results as obtained with our tool, more can be found on the implementation file `ExamplePit.hs`. For instance,

$[[\phi]] \models \Box_{swap_1} \mathrm{Kv}_a lb$ (a always know the value of lb after $swap_1$)

$[[\phi]] \not\models \Box_{swap_1} \neg K_a(lb \neq rb)$ (a may learn that $lb \neq rb$ after $swap_1$, e.g.: $\overset{\frown}{(3, 2)}$, $(3, 5)$, $(2, 5)$))

4.4.2 Nondeterministic swap. Secondly, we consider the action $swap_2$, in which a and b nondeterministically swap one of their card by putting their chosen card face down on the table.

Define a function $swap$ yielding a program that swaps two cards by putting face down on the table first.

$$swap(\gamma, \gamma') \coloneqq \mathbf{new}\ n_{\{\}} \cdot \mathbf{new}\ m_{\{\}} \cdot n \coloneqq \gamma; m \coloneqq \gamma'; \gamma \coloneqq m; \gamma' \coloneqq n$$

Now, $swap_2$ is given by

$$swap_2 = swap(la, lb) \sqcup swap(la, rb) \sqcup swap(ra, lb) \sqcup swap(ra, rb)$$

Again, we report some of the results from model checking program-epistemic formulas with $swap_2$

$[[\phi]] \not\models \Box_{swap_2} \mathrm{Kv}_a lb$ (a may not know the value of lb after $swap_2$, as lb is not necessarily swapped)

$[[\phi]] \models \Box_{swap_2} (\mathrm{Kv}_a lb \vee \mathrm{Kv}_a rb)$ (a always learn either lb or rb after $swap_2$)

4.4.3 Nondeterministic visible swap. Lastly, we consider the action $swap_3$, in which a and b swap one of their card nondeterministically by putting their chosen card, this time face up on the table. Unlike the two previous swaps, c can see the cards being swapped.

As for $swap_2$, we define a function $swap'$ for swapping two cards, with the difference that the new variables are observed by all.

$$swap'(\gamma, \gamma') := \mathbf{new}\ n_{\{a,b,c\}} \cdot \mathbf{new}\ m_{\{a,b,c\}} \cdot n := \gamma; m := \gamma'; \gamma := m; \gamma' := n$$

Now, we define $swap_3$ by

$$swap_3 = swap'(la, lb) \sqcup swap'(la, rb) \sqcup swap'(ra, lb) \sqcup swap'(ra, rb)$$

Again, we report some of the results from model checking program-epistemic formulas with $swap_3$.

$[\![\phi]\!] \models \Box_{swap_3} \mathrm{Kv}_c lb \vee \mathrm{Kv}_c rb$ (c always learns either lb or rb after $swap_3$)

$[\![\phi]\!] \not\models \Box_{swap_3} \neg K_c(lb = rb)$ (c may know b has a corner after $swap3$, e.g., $\overset{\curvearrowleft}{(2,3)}, \overset{\curvearrowright}{(5,2)}, (3,5)$)

$[\![\phi]\!] \not\models \Box_{swap_3} \neg K_c(lb \neq rb)$ (c may know that b does not have a corner after $swap3$, e.g., $\overset{\curvearrowleft}{(5,2)}, \overset{\curvearrowright}{(3,2)}, (3,5)$)

4.4.4 Experiments & Results. All the formulas that we tested for the three programs $swap_1$, $swap_2$, and $swap_3$ were solved under 0.2 seconds.

5 RELATED WORK

5.1 On SMT-Based Verification of Epistemic Properties of Programs.

We compare with the work of Gorogiannis *et al.* [23] which we extend, as well as a very recent work, in [5], which also improves on [23]. We discuss several aspects, comparing us and these two works.

General Logic-based Approach and Expressivity. Gorogiannis *et al.* [23] 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), and having an epistemic predicate logic as its base logic. Interestingly, à la [35, 40, 45], the epistemic model in [23] 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 [23] only supports "classical" programs; this means that [23] cannot support tests on agents' knowledge. Yet, such tests are clearly in AI-centric programs: e.g., in epistemic puzzles [31], in the so-called "knowledge-based" programs in [21], etc. Secondly, the logic in [23] 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 [23] does not allow for reasoning about nested knowledges operators (e.g., $K_{alice}(K_{bob}\phi)$).

