Knowledge-Based Programs as Plans - The Complexity of Plan Verification
Jérôme Lang, Bruno Zanuttini

To cite this version:
Knowledge-Based Programs as Plans
– The Complexity of Plan Verification –

Jérôme Lang\(^1\) and Bruno Zanuttini\(^2\)

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 online 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 \(K\varphi\) then \(\pi\) else \(\pi'\), where \(K\) is an epistemic modality and \(\pi, \pi'\) 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]).

KBPs have surprisingly been underlooked in the perspective of planning. Initially developed for distributed computing, they have been considered in AI for agent design [13] and game theory [7]. For planning, the only works we know of are by Reiter [12], 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. 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 \(\Pi^p_2\) completeness for while-free KBPs (Section 5) and \(\text{EXPSPACE}\)-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, \(x_1 = x_2 = \text{false}\) 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, \(M = \{x_1 \bar{x}_2, \bar{x}_1 x_2\}\) 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 \(K\). An \(S_5\) formula is objective if it does not contain any occurrence of \(K\). Objective formulas are denoted by \(\varphi, \psi\), etc. whereas general \(S_5\) formulas are denoted by \(\Phi, \Psi\) etc. For an objective formula \(\varphi\), we denote by \(\text{Mod}(\varphi)\) the set of all states which satisfy \(\varphi\) (i.e., \(\text{Mod}(\varphi) = \{s \in 2^X, s \models \varphi\}\)). The size \(|\Phi|\) of an \(S_5\) formula \(\Phi\) is the total number of occurrences of propositional symbols, connectives and modality \(K\) in \(\Phi\).

It is well-known (see, e.g., [4]) that any \(S_5\) formula is equivalent to a formula without nested \(K\) modalities; therefore we disallow them. An \(S_5\) formula \(\Phi\) is purely subjective if objective formulas occur only in the scope of \(K\). 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 \(K\)) or directly before a \(K\) modality. Any purely subjective \(S_5\) formula \(\Phi\) can be rewritten into an equivalently KNNF of polynomial size, by pushing all occurrences of \(\neg\) that are out of the scope of \(K\) as far as possible with de Morgan’s laws. For instance, \(K\neg(p \land q) \lor \neg(Kr \lor K\neg r)\) is not in KNNF, but is equivalent to \(K\neg(p \land q) \lor (\neg Kr \lor \neg(K\neg r))\). Summarizing, a subjective \(S_5\) formula \(\Phi\) in KNNF (for short, \(\Phi \in \text{SKNNF}\)) is either a positive (resp. negative) epistemic atom \(K\varphi\) (resp. \(\neg K\varphi\)), where \(\varphi\) is objective, or a combination of such atoms using \(\land, \lor\).

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

- \(M \models K\varphi\) if for all \(s' \in M, s' \models \varphi\),
- \(M \models K\varphi\) if \(M \models \varphi\),
- \(M \models \Phi \lor \Psi\) (resp. \(\Phi \lor \Psi\)) if \(M \models \Phi\) and (resp. 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 \(\Phi, \Psi\) entail \(\Psi\), written \(\Phi \models \Psi\), if every knowledge state \(M \subseteq 2^X\) which satisfies \(\Phi\) also satisfies \(\Psi\), and \(\Phi\) is equivalent to \(\Psi\) if \(\Phi\) and \(\Psi\) entail each other. Note that \(K\varphi \land K\psi\) is equivalent to

\(^1\) CNRS and LAMSADE, Université Paris-Dauphine, CNRS UMR 7243, email: lang@lamsade.dauphine.fr
\(^2\) GREYC, Université de Caen Basse-Normandie, CNRS UMR 6072, ENSI-CAEN, email: bruno.zanuttini@unicen.fr
Mod refer either to some Proposition 1 that can be solved using exponential space.ing complement can be solved in nondeterministic polynomial time us-
oracles, and problems that can be solved in deterministic polynomial time using

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into

Following [6], we assume without loss of generality that the set of

knowledge-based program

actions is partitioned into
Example 1 (from [6]). Consider the following KBP $\pi$:
\[
\text{test}(x_1 \leftrightarrow x_2); \\
\text{If } K(x_1 \leftrightarrow x_2) \text{ then } \text{test}(x_1 \land x_2) \text{ else } \text{(switch}(x_1); \text{ test}(x_1 \land x_2))
\]

With $M^0 = O^\top$ (nothing known), $\text{Prog}(M^0, K(\neg(x_1 \leftrightarrow x_2)))$ is $M^1 = O(x_1 \leftrightarrow \neg x_2)$, $\text{Prog}(M^1, \text{switch}(x_1))$ is $M^2 = O(x_1 \leftrightarrow x_2)$, and $\text{Prog}(M^2, K(\neg(x_1 \land x_2)))$ is $M^3 = O(\neg x_1 \land \neg x_2)$.

