Knowledge-Based Programs as Plans - The Complexity of Plan Verification - (Extended Abstract)

Jérôme Lang, Bruno Zanuttini

Knowledge-Based Programs as Plans
– The Complexity of Plan Verification –
(Extended Abstract)

Jérôme Lang
LAMSADE, Université Paris-Dauphine & CNRS
lang@lamsade.dauphine.fr

Bruno Zanuttini
GREYC, Université de Caen Basse-Normandie,
CNRS UMR 6072, ENSICAEN
bruno.zanuttini@unicaen.fr

June 7, 2012

Abstract

Knowledge-based programs (KBPs) are high-level protocols describing the course of action an agent should perform as a function of its knowledge. The use of KBPs for expressing action policies in AI planning has been surprisingly underlooked. Given that to each KBP corresponds an equivalent plan and *vice versa*, KBPs are typically more succinct than standard plans, but imply more on-line computation time. Here we compare KBPs and standard plans according to succinctness and to the complexity of plan verification.

1 INTRODUCTION

Knowledge-based programs (KBPs) [4] are high-level protocols which describe the actions an agent should perform as a function of its knowledge, such as, typically, if $\mathbf{K}\varphi$ then π else π', where $\mathbf{K}$ is an epistemic modality and π, π' are subprograms.

Thus, in a KBP, branching conditions are epistemically interpretable, and deduction tasks are involved at execution time (on-line). KBPs can be seen as a powerful language for expressing policies or plans, in the sense that epistemic branching conditions allow for exponentially more compact representations. In contrast, standard policies (as in POMDPs) or plans (as in contingent planning) either are sequential or branch on objective formulas (on environment and internal variables), and hence can be executed efficiently, but they can be exponentially larger (see for instance [1]).

Having said this, KBPs have surprisingly been underlooked in the perspective of planning. Initially developed for distributed computing, they have been considered in

AI for agent design [14] and game theory [7]. For planning, the only works we know of are by Reiter [13], who gives an implementation of KBPs in Golog; Classen and Lakemeyer [3] who implement KBPs in a decidable fragment of the situation calculus; Herzig *et al.* [6], who discuss KBPs for propositional planning problems, and Laverny and Lang [9, 10], who generalize KBPs to *belief*-based programs allowing for uncertain action effects and noisy observations.

None of these papers really addresses computational issues[1]. Our aim is to contribute to filling this gap. After some background on epistemic logic (Section 2), we define KBPs (Section 3). Then we address expressivity and succinctness issues (Section 4): we show that, as expected, KBPs can be exponentially more compact than standard policies/plans. Then we give our main contributions, about the complexity of verifying that a KBP is a valid plan for a planning problem: we show Π_2^{p}-completeness for while-free KBPs (Section 5) and $\mathsf{EXPSPACE}$-completeness in the general case (Section 6).

2 KNOWLEDGE

A KBP is executed by an agent in an environment. We model what the agent *knows* about the current state (of the environment and internal variables) in the propositional epistemic logic S_5. Let $X = \{x_1, \ldots, x_n\}$ be a set of propositional symbols. A *state* is a valuation of X. For instance, $\overline{x_1}x_2$ is the state where x_1 is false and x_2 is true. A *knowledge state* M for S_5 is a nonempty set of states, representing those which the agent considers as possible: at any point in time, the agent has a knowledge state $M \subseteq 2^X$ and the current state is some $s \in M$. For instance, $M = \{x_1\bar{x}_2, \bar{x}_1x_2\}$ means that the agent knows x_1 and x_2 to have different values in the current state.

Formulas of S_5 are built up from X, the usual connectives, and the knowledge modality $\mathbf{K}$. An S_5 formula is *objective* if it does not contain any occurrence of $\mathbf{K}$. Objective formulas are denoted by φ, ψ, etc. whereas general S_5 formulas are denoted by Φ, Ψ etc. For an objective formula φ, we denote by $Mod(\varphi)$ the set of all states which satisfy φ (*i.e.*, $Mod(\varphi) = \{s \in 2^X, s \models \varphi\}$). The size $|\Phi|$ of an S_5 formula Φ is the total number of occurrences of propositional symbols, connectives and modality $\mathbf{K}$ in Φ.

It is well-known (see, *e.g.*, [4]) that any S_5 formula is equivalent to a formula without nested $\mathbf{K}$ modalities; therefore we disallow them. An S_5 formula Φ is *purely subjective* if objective formulas occur only in the scope of $\mathbf{K}$. In the whole paper we only need purely subjective formulas, because we are only interested in what the agent knows, not on the actual state of the environment. A purely subjective S_5 formula is in *knowledge negative normal form (KNNF)* if the negation symbol $\neg$ occurs only in objective formulas (in the scope of $\mathbf{K}$) or directly before a $\mathbf{K}$ modality. Any purely subjective S_5 formula Φ can be rewritten into an equivalently KNNF of polynomial size, by pushing all occurrences of $\neg$ that are out of the scope of $\mathbf{K}$ as far as possible with de Morgan's laws. For instance, $\mathbf{K}\neg(p \wedge q) \vee \neg(\mathbf{K}r \vee \mathbf{K}\neg r)$ is not in KNNF, but

[1]On the other hand, there is a much more significant literature on the complexity of planning under partial observability, *e.g.*, [12, 15], but plans there are expressed as standard policies and not as knowledge-based programs. As we explain further, knowledge-based programs are more succinct than standard policies.

is equivalent to $\mathbf{K}\neg(p\wedge q)\vee(\neg\mathbf{K}r\wedge\neg\mathbf{K}\neg r)$. Summarizing, a subjective S_5 formula Φ in KNNF (for short, $\Phi\in$ SKNNF) is either a positive (resp. negative) epistemic atom $\mathbf{K}\varphi$ (resp. $\neg\mathbf{K}\varphi$), where φ is objective, or a combination of such atoms using $\wedge,\vee$.

The satisfaction of a purely subjective formulas depends only on a knowledge state M, not on the *actual* current state (see, *e.g.*, [4]):

- $M\models\mathbf{K}\varphi$ if for all $s'\in M$, $s'\models\varphi$,
- $M\models\neg\mathbf{K}\varphi$ if $M\not\models\mathbf{K}\varphi$,
- $M\models\Phi\wedge\Psi$ if $M\models\Phi$ and $M\models\Psi$,
- $M\models\Phi\vee\Psi$ if $M\models\Phi$ or $M\models\Psi$.

