Knowledge-Based Programs as Succinct Policies for Partially Observable Domains

Bruno Zanuttini, Jérôme Lang, Abdallah Saffidine, François Schwarzentruber

Knowledge-Based Programs as Succinct Policies for Partially Observable Domains[☆]

Bruno Zanuttini[a,*], Jérôme Lang[b], Abdallah Saffidine[c], François Schwarzentruber[d]

[a] *Normandie Univ.; UNICAEN, ENSICAEN, CNRS, GREYC, 14000 Caen, France*
[b] *CNRS and LAMSADE, Université Paris-Dauphine, CNRS UMR 7243, France*
[c] *University of New South Wales, Sydney, Australia*
[d] *Univ. Rennes, CNRS, IRISA, France*

Abstract

We suggest to express policies for contingent planning by knowledge-based programs (KBPs). KBPs, introduced by Fagin *et al.* [*Reasoning about Knowledge*, MIT Press, 1995], are high-level protocols describing the actions that the agent should perform as a function of their current knowledge: branching conditions are epistemic formulas that are interpretable by the agent. The main aim of our paper is to show that KBPs can be seen as a succinct language for expressing policies in single-agent contingent planning.

KBP are conceptually very close to languages used for expressing policies in the partially observable planning literature: like them, they have conditional and looping structures, with actions as atomic programs and Boolean formulas on beliefs for choosing the execution path. Now, the specificity of KBPs is that branching conditions refer to the belief state and not to the observations.

Because of their structural proximity, KBPs and standard languages for representing policies have the same power of expressivity: every standard policy can be expressed as a KBP, and every KBP can be "unfolded" into a standard policy. However, KBPs are more succinct, more readable, and more explainable than standard policies. On the other hand, they require more online computation time, but we show that this is an unavoidable tradeoff. We study knowledge-based programs along four criteria: expressivity, succinctness, complexity of online execution, and complexity of verification.

[☆] This paper is a thoroughly extended version of two conference papers [45, 46].

[*] Corresponding author

Email addresses: bruno.zanuttini@unicaen.fr (Bruno Zanuttini), lang@lamsade.dauphine.fr (Jérôme Lang), abdallah.saffidine@gmail.com (Abdallah Saffidine), francois.schwarzentruber@ens-rennes.fr (François Schwarzentruber)

1. Introduction

In many applications, artificial agents make decisions in a partially observable environment. They follow a policy, which is sometimes specified manually by a policy designer, sometimes automatically computed (and sometimes partly specified manually, and completed automatically). At an abstract level, such policies (or plans) in partially observable domains are mappings from belief states to actions.

The encoding of belief states, and the mapping from belief states to actions, varies from one framework to another. As soon as the space of belief states has a combinatorial structure (because a state is defined by the value taken by some variables), belief states cannot be represented explicitly as set of states but are represented in a factorized way: when executing a plan, rather than explicitly maintaining a current belief state, one proceeds to *belief tracking*: belief states are expressed implicitly, for instance by the initial belief state and the history of actions and observations, and can be "queried" so as to know whether the goal is reached or the preconditions of some action are satisfied. With this variety of representations of belief states comes a variety of representations of policies: they can be represented by listing all possible histories and their associated action, or all belief states and their associated action, or the set of reachable belief states and their associated action, or as a finite automaton.

What are the criteria that help us evaluating the quality of a representation of policies? We suggest the following list.

1. Policies should be *understandable* by humans in order for artificial intelligent agents to be socially accepted. Indeed, when a failure occurs, the decisions made by the agent should be justified, for instance for legal issues.
2. Policies should be *concise*, because of *limited storage space*. Consider a fully autonomous nanorobot navigating through a human body. Its tiny size imposes constraints on the size of the embedded plan, yet the number of possible state-action trajectories can be huge. Another example where a tension between the size of the plan and storage space arises is that of a robot exploring a totally unknown environment where communication with the base may not be possible, or possible only in extremely short periods.
3. Policies should be *generic*. They should be generic enough to be instantianted on a number of distinct planning problems of a common family.
4. Policies should be *verifiable*. There should exist reasonably efficient algorithms to check whether a given policy enables the agent to reach a given goal in a given environment.
5. Policies should be *reactive*: at execution time, the choice of the next action to execute should be made as quickly as possible.

As already well-known and well-understood in the AI planning community, *these five requirements are incompatible*. After making this statement formal,

in order to fill desiderata 1–4 we advocate for representing a policy with a *knowledge-based program* (KBP). KBPs have been originally introduced by the distributed system community [32]. They use the principle of belief tracking, but they also use a slightly different representation of policies than usual plans coupled with belief tracking.

