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

To cite this version:
Jérôme Lang, Bruno Zanuttini. Knowledge-Based Programs as Plans - The Complexity of Plan Verification -. Proc. 20th Conference on European Conference on Artificial Intelligence (ECAI 2012), Aug 2012, France. 6 p., 2012. <hal-00944347>

HAL Id: hal-00944347
https://hal.archives-ouvertes.fr/hal-00944347
Submitted on 13 Feb 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Knowledge-Based Programs as Plans
– The Complexity of Plan Verification –

Jérôme Lang1 and Bruno Zanuttini2

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 overlooked. 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 $Kp$ then $π′$, where $K$ 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]).

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 Laverty 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 $\Pi_2^P$-completeness for while-free KBPs (Section 5) and 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, $M = \{x_1 x_2, \overline{x_1} x_2\}$ 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 $ϕ, ψ$, etc. whereas general $S_5$ formulas are denoted by $Φ, Ψ$ etc. For an objective formula $ϕ$, we denote by $Mod(ϕ)$ the set of all states which satisfy $ϕ$ (i.e., $Mod(ϕ) = \{s \in 2^X | s \models ϕ\}$). The size $|Φ|$ of an $S_5$ formula $Φ$ is the total number of occurrences of propositional symbols, connectives and modality $K$ in $Φ$.

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 $Φ$ 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 $Φ$ 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, $K\neg(p \land q) \lor \neg K(r \lor K\neg r)$ is not in KNNF, but is equivalent to $K\neg(p \land q) \lor (\neg K r \land \neg K r)$. Summarizing, a subjective $S_5$ formula $Φ$ in KNNF (for short, $Φ ∈ SKNNF$) is either a positive (resp. negative) epistemic atom $K ϕ$ (resp. $\neg K ϕ$), where $ϕ$ is objective, or a combination of such atoms using $\lor, \land$.

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

- $M \models K ϕ$ if for all $s′ \in M$, $s′ \models ϕ$.
- $M \models \neg K ϕ$ if $M \not\models K ϕ$.
- $M \models Φ \land Ψ$ (resp. $Φ \lor Ψ$) if $M \models Φ$ and (resp. or) $M \models Ψ$.

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 $Φ \models Ψ$, 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 $K ϕ \land K ψ$ 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@unicaen.fr
K(φ ∧ ψ), but that Kφ ∨ Kψ is not equivalent to K(φ ∨ ψ): for instance, K(φ ∨ ¬ψ) is valid whereas Kφ ∨ K¬ψ is true only when the agent knows the value of φ. Also note that K¬φ entails ¬K¬φ, and that M |= ¬Kφ if and only if M |= K¬φ.

We use a syntactical representation of the current knowledge state M ⊆ 2X (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(φ), and we represent M by the epistemic atom Oφ. Intuitively, Oφ means “I know φ and nothing else”. Without loss of generality, we disallow occurrences of O or K in the scope of O or K. Formally, O is an epistemic modality whose semantics is given in the logic of all I know [11] by

- M |= Oφ if M = Mod(φ).

Hence any atom Φ of the form Oφ has exactly one model, written M(Φ) = Mod(φ) ⊆ 2X. We use the term “knowledge state” to refer either to some M ⊆ 2X or to some atom Oφ. For instance, M = {x1, x2, x1, x2, x1, x2} will also be written Φ = O(¬x1 ∨ x2). Satisfiability in the logic of all I know is Σ2-complete [14]. However, we only need restricted entailment tests, which we show to be easier. We recall that Σ2 = P^NP is the class of all decision problems that can be solved in deterministic polynomial time using NP-oracles. Π2 = coNP^NP the class of all decision problems whose complement can be solved in nondeterministic polynomial time using NP-oracles, and EXPSPACE the class of all decision problems that can be solved using exponential space.

**Proposition 1** Deciding Oφ |= Φ, where φ is objective and Φ is purely subjective, is in Δ2.

**Proof** Let φ and Φ be as in the claim. Hence φ is a Boolean combination of atoms Kψ. We give a polynomial time algorithm for deciding Oφ |= Φ with a linear number of calls to an oracle for propositional satisfiability, reasoning by induction on the structure of the φ.

First let φ = Kψ. Then Oφ |= Φ reads ∀M ⊆ 2X. (M |= Oφ ⇒ ∀s ∈ M, s |= ψ). Since Oφ has exactly one model M = Mod(φ), this is equivalent to φ |= ψ, i.e., φ ∧ ¬ψ is not satisfiable.

Now let φ = 1ψ. Then Oφ |= Φ if ϕ entails Φ, or it entails Φ2. Indeed, Oφ has only one model M(0φ), hence Oφ entails Φ if M(Oφ) satisfies 1ψ, or Φ2, that is, if Oφ |= 1ψ or Oφ |= 1ψ holds. Hence, deciding Oφ |= Φ 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 α ∈ A is a KBP.
- if π and π′ are KBPs, then π ∨ π′ is a KBP.
- for KBPs π, π′ and φ ∈ SKNNF, if Φ then π else π′ is a KBP.
- for a KBP π and φ ∈ SKNNF, while φ do π is a KBP.