Belardinelli et al., in [5], defined a *program-epistemic logic* $\mathcal{L}_{\mathrm{PK}}$ that is strictly more expressive than the program-epistemic logic $\mathcal{L}_{\Box\mathrm{K}}$ in [23]: i.e., in $\mathcal{L}_{\mathrm{PK}}$, the epistemic and the knowledge operators can commute. [5]'s program-epistemic logic $\mathcal{L}_{\mathrm{PK}}$ is more general than the logic in [23]: our relational semantics is not dependent on programs' predicate transformers, and our programs are fully mapped to logic operators. In that sense, [5]'s logic $\mathcal{L}_{\mathrm{PK}}$ can be seen as an extension of star-free linear dynamic logic (LDL) [12] with epistemic operators, or equivalently dynamic logic (DL) [26]

	[5]	[23]	this work
1.	K possible before $\Box_P$, only one agent	K possible only after $\Box_P$, only one agent	K possible before $\Box_P$, multiple agents, using disjoint choice
2.	unknown if program is public	unknown if program is public	program is public
3.	no announcements	no announcements	public announcements
4.	multiple assignments via substitutions	multiple assignments	single assignment
5.	asymptotic complexity drop into $O(2^x)$, due to K possible before $\Box_P$	asymptotic complexity in $O(x)$	asymptotic complexity kept in $O(x)$, via single assignment
,			

Table 2. This work vs. [5] & [23]: Main Comparisons (where x is the size of the translated formula)

extended with an epistemic operator. In [5], since the logic is more aligned to "standard" logics (such as linear dynamic logic – LDL [12] and dynamic logic – DL [26]), the translation (unlike in [23]) is entirely recursive, without the need to leverage special cases separately and/or Hoare-style predicate transformers. Also, the AAAI-2023 paper [5] includes formulas that [23] could not treat: i.e., $K_a\Box_P\varphi$ – expressing that agent a knows fact φ about the execution of P.

Program Models. The program models in Gorogiannis *et al.* [23] follow a classical program semantics (e.g., modelling nondeterministic choice as union, overwriting a variable in reassignment). This has been shown [35, 40] 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. Belardinelli et al [5] do not have perfect recall, and agents cannot see how nondeterministic choices are resolved.

Program Expressiveness. Gorogiannis *et al.* [23] have results of approximations for programs with loops, although there were no use cases of that. [5] and this work are focused on a loop-free programming language. 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 [23] requires the strongest postcondition be computed manually. Like [23], 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).

In Table 2, we give a summary of the main differences between this work and the closest two other works:

Let us discuss Table 2. Row 1 refers to whether, in various works of this type, the program operator $\Box_P$ and epistemic operators K can commute, but also if there's one or more agents and therefore epistemic operators. Row 4 is concerned with whether different methods operate under a model where the assignments of variables introduce new, duplicate variables with each assignation, a.k.a. the single static assignment (SSA) assumption [41]. If this is the case, then –intuitively– the treatment of epistemic interpretations across program domains may be more easily handled. An alternative, as row 4 says, is to carefully introduce and use substitutions over variables inside program-epistemic formula to correspond or emulate variable assignments inside the actual programs. Row 5 shows that, in fact, these trade-offs mentioned in rows 1 and 4 lead, naturally, to different efficiency when it comes to program-epistemic formulae being translated into first-order ones in such a way, as here, where model checking of the former can be reduced to satisfaction of the latter. We also see in row 5, that this work uses single static assignment (SSA) to have both expressivity (i.e., commuting program operators $\Box_P$ and epistemic operators K, multiple agents), whilst maintaining the efficiency of less expressive formalisms. In the conference version of this paper [39] and in [5], we actually compare numerically in efficiency across these different methods and formalisms. Row 2 may also have an impact in the asymptotic complexity of these validity-checking reductions, but this aspect is less studied and we still cannot quantify the exact role it plays in the efficiency of these translations; this is a good avenue for future work. Of course, in Table 2, we could add more comparison criteria, but we selected what we deemed to be the defining one.