An S_5 formula is *valid* (resp. *satisfiable*) if it is satisfied by all (resp. at least one) knowledge states $M\subseteq 2^X$. Given two S_5 formulas Φ and Ψ, Φ *entails* Ψ, written $\Phi\models\Psi$, if every knowledge state $M\subseteq 2^X$ which satisfies Φ also satisfies Ψ, and Φ is *equivalent* to Ψ if Φ and Ψ entail each other. Note that $\mathbf{K}\varphi\wedge\mathbf{K}\psi$ is equivalent to $\mathbf{K}(\varphi\wedge\psi)$, but that $\mathbf{K}\varphi\vee\mathbf{K}\psi$ is *not* equivalent to $\mathbf{K}(\varphi\vee\psi)$: for instance, $\mathbf{K}(\varphi\vee\neg\varphi)$ is valid whereas $\mathbf{K}\varphi\vee\mathbf{K}\neg\varphi$ is true only when the agent knows the value of φ. Also note that $\mathbf{K}\varphi$ entails $\neg\mathbf{K}\neg\varphi$, and that $M\models\neg\mathbf{K}\varphi$ is weaker than $M\models\mathbf{K}\neg\varphi$.

We use a syntactical representation of the current knowledge state $M\subseteq 2^X$ (at some timestep) of the agent executing a KBP. For this we observe that M can be identified with any objective formula φ which satisfies $M=Mod(\varphi)$, and we represent M by the epistemic atom $\mathbf{O}\varphi$. Intuitively, $\mathbf{O}\varphi$ means "I know φ and nothing else". Without loss of generality,we disallow occurences of $\mathbf{O}$ or $\mathbf{K}$ in the scope of $\mathbf{O}$ or $\mathbf{K}$. Formally, $\mathbf{O}$ is an epistemic modality whose semantics is given in the logic of *all I know* [11] by

- $M\models\mathbf{O}\varphi$ if $M=Mods(\varphi)$.

Hence any atom Φ of the form $\mathbf{O}\varphi$ (with φ satisfiable) has exactly one model, written $M(\Phi)=Mods(\varphi)\subseteq 2^X$. With this in mind, we use the term "knowledge state" to refer either to some $M\subseteq 2^X$ or to some atom $\mathbf{O}\varphi$. For instance, $M=\{x_1x_2,\bar{x}_1x_2,\bar{x}_1\bar{x}_2\}$ will also be denoted by $\Phi=\mathbf{O}(\neg x_1\vee x_2)$.

Satisfiability in the logic of *all I know* is Σ_2^{P}-complete [16]. However, we only need restricted entailment tests, the complexity of which we show to lie at the first level of the polynomial hierarchy only. We recall that $\Delta_2^{\mathsf{P}}=\mathsf{P}^{\mathsf{NP}}$ is the class of all decision problems that can be solved in deterministic polynomial time using NP-oracles, $\Pi_2^{\mathsf{p}}=\mathsf{coNP}^{\mathsf{NP}}$ the class of all decision problems whose complement can be solved in nondeterministic polynomial time using NP-oracles, and $\mathsf{EXPSPACE}$ the class of all decision problems that can be solved using exponential space.

Proposition 1 *Deciding $\mathbf{O}\varphi\models\Phi$, where φ is objective and Φ is purely epistemic (without any occurrence of $\mathbf{O}$), is in Δ_2^{P}.*

Proof Let φ and Φ be as in the claim. Hence Φ is a Boolean combination of atoms $\mathbf{K}\psi$. We give a polynomial time algorithm for deciding $\mathbf{O}\varphi\models\Phi$ with a linear number of calls to an oracle for propositional satisfiability, reasoning by induction on the structure of Φ.

First let $\Phi = \mathbf{K}\psi$. Then $\mathbf{O}\varphi \models \Phi$ reads $\forall M \subseteq 2^X, (M \models \mathbf{O}\varphi \Rightarrow \forall s \in M, s \models \psi)$. Since $\mathbf{O}\varphi$ has exactly one model $M = Mod(\varphi)$, this is equivalent to $\varphi \models \psi$, *i.e.*, $\varphi \wedge \neg\psi$ is not satisfiable.

Now let $\Phi = \Phi_1 \vee \Phi_2$ ($\wedge, \neg$ are similar). Then $\mathbf{O}\varphi$ entails Φ iff it entails Φ_1 or it entails Φ_2. Indeed, $\mathbf{O}\varphi$ has only one model $M(\mathbf{O}\varphi)$, hence $\mathbf{O}\varphi$ entails Φ iff $M(\mathbf{O}\varphi)$ satisfies Φ_1 or Φ_2, that is, iff $\mathbf{O}\varphi \models \Phi_1$ or $\mathbf{O}\varphi \models \Phi_2$ holds. Hence, deciding $\mathbf{O}\varphi \models \Phi$ involves a linear number of calls to the oracle by the induction hypothesis. □

3 KBPS AS PLANS

Our definitions specialize those in [4] to our propositional framework and to a single-agent version. Given a set A of primitive actions, a *knowledge-based program* (KBP) is defined inductively as follows:

- the empty plan is a KBP,
- any action $\alpha \in A$ is a KBP,
- if π and π' are KBPs, then $\pi; \pi'$ is a KBP;
- for KBPs π, π' and $\Phi \in \mathrm{SKNNF}$, **if** Φ **then** π **else** π' is a KBP;
- for a KBP π and $\Phi \in \mathrm{SKNNF}$, **while** Φ **do** π is a KBP.

The class of *while-free* KBPs is obtained by omitting the **while** construct. The *size* $|\pi|$ of a KBP π is defined to be the number of occurrences of actions, plus the size of branching conditions, in π.

3.1 Representation of Actions

Following [6], we assume without loss of generality that the set of actions is partitioned into *purely ontic* and *purely epistemic* actions.

An *ontic action* α modifies the current state of the environment but gives no feedback. Ontic actions may be nondeterministic. For the sake of simplicity we assume them to be fully executable[2].

Each ontic action is represented by a propositional theory expressing constraints on the transitions between the states of the environment before and after α is taken. Let $X' = \{x' \mid x \in X\}$, denoting the values of variables after the action was taken. The *theory* of α is a propositional formula Σ_α over $X \cup X'$ such that for all states $s \in 2^X$, the set $\{s' \in 2^{X'} \mid ss' \models \Sigma_\alpha\}$ is nonempty, and is exactly the set of possible states after α is performed in s. For instance, with $X = \{x_1, x_2\}$, the action α which nondeterministically reinitializes the value of x_1 has the theory $\Sigma_\alpha = (x'_2 \leftrightarrow x_2)$.