In such programs, branching conditions are epistemic formulas that are interpretable by the concerned agent. Hence to some extent, they can be seen as mappings from *sets of belief states* to actions, where sets of belief states are represented succinctly using epistemic formulas.

There are other representations of this kind. For instance, the representation of a policy by a set of α-vectors for POMDPs can also be seen as a map from (compact representations of) sets of belief states to actions in probabilistic settings. However, arguably this is *not* a representation which is easy to read, edit, or explain.[1]

It should be noted that KBPs are not maximally succinct: the most succinct policy for a planning problem is simply its specification. At execution time, it produces one action at a time, based on the observation, by using an on-line contingent planning algorithm.[2] This means that the complexity of execution is as hard as the complexity of partially planning observable planning with branching, that is, 2-EXPTIME-hard [62]. Thus KBPs offer a tradeoff between the latter representation (maximally succinct but with a huge cost at execution) and the explicit observation-based representation (maximally efficient to execute but extremely large). As we show, this tradeoff is reasonable, as the execution problem is only Θ_2^P-complete (hence one step of execution can be done using a logarithmic number of calls to a SAT solver) while we can have an exponential gain in succinctness. KBPs are certainly not the only representation "in-between": others have been used in the planning community, such as finite automata. Yet, such automata, as reactive representations, may also be exponentially larger than KBPs, as we show (Section 5).

To get a grasp of why it is interesting to express policies as KBPs, let us consider two examples.

Example 1 (minesweeper)**.** *Minesweeper is a single-player game, played on a (H, W)-board, where the objective is to*

> *clear a rectangular board containing hidden "mines" or bombs without detonating any of them, with help from clues about the number of neighbouring mines in each field.*[3]

The player initially has a number of hints, in the form of the number of neighbouring mines for some positions, as well as the total number of mines on the board.

[1] Of course, representations of policies for POMDPs are more complex, since the beliefs states are probability distributions over states.

[2] We thank one of the reviewers for this suggestion.

[3] Description taken from Wikipedia (`https://en.wikipedia.org/wiki/Minesweeper_(video_game)`) on November 14, 2019.

while the agent does not know that the objective is fullfilled **do**

if the agent knows there is no mine at $\langle 1,1\rangle$ **then** click on $\langle 1,1\rangle$
if the agent knows there is no mine at $\langle 1,2\rangle$ **then** click on $\langle 1,2\rangle$
...
if the agent knows there is no mine at $\langle H,W\rangle$ **then** click on $\langle H,W\rangle$

od

Figure 1: Example of a knowledge-based program for a Minesweeper board of size (H,W).

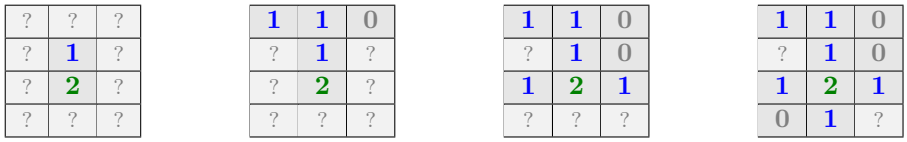

Figure 2: Example execution of the KBP of Example 1 for Minesweeper.

The knowledge-based program of Figure 1 prescribes the player to repeatedly clear each position in which it knows there is no mine, until the objective is achieved. Obviously, in minesweeper, whether the player has a winning strategy depends on the initial hints.

Assume for instance that the board is of size 4×3, and that the agent initially knows that there are 2 mines, among which 1 neighbouring $\langle 2,2\rangle$ and 2 neighbouring $\langle 3,2\rangle$, as depicted on Figure 2 (left). The agent infers that there is no mine in the first row, so that it clicks positions $\langle 1,1\rangle, \langle 1,2\rangle, \langle 1,3\rangle$. Assume that this reveals the numbers of neighbouring mines depicted on Figure 2 (middle left). The agent can now infer that there is a mine at $\langle 2,1\rangle$, the second one being at $\langle 4,1\rangle$, $\langle 4,2\rangle$, or $\langle 4,3\rangle$, and (consequently) that there is no mine at any of $\langle 2,3\rangle, \langle 3,1\rangle, \langle 3,3\rangle$. Again following the KBP, the agent clicks these positions (in a second round of the **while** *loop). Assuming the observations depicted on Figure 2 (middle right), the agent will finally infer that there is no mine at $\langle 4,1\rangle$ nor at $\langle 4,2\rangle$ and click these, resulting in the situation depicted on Figure 2 (right), in which the agent has (knowingly) achieved the goal and hence stops.*