The class of while-free KBPs is obtained by omitting the while construct. The size |π| 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\(^3\).

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’ | x ∈ X}, denoting the values of variables after the action was taken. The theory of α is a propositional formula Σα over X ∪ X’ such that for all states s ∈ 2X, the set {s’ ∈ 2X | s’ |= Σα} is nonempty and is exactly the set of possible states after α is performed in s. For instance, with X = {x1, x2}, the action α which nondeterministically initializes the value of x1 has the theory Σα = (x1 ↔ x2).

In the paper we will use the following actions:

- reinit(Y) (for some Y ⊆ X) with theory \(\bigwedge_{x \in Y} x’ \leftrightarrow x\).
- \(\alpha_2 := \varphi \) (for φ objective) with theory \(\alpha_2 = \varphi \land \bigwedge_{x \in Y} x’ \leftrightarrow x,\)
- switch(xi) with theory \(x’_i \leftrightarrow \neg x_i \land \bigwedge_{j \not= i} x’_j \leftrightarrow x_j,\)
- 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 \(\Omega = (K\varphi_1, \ldots, K\varphi_n).\) For instance, the epistemic action which senses the value of an objective formula φ is

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

Finally, for an objective formula φ over X, we write \(\varphi_t\) for the formula obtained from φ by replacing each occurrence of \(x \in X\) with \(x’\). We also write \(\Sigma^{t+1}\) for the formula obtained from \(\Sigma\) by replacing each unprimed variable \(x \in X\) with \(x\), and each primed variable \(x’\) with \(x’+\). For instance, for \(\varphi_1 = (x_1 \lor x_2), \varphi_1^t = (x_1^t \lor x_2^t),\) and for \(\Sigma = (x_2^t \leftrightarrow x_2), \Sigma^{t+1} = (x_2^t \leftrightarrow x_2^t)).\)

#### 3.2 Semantics

The agent executing a KBP starts in some knowledge state \(M_0\), and at any timestep \(t\), 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 \(M_t' = \varphi\)).

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 \(Prog(M, a) = M' \subseteq 2^X\) defined by \(M' = \{s' \in 2^X | s' |= \Sigma_a\}.\) Intuitively, after taking α in a state which it knows to be in one, 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 \(\Sigma_n\)), but not which one.

Now given an epistemic action α, a knowledge state \(M\), and a feedback \(K\varphi_1 \in \Omega_n\) with \(\varphi \neq K\varphi_1\), the progression of \(M\) by \(K\varphi_1\), is defined to be \(Prog(M, \varphi_1) = M' = \{s' \in M \mid s' |= \varphi_1\}.\) The progression is undefined when \(M = K\neg \varphi_1\). Intuitively, a state is considered to be possible after obtaining feedback \(\varphi_1\), if and only if 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.

\(^3\) This induces a loss of generality, but in practice, if α is not executable in s, this can be expressed by letting let α lead to a sink (nonglobal) state.
Example 1 (from [6]). Consider the following KBP \( \pi \):

\[
\text{test}(x_1 \leftrightarrow x_2);
\]

If \( M^0 = O \top \) (nothing known), \( \text{Prog}(M^0, K(-(x_1 \leftrightarrow x_2))) \) is

\[
M^1 = O(x_1 \leftrightarrow \neg x_2), \quad \text{Prog}(M^1, \text{switch}(x_1)) = M^2 = O(x_1 \leftrightarrow x_2), \quad \text{and} \quad \text{Prog}(M^2, K(-(x_1 \land x_2))) = M^3 = O(\neg x_1 \land \neg x_2).
\]

With \( M^0 = O \top \) (nothing known), \( \text{Prog}(M^0, K(-(x_1 \leftrightarrow x_2))) \) is

\[
M^1 = O(x_1 \leftrightarrow \neg x_2), \quad \text{Prog}(M^1, \text{switch}(x_1)) = M^2 = O(x_1 \leftrightarrow x_2), \quad \text{and} \quad \text{Prog}(M^2, K(-(x_1 \land x_2))) = M^3 = O(\neg x_1 \land \neg x_2).
\]

We are now ready to give an operational semantics for KBPs. Given a knowledge state \( M^* \) involving only primed variables of the form \( x^+ (x \in X) \), we write \( \text{plain}(M^*) \) 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, \( i.e. \tau = (M^0, M^1, \ldots, M^T) \), 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 \( \tau \), then \( \tau = (M^0, \text{Prog}(M^0, K_\pi)) \) for some \( K_\pi \in \Omega_\pi \) with \( M^0 \nleq K_\pi \);
- for \( \pi = \pi_1 \land \pi_2 \), either \( \tau = \pi_1 \) with an infinite trace of \( \pi_1 \), or \( \tau = \pi_1 \land \pi_2 \) with a finite trace of \( \pi_1 \) and \( \tau_2 \) a trace of \( \pi_2 \);
- if \( \pi \) is if \( \Phi \) then \( \pi_1 \) else \( \pi_2 \), then either \( M^0 \models \Phi \) and \( \tau \) is a trace of \( \pi_1 \), or \( M^0 \nleq \Phi \) 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, \( M^0 \nleq \Phi \) 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 \( \pi, M^0, \ldots, M^4 \) 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:

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

4 KBPS VS. STANDARD POLICIES

We now briefly compare KBPs with standard policies (or plans) with respect to succinctness and expressiveness. 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 \( |\pi|^2 \) different configurations (value of each variable plus control point in the policy), hence if it terminates, its traces can have length at most \( |\pi|^2 \) (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 \( 2^{n^2} \).