In the paper we will use the following actions:

- $\mathrm{reinit}(Y)$ (for some $Y \subseteq X$) with theory $\bigwedge_{x \notin Y} x' \leftrightarrow x$,
- $x_i := \varphi$ (for φ objective) with theory $x'_i \leftrightarrow \varphi \wedge \bigwedge_{j \neq i} x'_j \leftrightarrow x_j$,

[2]This, in theory, induces a loss of generality, but in practice if α is not executable in s, this can be expressed by letting α lead to a sink state incompatible with the goal.

- $\mathrm{switch}(x_i)$ with theory $x_i' \leftrightarrow \neg x_i \wedge \bigwedge_{j \neq i} x_j' \leftrightarrow x_j$,
- the void action λ with theory $\bigwedge_{x \in X} x' \leftrightarrow x$.

Now, an *epistemic action* has no effect on the current state, but gives some feedback about it, that is, it modifies only the knowledge state of the agent (typically, a sensing action). We represent such an action by the list of possible feedbacks. Formally, the *feedback theory* of α is a list of positive epistemic atoms, of the form $\Omega_\alpha = (\mathbf{K}\varphi_1, \ldots, \mathbf{K}\varphi_n)$. For instance, the epistemic action which senses the value of an objective formula φ is

- $\mathrm{test}(\varphi)$ with feedback theory $\Omega_{\mathrm{test}(\varphi)} = (\mathbf{K}\varphi, \mathbf{K}\neg\varphi)$.

Finally, for an objective formula φ over X, we write φ^t for the formula obtained from φ by replacing each occurrence of $x \in X$ with x^t. We also write $\Sigma_\alpha^{t \uparrow t+1}$ for the formula obtained from Σ_α by replacing each unprimed variable $x \in X$ with x^t, and each primed variable x' with x^{t+1}. For instance, for $\varphi_1 = (x_1 \vee x_2)$, φ_1^5 is $(x_1^5 \vee x_2^5)$, and for $\Sigma_\alpha = (x_2' \leftrightarrow x_2)$, $\Sigma_\alpha^{3 \uparrow 4}$ is $(x_2^4 \leftrightarrow x_2^3)$).

3.2 Semantics

The agent executing a KBP starts in some knowledge state M^0, and at any timestep t until the execution terminates, it has a current knowledge state M^t. When execution comes to a branching condition Φ, Φ is evaluated in the current knowledge state (the agent decides $M^t \models \Phi$).

The knowledge state M^t is defined inductively as the *progression* of M^{t-1} by the action executed between $t-1$ and t. Formally, given a knowledge state $M \subseteq 2^X$ and an *ontic* action α, the *progression* of M by α is defined to be the knowledge state $\mathrm{Prog}(M, \alpha) = M' \subseteq 2^{X'}$ defined by $M' = \{s' \in 2^{X'} \mid s \in M, ss' \models \Sigma_\alpha\}$. Intuitively, after taking α in a state which it knows to be one in M, the agent knows that the resulting state is one of those s' which are reachable from any $s \in M$ through α. Note that the agent knows that some outcome of the action has occurred (it knows Σ_α), but not which one.

Now given an *epistemic* action α, a knowledge state M, and a feedback $\mathbf{K}\varphi_i \in \Omega_\alpha$ with $M \not\models \mathbf{K}\neg\varphi_i$, the progression of M by $\mathbf{K}\varphi_i$ is defined to be $\mathrm{Prog}(M, \mathbf{K}\varphi_i) = M_i = \{s \in M \mid s \models \varphi_i\}$. The progression is undefined when $M \models \mathbf{K}\neg\varphi_i$. Intuitively, a state is considered to be possible after obtaining feedback φ_i iff it was considered to be possible before taking the epistemic action, and it is consistent with the feedback obtained. Here, observe that though an epistemic action can yield different feedbacks, at execution time the agent knows which one it gets.

Example 1 *(from [6]). Consider the following KBP*
π*:* $\mathrm{test}(x_1 \leftrightarrow x_2)$*;* **If** $\mathbf{K}(x_1 \leftrightarrow x_2)$ **then** $\mathrm{test}(x_1 \wedge x_2)$ **else** $(\mathrm{switch}(x_1); \mathrm{test}(x_1 \wedge x_2))$
With $M^0 = \mathbf{O}\top$ *(nothing known),* $\mathrm{Prog}(M^0, \mathbf{K}(\neg(x_1 \leftrightarrow x_2)))$ *is* $M^1 = \mathbf{O}(x_1 \leftrightarrow \neg x_2)$*,* $\mathrm{Prog}(M^1, \mathrm{switch}(x_1))$ *is* $M^2 = \mathbf{O}(x_1 \leftrightarrow x_2)$*, and* $\mathrm{Prog}(M^2, \mathbf{K}\neg(x_1 \wedge x_2))$ *is* $M^3 = \mathbf{O}(\neg x_1 \wedge \neg x_2)$.

We are now ready to give an operational semantics for KBPs. Given a knowledge state M' involving only primed variables of the form x' ($x \in X$), we write $\mathrm{plain}(M')$ for the knowledge state obtained by replacing x' with x for all $x \in X$.

An *execution trace* (or *trace*) τ of a KBP π in M^0 is a sequence of knowledge states, either infinite, *i.e.* $\tau = (M^i)^{i \geq 0}$, or finite, *i.e.* $\tau = (M^0, M^1, \ldots, M^T)$, and satisfying:

- if π is the empty plan, then $\tau = (M^0)$;
- if π is an ontic action α, then $\tau = (M^0, \mathrm{plain}(\mathrm{Prog}(M^0, \alpha)))$;
- if π is an epistemic action α, then $\tau = (M^0, \mathrm{Prog}(M^0, \mathbf{K}\varphi_i))$ for some $\mathbf{K}\varphi_i \in \Omega_\alpha$ with $M \not\models \mathbf{K}\neg\varphi_i$;
- for $\pi = \pi_1; \pi_2$, either $\tau = \tau_1$ with τ_1 an infinite trace of π_1, or $\tau = \tau_1\tau_2$ with τ_1 a finite trace of π_1 and τ_2 a trace of π_2;
- if π is **if** Φ **then** π_1 **else** π_2, then either $M^0 \models \Phi$ and τ is a trace of π_1, or $M^0 \not\models \Phi$ and τ is a trace of π_2;
- if π is **while** Φ **do** π_1, then either $M^0 \models \Phi$ and τ is a trace of $\pi_1; \pi$, or $M^0 \not\models \Phi$ and $\tau = (M^0)$.

We say that π *terminates* in M^0 if every trace of π in M^0 is finite.