5.2 On Verification of Information Flow with Program Algebra.

Verifying epistemic properties of programs with program algebra was done in [33, 35, 40]. 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. Akin to how our relational semantics is shown to coincide with weakest-precondition semantics, [40] proves the laws of refinement sound w.r.t. Morgan's relational Shadow model [34], which also has its corresponding weakest-precondition semantics. The work in [40] 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 [40]. 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.

5.3 On Dynamic Epistemic Logics

Dynamic epistemic logic (DEL, [4, 17, 37]) is a family of logics that extend epistemic logic with dynamic operators.

Logics' Expressivity. DEL originates from public announcement logic [37], and the public announcement operator is one of its basic dynamic operator. On the one hand, DEL logics are mostly propositional, and their extensions with assignment only considered propositional assignment (e.g., [16]); 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` [19] and `SMCDEL` [42]. We are not aware of the verification of DEL fragments being reduced to satisfiability problems. In this space, an online report [44] 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 [9, 25, 45]. The work of Grossi [24] 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 [23] and ours.

5.4 On Other Aspects.

Gorogiannis *et al.* [23] discussed more tenuously related work, such as on general verification of temporal-epistemic properties of systems which are not programs in tools like `MCMAS` [32], `MCK` [22], `VERICS` [30], or one line of epistemic verification of models specifically of `JAVA` programs [3]. [23] also discussed some incomplete method of SMT-based epistemic model checking [11], or even bounded model checking techniques, e.g., [29]. All of those are loosely related to us too, but there is little reason to reiterate.

6 CONCLUSIONS

We advanced a multi-agent epistemic logic 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 with respect 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 [23], 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 with respect to 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 [23] and the memoryful semantics here, and methods of verifying logics like $\mathcal{L}^m_{DK}$ but with such less "absolutist" semantics.

REFERENCES

[1] Pit game rules. https://www.hasbro.com/common/instruct/pit.pdf, accessed: 2024-08-17

[2] Baier, C., Katoen, J.: Principles of model checking. MIT Press (2008)

[3] 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

[4] 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)

[5] Belardinelli, F., Boureanu, I., Malvone, V., Rajaona, F.: Automatically verifying expressive epistemic properties of programs. In: Williams, B., Chen, Y., Neville, J. (eds.) Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023. pp. 6245–6252. AAAI Press (2023). https://doi.org/10.1609/AAAI.V37I5.25769, https://doi.org/10.1609/aaai.v37i5.25769

[6] Blackburn, P., van Benthem, J.F., Wolter, F.: Handbook of modal logic. Elsevier (2006)

[7] Blackburn, P., de Rijke, M., Venema, Y.: Modal logic. Cambridge University Press, New York (2001)

[8] Boureanu, I., Jones, A.V., Lomuscio, A.: Automatic verification of epistemic specifications under convergent equational theories. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 1141–1148. AAMAS '12, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2012)

[9] 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)

[10] Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology **1**(1), 65–75 (1988)

[11] 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)

[12] De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. pp. 854–860. IJCAI '13, AAAI Press (2013)

[13] De Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Proc. of TACAS-14. pp. 337–340. Springer-Verlag (2008)

[14] Dijkstra, E.W.: A Discipline of Programming. Prentice-Hall (1976)

[15] 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, New York, NY, USA (2005), https://doi.org/10.1145/1082473.1082495

[16] 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)

[17] van Ditmarsch, H.P., Hoek, W.v.d., Kooi, B.: Dynamic Epistemic Logic. Synthese Library, Springer (2007)

[18] van Ditmarsch, H.P., Hartley, M.I., Kooi, B., Welton, J., Yeo, J.B.: Cheryl's birthday. arXiv preprint arXiv:1708.02654 (2017)

[19] van Eijck, J.: A demo of epistemic modelling. Interactive Logic p. 303 (2007)

[20] Ezekiel, J., Lomuscio, A., Molnar, L., Veres, S., Pebody, M.: Verifying fault tolerance and self-diagnosability of an autonomous underwater vehicle. In: Proc. of IJCAI-22. pp. 1659–1664. AAAI Press (2011)

[21] Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Knowledge-Based Programs. In: Symposium on Principles of Distributed Computing. pp. 153–163 (1995)