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

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

- if $\pi$ is the empty plan, then $\tau = (M^0)$;
- if $\pi$ is an ontic action $\alpha$, then $\tau = (M^0, \text{plain}(\text{Prog}(M^0, \alpha)))$;
- if $\pi$ is an epistemic action $\alpha$, then $\tau = (M^0, \text{Prog}(M^0, K(\alpha)))$ for some $K(\alpha) \in \Omega$, with $M^0 \not\models K(\alpha)$;
- for $\pi = \pi_1; \pi_2$, either $\tau = \pi_1$ with $\pi_1$ an infinite trace of $\pi_1$, or $\tau = \pi_1 \pi_2$ with $\pi_1$ a finite trace of $\pi_1$ and $\pi_2$ a trace of $\pi_2$;
- if $\pi$ is $\Phi$ then $\pi_1$ else $\pi_2$, then either $M^0 \models \Phi$ and $\tau$ is a trace of $\pi_1$, or $M^0 \not\models \Phi$ and $\tau$ is a trace of $\pi_2$;
- if $\pi$ is while $\Phi$ do $\pi_1$, then either $M^0 \models \Phi$ and $\tau$ is a trace of $\pi_1$; or $M^0 \not\models \Phi$ and $\tau = (M^0)$.

We say that $\pi$ terminates in $M^0$ if every trace of $\pi$ in $M^0$ is finite.

Example 2 Let $\pi, M^0, \ldots, M^3$ as in Ex. 1, and $M^4 = O(x_1 \land x_2)$.

The traces of $\pi$ in $M^0$ (with the corresponding feedbacks) are:

\[
\begin{align*}
(M^0, M^1, M^2, M^3) &\quad \rightarrow \quad (x_1 \leftrightarrow x_2), K(\neg(x_1 \land x_2)) \\
(M^0, M^1, M^2, M^4) &\quad \rightarrow \quad (x_1 \leftrightarrow x_2), (x_1 \land x_2) \\
(M^0, M^1, M^2) &\quad \rightarrow \quad (x_1 \leftrightarrow x_2), K(\neg(x_1 \land x_2)) \\
(M^0, M^1, M^4) &\quad \rightarrow \quad (x_1 \leftrightarrow x_2), (x_1 \land x_2)
\end{align*}
\]

4 KBPs vs. Standard Policies

We now briefly compare KBPs with standard policies (or plans) with respect to succinctness and expressiveness\(^4\). 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 $\pi$ can be translated into an equivalent standard policy (a “protocol” in [4]), by simulating all possible executions of $\pi$ 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, a standard policy $\pi$ 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 if it terminates, its traces can have length at most $|\pi|2^n$ (being twice in the same configuration would imply a potential 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 $2^{2n}$.

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 $\Delta^P_2$, but is both NP- and coNP-hard. Indeed, it is coNP-hard because $O \models K\varphi$ corresponds to $\varphi$ being valid, and NP-hard because $O \models \neg K\varphi$ corresponds to $\varphi$ 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 guaranteed to be executed less than $2^{2n}$ 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 (provided the size of plans is not restricted). 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 coNP-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 $\pi$ is valid for a planning problem. Precisely, we define a knowledge-based planning problem $P$ to be a tuple $(\Phi^0, A_0, A_E, G)$, where $\Phi^0 = O\varphi^0$ is the initial knowledge state, $G$ is a SKNNF $\varphi_0$ formula called the goal, and $A_0$ (resp. $A_E$) is a set of ontic (resp. epistemic) actions together with their theories. Then a KBP $\pi$ (using actions in $A_0 \cup A_E$) is said to be a (valid) plan for $P$ if its execution in $\Phi^0$ terminates, and for all traces $(M^0, \ldots, M^f)$ of $\pi$ with $M^0 = M(\Phi^0), M^f \models G$ holds. Intuitively, this means that executing $\pi$ always leads to a knowledge state where the agent is sure that $G$ holds. For instance, in Example 1, $\pi$ is a plan for $\Phi^0 = O \top$ and the goal $G = (K x_1 \lor K:\neg x_1) \land (K x_2 \lor K:\neg x_2)$.

Definition 1 (verification) The plan verification problem takes as input a knowledge-based planning problem $P = (\Phi^0, A_0, A_E, G)$ and a KBP $\pi$, and asks whether $\pi$ is a plan for $P$.

In this section we show that verification is $\Pi^P_2$-complete for while-free KBPs, even under several further restrictions. Observe first that a while-free KBP always terminates.