Example 2 *Let $\pi, M^0, \ldots, M^3$ as in Ex. 1, and $M^4 = \mathbf{O}(x_1 \wedge x_2)$. The traces of π in M^0 (with the corresponding feedbacks) are:*

(M^0, M^1, M^2, M^3)	$\mathbf{K}\neg(x_1 \leftrightarrow x_2), \mathbf{K}\neg(x_1 \wedge x_2)$
(M^0, M^1, M^2, M^4)	$\mathbf{K}\neg(x_1 \leftrightarrow x_2), \mathbf{K}(x_1 \wedge x_2)$
(M^0, M^2, M^3)	$\mathbf{K}(x_1 \leftrightarrow x_2), \mathbf{K}\neg(x_1 \wedge x_2)$
(M^0, M^2, M^4)	$\mathbf{K}(x_1 \leftrightarrow x_2), \mathbf{K}(x_1 \wedge x_2)$

4 KBPS VS. STANDARD POLICIES

We now briefly compare KBPs with standard policies (or plans) with respect to succinctness and expressiveness[3]. As opposed to a KBP, define a *standard* policy to be a program with *objective* branching conditions. This encompasses plans for classical planning, which are simply sequences of actions $a_1; a_2; \ldots; a_n$, but also POMDP policies, which branch on observations, and other types of policies, such as controllers with finite memory [2].

Clearly, every KBP π can be translated into an equivalent standard policy (a "protocol" in [4]), by simulating all possible executions of π and, for all possible executions of the program, evaluating all (epistemic) branching conditions. *Vice versa*, it is clear that any standard policy can be translated to an equivalent KBP.

Such translations are of course not guaranteed to be polynomial. In particular, it is easily seen that a standard policy π described in space $O(n)$ can manipulate at most n variables (through actions or branching conditions). It follows that it can be in at most $|\pi|2^n$ different configurations (value of each variable plus control point in the policy), hence any finite trace of such a policy can have length at most $|\pi|2^n$ (being twice in the

[3]For space reasons, our discussion is informal. Proofs and details are omitted.

same configuration would imply an infinite loop). In contrast, we will give in Section 6 a KBP described in space polynomial in n but with a finite trace of length 2^{2^n}.

However, what is gained on succinctness is lost on the complexity of execution. When executing a KBP, the problem of evaluating a branching condition is in Δ_2^{P}, but is both NP- and coNP-hard. Indeed, it is coNP-hard because $\mathbf{O}\top \models \mathbf{K}\varphi$ corresponds to φ being valid, and NP-hard because $\mathbf{O}\top \models \neg\mathbf{K}\neg\varphi$ corresponds to φ being satisfiable. On the other hand, when executing a standard policy, evaluating an (objective) condition can be done in linear time by reading the values of the (internal and environment) variables involved.

Interestingly, even the restriction to while-free KBPs does not imply a loss of expressivity. Indeed, if a loop terminates, then it is guaranted to be executed less than 2^{2^n} times (see Section 6), and hence it can be unrolled, yielding an equivalent while-free KBP. However, this obviously comes also with a loss of succinctness.

When KBPs are seen as *plans* which achieve *goals*, as we consider in this article, the translations outlined above preserve the property that the KBP/policy indeed achieves the goal. Therefore, from the point of view of *plan existence*, considering KBPs or standard plans makes no difference: there is a plan for a given problem if and only if there is a KBP for it. Moreover, since the input is the same in both cases, the complexity of plan existence is independent of whether we look for policies of KBPs. Things are different for the problem of *verifying* that a KBP/policy is a plan for some goal, because the KBP or policy is part of the input. For example, we will see in Section 6 that verifying while-free KBPs is Π_2^{P}-complete. In contrast, it can easily be shown that verifying a while-free policy is in coNP (with essentially the same proof as Proposition 2).

5 VERIFYING WHILE-FREE KBPS

We now investigate the computational problem of *verifying* that a KBP π is valid for a planning problem. Precisely, we define a *knowledge-base planning problem* P to be a tuple (Φ^0, A_O, A_E, G), where $\Phi^0 = \mathbf{O}\varphi^0$ is the *initial knowledge state*, G is a SKNNF S_5 formula called the *goal*, and A_O (resp. A_E) is a set of ontic (resp. epistemic) actions together with their theories. Then a KBP π (using actions in $A_O \cup A_E$) is said to be a *(valid) plan* for P if its execution in Φ^0 terminates, and for all traces $(M^0, \dots, M^T)$ of π with $M^0 = M(\Phi^0)$, $M^T \models G$ holds. Intuitively, this means that executing π always leads to a knowledge state where the agent is sure that G holds. For instance, in Example 1, π is a plan for $\Phi^0 = \mathbf{O}\top$ and the goal $G = (\mathbf{K}x_1 \vee \mathbf{K}\neg x_1) \wedge (\mathbf{K}x_2 \vee \mathbf{K}\neg x_2)$.

Definition 1 (verification) *The* plan verification problem *takes as input a knowledge-based planning problem $P = (\Phi^0, A_O, A_E, G)$ and a KBP π, and asks whether π is a plan for P.*

In this section we show that verification is Π_2^{P}-complete for while-free KBPs, even under several further restrictions. Observe first that a while-free KBP always terminates.

We start with membership in Π_2^{P}. In the broad lines, the argument is that π is *not* a plan for P if there exists a trace τ of π (or, equivalently, a sequence of feedbacks for the epistemic actions executed) in which the last knowledge state does *not* satisfy G. Hence π can be verified *not* to be a plan for P by guessing such a sequence of feedbacks and simulating the corresponding execution.

Nevertheless, we must perform such simulation in polynomial space. Unfortunately, in general the progression of a knowledge state M represented as $\mathbf{O}\varphi$ cannot be performed in polynomial space.

Example 3 *The progression of* $\mathbf{O}\varphi$*, for* $\varphi = \bigwedge_{i=1}^{n}(x_i \vee y_i \to z_i) \wedge ((\bigwedge_{i=1}^{n} z_i) \to z)$*, by* $\mathrm{reinit}(\{z_1, \ldots, z_n\})$*, is equivalent to* $\mathbf{O}(\exists z_1, \ldots, z_n, \varphi) \equiv \bigwedge_{\ell_1 \in \{x_1, y_1\}, \ldots, \ell_n \in \{x_n, y_n\}}(\ell_1 \wedge \cdots \wedge \ell_n \to z)$*, which has no polynomial representation, while* $|\varphi|$ *is linear.*

Hence we introduce another form of progression, called *memoryful progression*, which explicitly keeps track of successive knowledge states instead of projecting to the current instant. Namely, we define a *memoryful knowledge state* for a timestep t to be a formula of the form $\mathbf{O}\varphi^{\uparrow t}$, where $\varphi^{\uparrow t}$ is an objective formula over the set of variables $\bigcup_{i=0}^{t}\{x^i \mid x \in X\}$. Intuitively, $\mathbf{O}\varphi^{\uparrow t}$ represents the past and present knowledge of the agent at timestep t. Formally:

- for ontic α, $\mathrm{MemProg}(\mathbf{O}\varphi^{\uparrow t}, \alpha) = \mathbf{O}(\varphi^{\uparrow t} \wedge \Sigma_{\alpha}^{t \uparrow t+1})$,
- for a feedback $\mathbf{K}\varphi_i$, $\mathrm{MemProg}(\mathbf{O}\varphi^{\uparrow t}, \mathbf{K}\varphi_i) = \mathbf{O}(\varphi^{\uparrow t} \wedge \varphi_i^t)$.