[22] 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

[23] 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

[24] 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)
[25] 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
[26] Harel, D.: Dynamic Logic, pp. 497–604. Springer Netherlands, Dordrecht (1984). https://doi.org/10.1007/978-94-009-6259-0_10
[27] Hintikka, J.: Knowledge and Belief. Cornell University Press (1962)
[28] Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (Oct 1969). https://doi.org/10.1145/363235.363259, https://doi.org/10.1145/363235.363259
[29] 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)
[30] 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)
[31] Lehman, D.: Knowledge, common knowledge, and related puzzles. In: Proc. of the 3rd ACM Symposium on Principles of Distributed Computing. pp. 62–67 (1984)
[32] 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
[33] 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)
[34] Morgan, C.: Programming from Specifications. Prentice Hall International Series in Computer Science, Prentice Hall, 2 edn. (1994)
[35] 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)
[36] Parikh, R., Ramanujam, R.: Distributed processing and the logic of knowledge. Lecture Notes in Computer Science **193**, 256–268 (1985)
[37] Plaza, J.A.: Logics of public communications. Proceedings of the 4th International Symposium on Methodologies for Intelligent Systems (1989)
[38] Pratt, V.R.: Semantical considerations on Floyd-Hoare logic. In: 17th Annual Symposium on Foundations of Computer Science. pp. 109–121. IEEE (1976)
[39] Rajaona, S.F., Boureanu, I., Malvone, V., Belardinelli, F.: Program semantics and verification technique for ai-centred programs. In: Chechik, M., Katoen, J., Leucker, M. (eds.) Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings. Lecture Notes in Computer Science, vol. 14000, pp. 473–491. Springer (2023). https://doi.org/10.1007/978-3-031-27481-7_27, https://doi.org/10.1007/978-3-031-27481-7_27
[40] Rajaona, S.F.: An algebraic framework for reasoning about privacy. Ph.D. thesis, Stellenbosch: University of Stellenbosch (2016)
[41] Rosen, B.K., Wegman, M.N., Zadeck, F.K.: Global value numbers and redundant computations. In: Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. p. 12–27. POPL '88, Association for Computing Machinery, New York, NY, USA (1988). https://doi.org/10.1145/73560.73562, https://doi.org/10.1145/73560.73562
[42] 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)
[43] Van Ditmarsch, H., Kooi, B.: Semantic results for ontic and epistemic change. Logic and the foundations of game and decision theory (LOFT 7) **3**, 87–117 (2008)
[44] Wang, S.: Dynamic epistemic model checking with Yices. https://github.com/airobert/DEL/blob/master/report.pdf (2016), accessed 28/06/2022
[45] Wooldridge, M., Lomuscio, A.: A computationally grounded logic of visibility, perception, and knowledge. Logic Journal of IGPL **9**(2), 257–272 (2001)

A LEMMAS

Lemma A.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*

$$(W, s) \models \alpha \text{ iff } (W_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)]) \models \alpha[x_G \backslash k_G].$$

Proof. The proof is by induction on the structure of α, starting from the base case $\alpha = \pi$:

$$\begin{aligned} (W, s) \models \pi \quad & \text{iff} \quad s \models_{QF} \pi \\ & \text{iff} \quad s[k_G \mapsto s(x_G)] \models_{QF} \pi[x_G \backslash k_G] \\ & \text{iff} \quad (W_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)]) \models \alpha[x_G \backslash k_G]. \end{aligned}$$

The inductive cases for Boolean connectives are immediate.

For $\alpha = K_a\beta$:

$$(W, s) \models \alpha \quad \text{iff} \quad \text{for all } s' \in W, s' \approx_a s \text{ implies } (W, s') \models \beta$$

Notice that $s' \approx_a s$ iff $s'[k_G \mapsto s(x_G)] \approx_a s[k_G \mapsto s(x_G)]$. Hence, by induction hypothesis,

$$\begin{aligned}(W, s) \models \alpha \quad &\text{iff} \quad \text{for all } s'' \in W_{x_G \backslash k_G}, s'' \approx_a s[k_G \mapsto s(x_G)] \text{ implies } (W_{x_G \backslash k_G}, s'') \models \beta[x_G \backslash k_G] \\ &\text{iff} \quad (W_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)]) \models K_a\beta[x_G \backslash k_G] = \alpha[x_G \backslash k_G].\end{aligned}$$