We start with membership in $\Pi^P_2$. In the broad lines, the argument is that $\pi$ is not a plan for $P$ if there exists a trace $\tau$ of $\pi$ (or, equivalently, a sequence of feedbacks for the epistemic actions executed) in which the last knowledge state does not satisfy $G$. Hence $\pi$ 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 $O\varphi$ cannot be performed in polynomial space.

Example 3 The progression of $O\varphi$, for $\varphi = \bigwedge_{i=1}^n (x_i \lor y_{i} \rightarrow z_i) \land (\bigwedge_{i=1}^n z_i \rightarrow z)$, by reinit($\{z_1, \ldots, z_n\}$), is equivalent to
Proposition 3 Plan verification is $\Pi^0_2$-hard. Hardness holds even if the KBPs $\pi$ 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 ($\Pi^0_2$-complete) problem of deciding the validity of a QBF $\forall x_1 \ldots x_p \exists y_1 \ldots y_q \varphi$, where $\varphi$ 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 $\pi$, with $\Phi^0 = \top$ and $G = \neg K \varphi$.

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

Similarly, with $A_E = \emptyset$ let $\pi = \text{reinit}((x_1, \ldots, x_p))$. Then $\pi$ is not valid if and only if there is a trace of $\pi$, i.e., values for $x_1, \ldots, x_p$, such that $K \neg \varphi$ 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 $x_1x_2x_3 > x_1x_2x_3 > x_1x_2x_3 > \cdots > x_1x_2x_3$ for $n = 3$ variables. Given $X$ and a knowledge state $M$ over a superset of $X$, we write $M_X$ for $\{x \mid s \in M\}$, 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 $2^{2^{\mathcal{O}(n)}} - 1$. 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 $2^{2^{\mathcal{O}(n)}} - 1$ of them (every nonempty subset of $2^n$). Hence there can be a program $\pi$ which passes through $2^{2^{\mathcal{O}(n)}}$ different configurations while being specified with only $\mathcal{O}(n)$ 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 $M_X = \{x \mid s \in M\}$, and stores it over some auxiliary variables $g_1, \ldots, g_n$. For instance, if the current knowledge state satisfies $M_X = \{x_1x_2x_3, x_1x_2x_3, x_1x_2x_3\}$, then after executing the routine, the agent knows $g_1 \land \neg g_2 \land \neg g_3$ (and $M_X$ is unchanged).

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

If $K(\neg x_1)$ then $g_1 := 0$ else $g_1 := 1$;
If $K((x_1 \iff g_1) \rightarrow \neg x_2)$ then $g_2 := 0$ else $g_2 := 1$;
We now introduce an ontic action which adds a given state \( s_a \in 2^X \) to \( M \). We assume \( s_a \) is encoded over some auxiliary variables \( a_1, \ldots, a_n \). The action \( a_\text{add} \) 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 \( \bigwedge_{i=1}^n x_i \leftrightarrow a_i \) (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 \( \bar{a} \).

We finally introduce a routine \( \pi^n \), which removes a given state \( s_r \in 2^X \) (encoded over auxiliary variables \( r_1, \ldots, r_m \)) from \( M \). We assume that the agent knows the state to be removed, that is, \( M \) satisfies \( K_r \lor K\neg r \) 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 \( s_g \in M \), ensuring \( M \setminus \{s_r\} \neq \emptyset \). Then \( \pi^n \) removes \( s_r \) from \( M \) by identifying a distinguished state \( s_g \neq s_r \) in \( M \), then executing an action \( a_\text{add} \) which maps any state to itself except for \( s_r \), which it maps to \( s_g \).

We identify \( s_g \) by running \( \pi^n_2 \). 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 \( \pi^n_2 \), then \( \pi^n_2 \) replaces \( s_g \) with the least assignment in \( M \), using the dual of \( \pi^n_2 \). Finally, \( a_\text{add} \) is defined to be the deterministic ontic action with theory \( \bigwedge_{i=1}^n x_i \leftrightarrow (x_i \oplus (\bigwedge_{i=1}^n x_i \leftrightarrow r_i) \land (x_i \oplus g_i)) \) (and no effect on auxiliary variables). A case analysis shows that \( a_\text{add} \) maps \( s \) to itself except for \( s_r \), which it maps to \( s_g \), as desired.

**Proposition 4** The progression \( M' \) of a knowledge state \( M \) by \( \pi^n \) satisfies \( \bigwedge_{i=1}^n K\pi^n_i \), where \( \pi^n_i = g_i \) (resp. \( \neg g_i \)) if the greatest state in \( M \) satisfies \( x_i \) (resp. \( \neg x_i \)). The progression by \( a_\text{add} \) (resp. \( \pi^n_2 \)) satisfies \( M' = M \cup \{s \} \) (resp. \( M' = M \setminus \{s \} \)).