Observe that ontic actions increment the current timestep, while epistemic actions do not (they do not modify the current state).

Example 4 (Example 1, continued) *The memoryful progression of* $\mathbf{O}\top$ *by* $\mathbf{K}\neg(x_1 \leftrightarrow x_2)$*, then* $\mathrm{switch}(x_1)$*, then* $\mathbf{K}\neg(x_1 \wedge x_2)$ *is*

$$\mathbf{O}\left(\top^0 \wedge (x_1^0 \leftrightarrow \neg x_2^0) \wedge (x_1^1 \leftrightarrow \neg x_1^0) \wedge (x_2^1 \leftrightarrow x_2^0) \wedge \neg(x_1^1 \wedge x_2^1)\right)$$

Clearly, the memoryful progression of $\mathbf{O}\varphi^{\uparrow t}$ by α (resp. $\mathbf{K}\varphi_i$) has a size linear in $|\varphi^{\uparrow t}|$ and $|\Sigma_\alpha|$ (resp. $|\mathbf{K}\varphi_i|$). Hence iterating the memoryful progression of Φ^0 a polynomial number of times $poly(|\pi|)$ yields a (memoryful) knowledge state of polynomial size.

Lemma 1 *Let* $\mathbf{O}\varphi^0$ *be an initial knowledge state, and let* $(\ell_1, \ldots, \ell_T)$ *be a sequence containing* U *ontic actions and* $T - U$ *feedbacks. Then the iterated progression* M *of* M^0 *by* $\ell_1, \ldots, \ell_T$ *satisfies an epistemic formula* Φ *if and only if the iterated memoryful progression* $\mathbf{O}\varphi^{\uparrow U}$ *of* $\mathbf{O}\varphi^0$ *by* $\ell_1, \ldots, \ell_T$ *entails* Φ^U*.*

Proof Sketch Renaming variables, $M \models \Phi$ is equivalent to $M^U \models \Phi^U$. On the other hand, it is easily seen from the definitions that M^U is exactly $Mod(\exists X^0, \ldots, \exists X^{U-1}\varphi^{\uparrow U})$. Because Φ^U contains only variables in X^U, it follows that $M^U \models \Phi^U$ is equivalent to $Mod(\varphi^{\uparrow U}) \models \Phi^U$ ([8], Corollary 7), *i.e.*, to $\mathbf{O}\varphi^{\uparrow U} \models \Phi^U$. □

Recall that a problem is in Π_2^{p} if its complement can be solved by a polytime nondeterministic algorithm which uses an NP-oracle.

Proposition 2 *Plan verification is in* Π_2^P *for while-free KBPs.*

Algorithm 1: Deciding whether a while-free π is not valid

begin
- $t := 0$, $\mathbf{O}\varphi^{\uparrow 0} = \Phi^0$;
- **while** π *is not the empty KBP* **do**
  - **if** $\pi = \alpha; \pi'$ *and* α *is ontic* **then**
    - $\mathbf{O}\varphi^{\uparrow t+1} := MemProg(\mathbf{O}\varphi^{\uparrow t}, \alpha)$;
    - $\pi := \pi'$, $t := t + 1$;
  - **else if** $\pi = \alpha; \pi'$ *and* α *is epistemic* **then**
    - guess a feedback $\mathbf{K}\varphi_i$ in Ω_α;
    - check that $\varphi^{\uparrow t} \wedge \varphi_i$ is satisfiable;
    - $\mathbf{O}\varphi^{\uparrow t} := MemProg(\mathbf{O}\varphi^{\uparrow t}, \mathbf{K}\varphi_i)$;
  - **else** $\{\pi$ is of the form **if** Φ **then** π_1 **else** $\pi_2; \pi'\}$
    - **if** $\mathbf{O}\varphi^{\uparrow t} \models \Phi^t$ **then** $\pi := \pi_1; \pi'$;
    - **else** $\pi := \pi_2; \pi'$;
- check $\mathbf{O}\varphi^{\uparrow t} \not\models G^t$;

end

Proof We use Algorithm 1, which decides whether π is not valid in nondeterministic polynomial time with access to an oracle for propositional satisfiability. Intuitively, the algorithm simulates an execution of π, guessing a sequence of feedbacks witnessing that π is not valid. Clearly, it runs in nondeterministic polynomial time, and it uses a polynomial number of calls to the oracle: one per check that $\varphi^{\uparrow t} \wedge \varphi_i$ is satisfiable, a linear number per check $\mathbf{O}\varphi^{\uparrow t} \models \Phi^t$ (Proposition 1), and a linear number for the final check. $\square$

Proposition 3 *Plan verification is Π_2^P-hard. Hardness holds even if the KBPs π are restricted to be while-free and either to $A_O = \emptyset$ (no ontic action), or to $A_E = \emptyset$ (no epistemic action).*

Proof We give two reductions from the (Π_2^{P}-complete) problem of deciding the validity of a QBF of the form $\forall x_1 \ldots x_p \exists y_1 \ldots y_q \varphi$, where φ is a propositional formula over $\{x_1, \ldots, x_p\} \cup \{y_1, \ldots, y_q\}$. In both cases we build a planning problem $P = (\Phi^0, A_O, A_E, G)$ and a KBP π, with $\Phi^0 = \mathbf{O}\top$ and $G = \neg\mathbf{K}\neg\varphi$.

Given only epistemic actions, we let $\pi = test(x_1); \ldots; test(x_p)$. Then π is not a valid plan if and only if there is a sequence of feedbacks for $test(x_1), \ldots, test(x_p)$ such that $\mathbf{K}\neg\varphi$ holds, i.e., the agent knows that φ is false. This is equivalent to there being values for $x_1, \ldots, x_p$ such that whatever the value of $y_1, \ldots, y_q$, φ is false, that is, to $\forall x_1 \ldots x_p \exists y_1 \ldots y_q \varphi$ not being valid.