Example 2 (diagnosis)**.** *Consider a system composed of three components; for each $i = 1,2,3$, we have a propositional symbol ok_i meaning that the component is in working order, an action $\mathrm{replace}_i$ that makes ok_i true, and an action test_i that returns the truth value of ok_i; the background knowledge about the system,* i.e.*, the logical relationship between which components are in order and which ones are not, can be anything. Let κ be the following KBP ($\mathbf{K}\varphi$ is read "the agent knows that formula φ is true"):*

while $\bigvee_i (\neg \mathbf{K}\mathrm{ok}_i \wedge \neg \mathbf{K}\neg \mathrm{ok}_i)$

do

let i be the first index such that $\neg\mathbf{K}\mathrm{ok}_i \wedge \neg\mathbf{K}\neg\mathrm{ok}_i$**;**

test_i **;**

if $\mathbf{K}\neg\mathrm{ok}_i$ **then** $\mathrm{replace}_i$ **fi**

od

It is easy to see that κ succeeds in making the goal $\mathrm{ok}_1 \wedge \mathrm{ok}_2 \wedge \mathrm{ok}_3$ *true, while avoiding unnecessary replacing actions, and, secondarily, avoiding unnecessary test actions. For instance, if the initial knowledge state of the agent is* $\mathbf{K}((\mathrm{ok}_1 \leftrightarrow (\mathrm{ok}_2 \wedge \mathrm{ok}_3)) \wedge (\neg\mathrm{ok}_1 \vee \neg\mathrm{ok}_2))$*, then the plan will start by replacing component 1, because it already knows, from its background knowledge, that it is not working correctly; then it will test 2; if* ok_2 *is observed then it will replace 3; else, it will replace 2, test 3, and replace it if needed.*

Observe that κ is generic *in the sense that it is a valid plan for* any *initial knowledge state.*

As we have already suggested, and will soon demonstrate, succinctness comes with a computational price: KBPs involve computationally hard (more precisely, Θ^2_{P}-hard) inference tasks at execution time. Admittedly, this is sometimes a serious issue. However, there are two reasons why it may be not that serious. First, these reasoning tasks consist of solving a small (logarithmic in the size of the input data) number of classical satisfiability problems; at an era when SAT solvers have become very efficient, for many applications and domain sizes, this is perfectly reasonable. Second, nothing prevents *unrolling* the KBP into a full explicit policy before executing it. For example, an exploring robot to which a succinct KBP is transmitted can unfold it into a fully explicit policy (possibly only up to some horizon, or only along some branches) and then execute it, in the manner of knowledge compilation [29].

In some domains, it is reasonable to assume that the KBP *is written by a human*, either an expert or a nonexpert, depending on the case. In other domains, the KBP can be computed automatically by compacting a tree-like policy generated by a standard planner. Yet in other domains, it can be partially specified by an expert and then completed automatically.[4] In all cases, all the rest is automated: verification, execution, and (if needed) unrolling. In this paper we focus on verification and execution.

To sum up, we study the use of knowledge-based programs as a language which is *generic*, *succinct*, and *easy to understand* for representing policies. Because of an unavoidable tradeoff, *executing KBPs is computationally hard* and requires online *belief tracking*. We put this into contrast with *reactive* representations of policies, such as finite-state automata branching on observations, which are easy to execute but are less generic, less succinct, and less easy to understand.

To avoid any misunderstanding we want to stress the following two points

[4] As an example of such partial specification, see [42]: the human expert gives rules specifying branches that are useless to explore, which the planning algorithm makes uses of.

concerning what our paper is *not* about:

- *we do not present a new paradigm for planning*: plans with branching and loops have been considered for long in the planning community; any knowledge-based program can be transformed into an equivalent standard policy, the advantage of knowledge-based programs being that they are more succinct than standard policies.

- *we do not present new algorithms for plan generation*: plan generation is orthogonal to our work; we assume that the plan has been written by an expert or (semi-)automatically computed.

Outline. In Section 2 we give an overview of related work, which spans several research fields: partially observable planning, agent programming languages, and logics of knowledge and action. In Section 3 we settle our context of partially observable planning domains and give the necessary background on epistemic logic. In Section 4 we recall knowledge-based programs and study their expressivity as representations of policies. In Section 5 we show that KBPs can be exponentially more succinct than reactive representations of policies. In Section 6 we study the complexity of executing a KBP and relate this to belief tracking. In Section 7 we study the computational complexity of verifying that a KBP is valid, considering various possible restrictions on the planning problem. Finally, in Section 8 we briefly discuss extensions of our work beyond the strict framework of this paper, and we conclude in Section 9.