Importantly, \( \pi^n_2, a_\text{add}, \pi^n \) all have a description of size at most quadratic in \( n \). Finally, we use a routine, written \( \pi_d \) (“decrement”), which replaces the state encoded by \( g_1, \ldots, g_n \) by its predecessor \( \pi > \) (its definition is straightforward, and omitted for space reasons).

**A Slow KBP** We see a knowledge state \( M \) as a vector \( \vec{m} = m_1 m_2 \cdots m_n \) where \( m_i = 1 \) if and only if the \( i \)-th state \( s_i \) (wrt \( \pi > \)) is in \( M \). Then our KBP starts with \( \vec{m} = 00 \ldots 01 \) (i.e., \( M = \{1 \ldots 1\} \equiv K \pi_1 \land \cdots \land \pi_n \)), and loops until \( \vec{m} = 10 \ldots 00 \) (\( M = \{00 \ldots 0\} \equiv K \neg x_1 \land \cdots \land \neg x_n \)). The loop changes the current \( \vec{m} \) to \( \vec{m} + 1 \) 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 \( m \) according to the Gray Code is obtained from \( m \) as follows:

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

For instance, the enumeration is 0001, 0011, 0101, 0110, \ldots, 1000 for \( n = 2 \) (we do not use 0000). In terms of knowledge states, this is \( \{x_2, x_1, x_1, x_2, x_1, x_2, x_2, x_2, x_1, x_1, x_2, x_1, x_2, x_2, x_2\} \), which indeed passes through all knowledge states.

**Proposition 5** The unique trace for \( \vec{m} \) in \( M \) has size \( 2^{2^n} - 1 \).
π(α := ψ) is the same as for the action α^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 reinit(h), where h is an auxiliary variable, for simulating a coin flip (h stands for "heads"), and we define π(a1|a2) to be (reinit(h); πh(a1); πh(a2)).

**Lemma 2** For any HJ-action a, the KBP π(a) can be built efficiently, and for any M, the progressions M[HJ] and M' of M by a (resp. π(a)) satisfy (M[HJ])X = M'X (ignoring auxiliary variables).

**Proposition 6** The verification problem for KBPs is EXPSPACE-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^n [5].

Given an NUP instance (ϕ^0, A_1, ϕ_G), we build the knowledge-based planning problem (Q_π^0, A_2, A_E, ¬K_ϕ_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 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_1 = {a_1, ..., a_k}: this is achieved by flipping k coins (using reinit({h_1, ..., h_k})) and executing the first action whose coin turned heads (determined by taking the epistemic actions test(h_i)^p).

The KBP simulate is depicted as Algorithm 3. We let clock^n be a KBP which counts up to 2^n (obtained, say, by adding a dummy action to clock^n). Clearly, simulate can be built in polynomial time.

**Algorithm 3: The KBP simulate**

```
initialize clock^n +
{Loop until NUP goal reached for sure or clock beeps};
while ¬K_ϕ_G ∧ ¬K(¬x_1 ∧ ... ∧ ¬x_n) do
  run one step of clock^n +;
  reinit({h_1, ..., h_k}); test(h_1); ...; test(h_k);
  if K[h_1 then π(a_1)];
  else if K[h_2 then π(a_2)];
  ...
  else if K[h_k then π(a_k)];
```

Let p be a plan of length at most 2^n for the NUP instance. Then by definition, the trace of simulate in which precisely the actions in p are chosen by reinit({h_1, ..., h_k}) ends up with K_ϕ_G being true, i.e., the goal ¬K_ϕ_G being false. Hence simulate is not valid. Conversely, only a choice of actions which achieve K_ϕ_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 KVP verification, hence the latter is coEXPSPACE-hard, that is, EXPSPACE-hard. □

**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^n times would necessarily start at least twice in the same knowledge state and hence run forever, such loops are unrolled 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 (which would grow as the number of steps unrolled), we maintain M 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. □

## 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 ones in the sense that they can be exponentially more compact, but execution is computationally more difficult, as well as verification.

Compactness of plans is an important feature in many contexts. An example context is the sending of plans from the Earth to a rover on a distant planet, where time windows severely limit communication. In general, this work is a first step towards bridging knowledge-based programming and AI planning. We are currently investigating plan generation, that is, synthesis of KBPs.

Another interesting direction for future work is to identify (more) tractable fragments for verification. The complexity of verifying purely epistemic or purely ontic KBPs with while is still open. On another dimension, it could be interesting to consider KBPs with executions bounded by, for instance, a polynomial, which can be verified by simulation with memoryful progression.

## ACKNOWLEDGEMENTS

Supported by the French National Research Agency under grant ANR-10-BLAN-0215.

## REFERENCES