Similarly, given only ontic actions we let $\pi = \mathrm{reinit}(\{x_1, \ldots, x_p\})$. Again, π is not a valid plan if and only if there is an outcome for α, that is, values for $x_1, \ldots, x_p$, such that $\mathbf{K}\neg\varphi$ holds, i.e., $\forall x_1 \ldots x_p \exists y_1 \ldots y_q \varphi$ is not valid. $\square$

6 VERIFYING KBPS WITH LOOPS

For general KBPs, we now show verification to be EXPSPACE-complete (EXPSPACE is the class of decision problems with an exponential space algorithm). On the way, we build a polysize KBP with a doubly exponentially long trace, which we use as a clock. Since the construction is of independent interest, we present it first.

6.1 A Very Slow KBP

We write $>$ for the lexicographic order on states. For instance, 2^X is ordered by $x_1x_2x_3 > x_1x_2\bar{x}_3 > x_1\bar{x}_2x_3 > \cdots > \bar{x}_1\bar{x}_2\bar{x}_3$ for $n = 3$ variables. Given X and a knowledge state M over a superset of X, we write M_X for $\{s_X \mid s \in M\}$, where s_X denotes the restriction of s to the variables in X. This allows us to use auxiliary variables and still talk about the knowledge state about X.

We build a compact KBP (of size polynomial in n) with exactly one trace, of size $2^{2^n} - 1$. As discussed in Section 4, this is impossible with standard policies, but possible for KBPs because their configurations include a *knowledge* state, and there are $2^{2^n} - 1$ of them (every nonempty subset of 2^X). Hence there can be a program π which passes through 2^{2^n} different configurations while being specified with only $O(n)$ variables and in space $|\pi|$ polynomial in n.

Routines and Actions We build our KBP so that its execution passes through each possible knowledge state exactly once. To do so, we need some specific actions and routines which allow to go from a knowledge state to the next one.

The first routine determines the state s in M with the greatest restriction M_X (wrt $>$), and stores it over some auxiliary variables $g_1, \ldots, g_n$. For instance, if the current knowledge state satisfies $M_X = \{\bar{x}_1\bar{x}_2x_3, \bar{x}_1x_2x_3, x_1\bar{x}_2\bar{x}_3\}$, then after executing the routine, the agent knows $g_1 \wedge \neg g_2 \wedge \neg g_3$ (and M_X is unchanged).

We define π_g^n to perform a dichotomic search in M. For instance, if $K((x_1 \leftrightarrow g_1) \rightarrow \neg x_2)$ is true, then no assignment in M which satisfies $x_1 \leftrightarrow g_1$ (*i.e.*, by construction, none of the assignments with greatest x_1) satisfies x_2, hence the greatest one satisfies $\neg x_2$. Precisely, π_g^n is the following KBP:

If $\mathbf{K}(\neg x_1)$ **then** $g_1 := 0$ **else** $g_1 := 1$;
If $\mathbf{K}((x_1 \leftrightarrow g_1) \rightarrow \neg x_2)$ **then** $g_2 := 0$ **else** $g_2 := 1$;
...
If $\mathbf{K}(\bigwedge_{i=1}^{n-1} x_i \leftrightarrow g_i) \rightarrow \neg x_n)$ **then** $g_n := 0$ **else** $g_n := 1$;

We now introduce an ontic action which adds a given state $s_a \in 2^X$ to M_X. We assume s_a is encoded over some auxiliary variables $a_1, \ldots, a_n$. The action a^n_{add} is a simple nondeterministic one, which either does nothing or sets $x_1, \ldots, x_n$ to the values of $a_1, \ldots, a_n$. Formally, its action theory is $(\bigwedge_{i=1}^n x'_i \leftrightarrow x_i) \vee (\bigwedge_{i=1}^n x'_i \leftrightarrow a_i)$ (and no effect on auxiliary variables). Hence after taking this action, the agent exactly knows that either the environment is in the same state as before, or it is in the state $a_1 \ldots a_n$.

We finally introduce a routine π_r^n, which removes a given state $s_r \in 2^X$ (encoded over auxiliary variables $r_1, \ldots, r_n$) from M_X. We assume that the agent knows the state to be removed, that is, M satisfies $\mathbf{K}r_i \vee \mathbf{K}\neg r_i$ for all $i = 1, \ldots, n$.

Recall that by definition, knowledge states are nonempty. Hence we allow removal of s_r only if there is another state $s_g \in M_X$, ensuring $M_X \setminus \{s_r\} \neq \emptyset$. Then π_r^n removes s_r from M_X by identifying a distinguished state $s_g \neq s_r$ in M_X, then executing an action a_r^n which maps any state to itself except for s_r, which it maps to s_g.

We identify s_g by running π_g^n. If it turns out that s_g is precisely s_r, as can be decided since the agent knows (i) the value of s_r by assumption, and (ii) that of s_g by construction of π_g^n, then π_r^n replaces s_g with the least assignment in M_X, using the dual of π_g^n. Finally, a_r^n is defined to be the deterministic ontic action with theory $\bigwedge_{i=1}^n x_i' \leftrightarrow (x_i \oplus ((\bigwedge_{i=1}^n x_i \leftrightarrow r_i) \wedge (x_i \oplus g_i)))$ (and no effect on auxiliary variables). A case analysis shows that a_r^n maps s to itself except for s_r which it maps to s_g, as desired.

Proposition 4 *Let π_g^n run in knowledge state M. Then the progressed knowledge state M' satisfies $\bigwedge_{i=1}^n \mathbf{K}g_i^{\epsilon_i}$, where $g_i^{\epsilon_i}$ is g_i (resp. $\neg g_i$) if the greatest state in M_X satisfies x_i (resp. $\neg x_i$). Now if a_add^n (resp. π_r^n) is run, then $M_X' = M_X \cup \{s_a\}$ (resp. $M_X' = M_X \setminus \{s_g\}$) holds.*

Importantly, observe that $\pi_g^n, a_\mathrm{add}^n, \pi_r^n$ all have a description of size at most quadratic in n. Finally, we use a routine, written π_d ("decrement"), which replaces the state encoded by $g_1, \ldots, g_n$ by its predecessor wrt $>$ (the definition of π_d is straightforward, and omitted for space reasons).

A Slow KBP We see a knowledge state M as a vector $\vec{m} = m_1 m_2 \ldots m_{2^n-1} m_{2^n}$, with $m_i = 1$ if and only if the ith state s_i (wrt $>$) is in M. Then our KBP starts with $\vec{m}^0 = 00\ldots01$ (*i.e.*, $M^0 = \{11\ldots1\} \equiv \mathbf{K}x_1 \wedge \cdots \wedge x_n$), and loops until $\vec{m}^t = 10\ldots00$ ($M^t = \{00\ldots0\} \equiv \mathbf{K}\neg x_1 \wedge \cdots \wedge \neg x_n$). The loop changes the current $\vec{m}^t$ to $\vec{m}^{t+1}$ using the *Gray code*, which is a way to enumerate all Boolean vectors by changing exactly one bit at a time.