For $\alpha = [\beta']\beta$:

$$(W, s) \models \alpha \quad \text{iff} \quad (W, s) \models \beta' \text{ implies } (W_{|\beta'}, s) \models \beta$$

By induction hypothesis,

$$(W, s) \models \alpha \quad \text{iff} \quad (W_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)]) \models \beta'[x_G \backslash k_G] \text{ implies } ((W_{|\beta'})_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)]) \models \beta[x_G \backslash k_G]$$

Now notice that $(W_{|\beta'})_{x_G \backslash k_G} = (W_{x_G \backslash k_G})_{|\beta'[x_G \backslash k_G]}$. Hence,

$$\begin{aligned}(W, s) \models \alpha \quad &\text{iff} \quad (W_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)]) \models \beta'[x_G \backslash k_G] \text{ implies } ((W_{x_G \backslash k_G})_{|\beta'[x_G \backslash k_G]}, s[k_G \mapsto s(x_G)]) \models \beta[x_G \backslash k_G] \\ &\text{iff} \quad (W_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)]) \models [\beta'[x_G \backslash k_G]]\beta[x_G \backslash k_G] = \alpha[x_G \backslash k_G]\end{aligned}$$

For $\alpha = \Box_P\beta$:

$$(W, s) \models \alpha \quad \text{iff} \quad \text{for all } s' \in R_P(W, s), (R_P^*(W, W), s') \models \beta$$

Notice that $s' \in R_P(W, s)$ iff $s'[k_G \mapsto s(x_G)]] \in R_P(W_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)])$ and $R_P^*(W, W)_{x_G \backslash k_G} = R_P^*(W_{x_G \backslash k_G}, W_{x_G \backslash k_G})$. Hence, by induction hypothesis,

$$\begin{aligned}(W, s) \models \alpha \quad &\text{iff} \quad \text{for all } s'' \in R_P(W_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)]), (R_P^*(W_{x_G \backslash k_G}, W_{x_G \backslash k_G}), s'') \models \beta[x_G \backslash k_G] \\ &\text{iff} \quad (W_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)]) \models \Box_P\beta[x_G \backslash k_G] = \alpha[x_G \backslash k_G]\end{aligned}$$

For $\alpha = \forall x_G \cdot \beta$:

$$(W, s) \models \alpha \quad \text{iff} \quad \text{for all } c \in D, (\bigcup_{d \in D} \{s'[x_G \mapsto d] \mid s' \in W\}, s[x_G \mapsto c]) \models \beta$$

Now observe that $s' \in W$ iff $s'[k_G \mapsto s(x_G)] \in W_{x_G \backslash k_G}$, and $s[k_G \mapsto c] = s[x_G \mapsto c][k_G \mapsto s(x_G)]$. Hence, by induction hypothesis,

$$\begin{aligned}(W, s) \models \alpha \quad &\text{iff} \quad \text{for all } c \in D, (\bigcup_{d \in D} \{s'[x_G \mapsto d][k_G \mapsto s(x_G)] \mid s'[k_G \mapsto s(x_G)] \in W_{x_G \backslash k_G}\}, s[x_G \mapsto c][k_G \mapsto s(x_G)]) \models \beta[x_G \backslash k_G] \\ &\text{iff} \quad \text{for all } c \in D, (\bigcup_{d \in D} \{s'[k_G \mapsto d] \mid s' \in W_{x_G \backslash k_G}\}, s[k_G \mapsto c]) \models \beta[x_G \backslash k_G] \\ &\text{iff} \quad (W_{x_G \backslash k_G}, s[k_G \mapsto s(x_G)]) \models \forall x_G \cdot \beta[x_G \backslash k_G] = \alpha[x_G \backslash k_G]\end{aligned}$$

This completes the proof. ■

B EQUIVALENCE BETWEEN THE RELATIONAL SEMANTICS

Proposition B.1. *For any program $P \in \mathcal{PL}$ and $W \in \mathcal{P}(\mathcal{U})$, we have*

$$F(P, W) = R_P^*(W, W).$$

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

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

■