4 For space reasons, our discussion is informal. Proofs and details are omitted.
\(O(\exists z_1, \ldots, z_n, \varphi) \equiv \bigwedge_{i \leq r \leq s} (x_{r, g_0}, \ldots, x_{r, g_0} (f_1 \land \cdots \land f_n \rightarrow z))\), which has no polynomial representation, while \(|\varphi|\) is linear.

Hence we introduce another form of progression, called memo-

### Proposition 3 Plan verification is \(P^2\)-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 \((P^2\)-complete) problem of deciding the validity of a QBF \(\forall x_1 \ldots x_n \exists y_1 \ldots y_m \varphi\), where \(\varphi\) is a propositional formula over \(\{x_1, \ldots, x_n\} \cup \{y_1, \ldots, y_m\}\). In both cases we build a planning problem \(P = (\Phi^0, A_O, A_E, G)\) and a KBP \(\pi\), with \(\Phi^0 = \emptyset\) 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 \(\neg K \varphi\) 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 \(y_1, \ldots, y_m\), \(\varphi\) is false, that is, to \(\forall x_1 \ldots x_p \exists y_1 \ldots y_m \varphi\) 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 \(\neg K \varphi\) holds, i.e., \(\exists x_1 \ldots x_p \exists y_1 \ldots y_m \varphi\) is not valid. □

### 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 poly-size 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, \(2^n\) 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 \(\{s_X \mid s \in M\}\), 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 \(2^n - 1\). 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 \(2^n\)). Hence there can be a program \(\pi\) which passes through \(2^n\) 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 one 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 \(M_X = \{x_1x_2x_3, x_1x_2x_3, x_1x_2x_3\}\), then after executing the routine, the agent knows that the greatest satisfies \(g_1 \land \neg g_2 \land \neg g_3\) (and \(M_X\) is unchanged).

We define \(\pi^0\) to perform a dichotomic search in \(M\). For instance, if \(K(x_1 \rightarrow g_1) \rightarrow \neg x_2\) is true, then no assignment in \(M\) which satisfies \(x_1 \rightarrow g_1\), i.e., by construction, none of the assignments with greatest \(x_1\) satisfies \(x_2\) hence the greatest one satisfies \(\neg x_2\). Precisely, \(\pi^0\) is the following KBP:

If \(K(\neg x_1)\) then \(g_1 := 0\) else \(g_1 := 1\);
If \(K((x_1 \rightarrow g_1) \rightarrow \neg x_2)\) then \(g_2 := 0\) else \(g_2 := 1\);
If $\Lambda^m$...
Finally, for nondeterminism we use the action \( \text{reinit}(h) \), 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 \( (\text{reinit}(h); \pi_h(a_1); \pi_h(a_2)) \).

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

**Proposition 6** The verification problem for KPBs is \( \text{EXPSPACE-hard} \). Hardness holds even if only one while-loop is allowed and the KPBs to be verified are known to terminate.

**Proof** We use both results that deciding whether an NUP instance has a plan is \( \text{EXPSPACE-complete} \), 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 \( (\varphi^0, A_{HJ}, \varphi_C) \), we build the knowledge-based planning problem \( \langle O_{\varphi}^1, A_0, A_{E}, \neg K_\varphi, G \rangle \) and the KPB simulate. This KPB 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 \( A_{HJ} = \{ a_1, \ldots, a_k \} \): this is achieved by flipping \( k \) coins (using \( \text{reinit} \{ h_1, \ldots, h_k \} \)) and executing the first action whose coin turned heads (determined by taking the epistemic actions \( \text{test}(h_i) \)).

The KPB simulate is depicted as Algorithm 3. We let clock\(^n\) be a KPB 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 KPB simulate

```plaintext```
Pseudocode: simulate
```
```
Let \( n \) 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 \( \text{reinit} \{ h_1, \ldots, h_k \} \) ends up with \( K_\varphi G \) being true, i.e., the goal \( \neg K_\varphi \neg G \) being false. Hence simulate is not valid. Conversely, only a choice of actions which achieve \( K_\varphi G \) 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 KPB verification, hence the latter is \( \text{coEXPSPACE-hard} \), that is \( \text{EXPSPACE-hard} \).

**Proposition 7** The verification problem for KPBs is \( \text{EXPSPACE} \).

**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

---

\[ \pi_r(\psi := \psi) \text{ is the same as for the action } a^r \text{ in Section 6.1.} \]

---

**REFERENCES**