Definition 2 (Gray Code) *The successor of $\vec{m}$ according to the* Gray Code *is obtained from $\vec{m}$ as follows:*

1. *if $\vec{m}$ has an even number of 1's, flip m_{2^n},*
2. *otherwise, let $g = \max\{i \mid m_i = 1\}$ and flip m_{g-1}.*

For instance, the enumeration is $0001, 0011, 0010, 0110 \ldots 1000$ for $n = 2$ (we do not use 0000). In terms of knowledge states, this is $\{x_1 x_2\}$,, $\{x_1 \bar{x}_2, x_1 x_2\}$, $\{x_1 \bar{x}_2\}$, $\{\bar{x}_1 x_2, x_1 \bar{x}_2\} \ldots \{\bar{x}_1 \bar{x}_2\}$, which indeed passes through all knowledge states.

By definition of $\vec{m}^t$, the greatest i with $m_i^t = 1$ identifies the greatest state in M^t, and flipping m_i^t amounts to add/remove s_i to M^t. With this in hand, our KBP $clock^n$ (Figure 2) uses a set of n variables X and auxiliary variables $g_1, \ldots, g_n$, $a_1, \ldots, a_n$, $r_1, \ldots, r_n$.

Proposition 5 *The unique trace for $clock^n$ in M^0 has size $2^{2^n} - 1$.*

Algorithm 2: The KBP $clock^n$

```
begin
    odd ← 1 {number of 1's in knowledge state M^0};
    while ¬K(¬x_1 ∧ ··· ∧ ¬x_n) do
        if K¬o then {even number of 1's, flip m_{2^n}}
            if K(¬x_1 ∨ ··· ∨ ¬x_n) then
                {11...1 ∉ M^t, add it} a_1 := 1; a_2 := 1; ...; a_n := 1; a^n_add;
            else r_1 := 1; r_2 := 1; ...; r_n := 1; π^n_r
        else {odd number of 1's, flip m_{g-1}}
            π_g; π_d;
            if K(x_1 ↮ g_1 ∨ ··· ∨ x_n ↮ g_n) then
                {s_{g-1} ∉ M^t, add it} a_1 := g_1; a_2 := g_2; ...; a_n := g_n; a^n_add;
            else r_1 := g_1; r_2 := g_2; ...; r_n := g_n; π^n_r;
        odd := ¬odd;
end
```

6.2 EXSPACE-hardness

We now show that verifing general KBPs is EXPSPACE-complete. We prove harness with a reduction from nondeterministic unobservable planning (NUP) [5]. An instance of NUP is a triple $(\varphi^0, A_{\text{HJ}}, \varphi_G)$ where φ^0, φ_G are propositional formulas and A_{HJ} is a set of ontic, nondeterministic actions (see below). The question is whether there is a plan, i.e., a sequence of actions, which reaches a state satisfying φ_G from any state satisfying φ^0 (in our terms, whose traces in $\mathbf{O}\varphi^0$ all end in a knowledge state satisfying $\mathbf{K}\varphi_G$).

The actions considerer by Haslum and Jonsson (*HJ-actions* for short) are different from ours. They are defined inductively as follows (we adapt their notation for consistency):

- $x_i := 0$ and $x_i := 1$ are HJ-actions for any $x_i \in X$,
- if a_1, a_2 are HJ-actions, then $a_1; a_2$ is an HJ-action,
- if φ is a propositional formula and a_1, a_2 are HJ-actions, then **if** φ **then** a_1 **else** a_2 is an HJ-action[4],
- if a_1, a_2 are HJ-actions, then $a_1 | a_2$ is an HJ-action.

The semantics of executing such an action is the same as ours for the three first constructs, given that φ_0 defines the initial knowledge state $\mathbf{O}\varphi^0$. As for nondeterminism, the progression of M^t by $a_1|a_2$ is simply defined to be $Prog(M^t, a_1) \cup Prog(M^t, a_2)$ (at execution time exactly one of a_1, a_2 occurs, but we do not know which one).

The idea of our reduction is to build a KBP, written $simulate$, which explores all possible plans (up to size 2^{2^n}, see below) for an NUP problem, and which is valid if and

[4]In [5] this operator is n-ary, but as the conditions are mutually inconsistent, their $\varphi_1?a_1 : \ldots : \varphi_k?a_k$ can be rewritten as **if** φ_1 **then** a_1 **else** (**if** φ_2 **then** a_2 **else** $(\ldots)$), which has the same size.

only if none achieves the goal. For this, we first associate a routine (KBP) $\pi(a)$ to any HJ-action a, so as to be able to use a in $simulate$.Indeed, conditional HJ-actions are not allowed in KBPs because they branch on objective formulas, and nondeterministic HJ-actions are not directly allowed.

For a of the form **if** φ **then** a_1 **else** a_2, we define $\pi(a)$ by "pushing in" the objective test to the assignments. Precisely, we define $\pi(a)$ to be the $c := \varphi; \pi_c(a_1); \pi_{\neg c}(a_2)$ with π_c defined inductively by:

- $\pi_c(x := \psi) = (x := (x \oplus (c \wedge (x \oplus \psi))))$,
- $\pi_c(a_1; a_2) = (\pi_c(a_1); \pi_c(a_2))$,
- $\pi_c($**if** φ **then** a_1 **else** $a_2) = (c' := c \wedge \varphi; \pi_{c'}(a_1); \pi_{\neg c'}(a_2))$,
- $\pi_c(a_1 | a_2) = (\pi_c(a_1) | \pi_c(a_2))$.

Intuitively, executing a or $\pi_1(a)$ in M^t leads to the same knowledge state M^{t+1}, while $\pi_0(a)$ leaves M^t unchanged (the construction of $\pi_c(x := \psi)$ is the same as for the action a_{r}^n in Section 6.1). Interestingly, this construction shows that the restriction to purely subjective branching conditions in KBPs is without loss of generality.

Finally, for nondeterminism we use the action $\mathrm{reinit}(h)$, where h is an auxiliary variable, for simulating a coin flip (h stands for "heads"), and we define $\pi(a_1|a_2)$ to be $(\mathrm{reinit}(h); \pi_h(a_1); \pi_{\neg h}(a_2))$.

Lemma 2 *For any HJ-action a, the KBP $\pi(a)$ can be built efficiently, and for any M, the progressions M'_{HJ} and M' of M by a (resp. $\pi(a)$) satisfy $(M'_{\mathrm{HJ}})_X = M'_X$ (ignoring auxiliary variables).*

Proposition 6 *The verification problem for KBPs is EXSPACE-hard. Hardness holds even if only one while-loop is allowed and the KBPs to be verified are known to terminate.*

Proof We use both results that deciding whether an NUP instance has a plan is EXPSPACE-complete, and that an instance has a plan if and only if it has one of size at most 2^{2^n} [5].

Given an instance $(\varphi^0, A_{\mathrm{HJ}}, \varphi_G)$ of NUP, we build the knowledge-based planning problem $(\mathbf{O}\varphi^0, A_O, A_E, \neg\mathbf{K}\varphi_G)$ and the KBP $simulate$. This KBP uses the set of n variables X of the NUP instance, together with auxiliary sets of variables of size n for use by $clock^n$ (using disjoint sets of variables makes $clock^n$ run in parallel of the simulation itself). Then it loops over a guess of an action in $A_{\mathrm{HJ}} = \{a_1, \ldots, a_k\}$: this is done by flipping k coins (using $\mathrm{reinit}(\{h_1, \ldots, h_k\})$) and executing the first action whose coin turned heads (determined by taking the epistemic actions $\mathrm{test}(h_i)$)[5].

The KBP $simulate$ is depicted on Figure 3. We let $clock^{n+}$ be a KBP which counts up to 2^{2^n} (obtained, say, by adding a dummy action to $clock^n$). Clearly, $simulate$ can be built in polynomial time. Let p be a plan of length at most 2^{2^n} for the NUP instance. Then by definition, the trace of $simulate$ in which precisely the actions in p are chosen by $\mathrm{reinit}(\{h_1, \ldots, h_k\})$ ends up with $\mathbf{K}\varphi_G$ being true, i.e., the goal $\neg\mathbf{K}\varphi_G$

[5] Clearly, $\log k$ coins would be enough, but we keep the presentation simple.

Algorithm 3: The KBP $simulate$

begin
  initialize $clock^{n+}$;
  $\{$Loop until NUP goal reached for sure or clock beeps$\}$;
  while $\neg\mathbf{K}\varphi_G \wedge \neg\mathbf{K}(\neg x_1 \wedge \cdots \wedge \neg x_n)$ **do**
    run one step of $clock^{n+}$;
    $\mathrm{reinit}(\{h_1, \ldots, h_k\})$; $\mathrm{test}(h_1)$; $\ldots$; $\mathrm{test}(h_k)$;
    if $\mathbf{K}h_1$ **then** $\pi(a_1)$;
    else if $\mathbf{K}h_2$ **then** $\pi(a_2)$;
    $\ldots$;
    else if $\mathbf{K}h_k$ **then** $\pi(a_k)$;
end

being false. Hence $simulate$ is not valid. Conversely, only a choice of actions which achieve $\mathbf{K}\varphi_G$ can witness that $simulate$ is not valid, hence if $simulate$ is not valid then there is a plan for the NUP instance. Hence NUP reduces to the complement of KBP verification, hence the latter is coEXPSPACE-hard, that is, EXPSPACE-hard. $\Box$

Proposition 7 *The verification problem for KBPs is in* EXPSPACE.

Proof The proof mimicks Proposition 2 and Algorithm 1. Because a **while** loop being executed more than 2^{2^n} times would necessarily start at least twice in the same knowledge state and hence run forever, such loops are unrolled 2^{2^n} times. These can be counted over 2^n bits, hence in exponential space. As for the current knowledge state, instead of using memoryful progression, we maintain M^t in extension (as its explicit list of states), again in exponential space. Hence the problem is in coNEXPSPACE, hence in coEXPSPACE=EXPSPACE by Savitch's theorem. $\Box$

7 CONCLUSION

We investigated the use of knowledge-based programs as compact representations of policies or plans. It turns out that they are an interesting representation, dual to standard once in the sense that they can be exponentially more compact, but execution is computationally more difficult, as well as verification.

This work is a first step towards bridging knowledge-based programming and AI planning. Plan generation (that is, synthesis of KBPs) is the next issue to be addressed.

Acknowledgements

This research has been supported by the ANR-10-BLAN-0215 project *Learning And Reasoning for Deciding Optimally using Numerical and Symbolic information* (LARDONS).

References

[1] C. Bäckström and P. Jonsson. Limits for compact representations of plans. In *Proc. ICAPS 2011*, pages 146–153, 2011.

[2] B. Bonet, H. Palacios, and H. Geffner. Automatic derivation of finite-state machines for behavior control. In *Proc. AAAI-10*, 2010.

[3] J. Claßen and G. Lakemeyer. Foundations for knowledge-based programs using es. In *KR*, pages 318–318, 2006.

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

[5] P. Haslum and P. Jonsson. Some results on the complexity of planning with incomplete information. In *Proc. 5th European Conference on Planning (ECP 1999)*, pages 308–318, 1999.

[6] A. Herzig, J. Lang, and P. Marquis. Action representation and partially observable planning in epistemic logic. In *Proceedings of IJCAI03*, pages 1067–1072, 2003.

[7] J.Halpern and Y. Moses. Characterizing solution concepts in games using knowledge-based programs. In *Proceedings of IJCAI-07*, 2007.

[8] J. Lang, P. Liberatore, and P. Marquis. Propositional independence: Formula-variable independence and forgetting. *J. Artif. Intell. Res. (JAIR)*, 18:391–443, 2003.

[9] N. Laverny and J. Lang. From knowledge-based programs to graded belief-based programs part i: On-line reasoning. *Synthese*, 147(2):277–321, 2005.

[10] N. Laverny and J. Lang. From knowledge-based programs to graded belief-based programs, part ii: off-line reasoning. In *IJCAI*, pages 497–502, 2005.

[11] H. J. Levesque. All i know: a study in autoepistemic logic. *Artificial Intelligence*, 42(2–3):263–309, March 1990.

[12] M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Complexity of finite-horizon markov decision process problems. *J. ACM*, 47(4):681–720, 2000.

[13] R. Reiter. On knowledge-based programming with sensing in the situation calculus. *ACM Trans. Comput. Log.*, 2(4):433–457, 2001.

[14] J.Y. Halpern R.I. Brafman and Y. Shoham. On the knowledge requirements of tasks. *Journal of Artificial Intelligence*, 98(1–2):317–350, 1998.

[15] J. Rintanen. Complexity of planning with partial observability. In *ICAPS*, pages 345–354, 2004.

[16] R. Rosati. On the decidability and complexity of reasoning about only knowing. *Artificial Intelligence*, 116:193–215, 2000.