2. Related Work

In this section, we review work related to ours, first about concise representations for planning (Section 2.1), then about knowledge-based programming (Section 2.2).

2.1. Related Work in Planning

In partially observable planning (and more generally in planning), the key components are (1) *the model*, (2) *the language*, and (3) *the algorithms* [34].

Choosing a model consists of choosing the mathematical objects for the various elements of planning problems, especially belief states, action effects, and goals. Uncertainty about states and action effects can be modelled qualitatively or probabilistically (a belief state being, respectively, a nonempty set of states or a probability distribution over states). Our model is the classical qualitative model, also referred to as that of *contingent planning*. In Sections 2.1.1 and 2.1.2 we position our work with respect to the state of the art in succinct representations and online execution of plans, respectively, for contingent planning. When the uncertainty is probabilistic, the associated model is that of *Partially Observable Markov Decision Processes*, which we discuss separately in Section 2.1.3.

2.1.1. Succinct Representations in Contingent Planning

When the set of states has an inherent combinatorial structure, its size is too large for allowing an explicit representation of sets of states, action effects, goals, and policies. In such contexts, a language for expressing succinctly the input and the output of a problem is needed. In such a case, a *factored* representation language consists of defining a problem through a set of relevant variables, whose domains are finite, and often binary. Here too we follow a classical approach by assuming binary variables (which can be done without loss of generality). Belief states are usually expressed as propositional formulas, possibly with some syntactical restriction; we do not make such a restriction (but our results could be extended to such contexts). Action effects can be represented in various languages; choosing a factored language or another has no significant impact on the succinctness and complexity of tasks [57].

Where we significantly depart from the classical literature is about the *representation of policies*.

The compact representation of plans and policies is already a relevant question for classical planning [7] and for conformant planning [2, 50]. However, as there are no observations in classical nor in conformant planning, plans do not need branching conditions; nor do they in *online contingent planning* (where the planner only computes the next action to execute) [17, 20, 52] and in *replanning* (where an unconditional plan is computed only for some branch of the execution tree) [16, 19]. Note however that we will exhibit cases when a sequential plan is much more succinct if represented with branching conditions (see Section 7.2).

In contrast, *offline contingent planning* corresponds to our setting, in which there is uncertainty, nondeterminism, and partial observability: then a policy must specify which action should be taken at each step, conditionally on what has occurred so far and for all possible contingencies. One can distinguish *weak*, *strong*, and *strong cyclic* policies [26], depending on when we consider the policy to be valid. We focus here on *strong* policies, which must terminate and achieve the goal under all circumstances.

The literature considers three formats for the representation of policies in offline contingent planning [34]: (1) mappings from histories (sequences of actions and observations performed so far) to actions, (2) mappings from succinctly represented belief states to actions; (3) finite automata whose transitions are labelled by actions.

Each of these representations comes with its own problems: history-based policies need to consider all histories (whose number is exponential in the number of observations made until the current stage); belief-based policies need to consider all belief states (whose number is doubly exponential in the number of variables); finite automata are difficult to read and understand by humans.

These three representations differ in the choice they make for the representation of branching conditions: sequences of observations, succinctly represented belief states, or states in a finite automaton. Belief-based policies often consider only a *subset* of possible beliefs, namely those that can occur at some point of the execution of the policy. Such policies are used in particular by works which

consider *regression* for planning, that is, computing policies backwards from the goal to the initial knowledge state [63, 23]. In such approaches, algorithms naturally compute a representation of policies mapping belief states to actions.

The specificity of *knowledge-based programs* with respect to other representations is that their branching conditions are complex epistemic formulas, that can represent both succinctly and intuitively sets of belief states, whose representation by belief states (even succinctly represented) would be exponential. A simple example (discussed further in the paper, see Section 5) is a branching condition such as *knowing the truth value of an even number of variables.*

Closest to our work is the representation of policies used by Muise et al. [56], where policies are represented as functions from compact belief states of the form $\bigwedge_i \mathbf{K}x_i \wedge \bigwedge_j \mathbf{K}\neg x_j$ to actions. The authors discuss the fact that such policies are succinct but that their execution requires maintaining a belief state, hence they resort to a conversion into a standard representation [56, Section 3.4]. However, no further investigation of this representation is done; it can be seen easily that it is a special case of the representation by KBPs.

Finally, let us mention the approach by Bolander et al. [14], where a representation of contingent policies in propositional dynamic logic is investigated. Due to branching on arbitrary formulas, this work bears close relationships to ours. However, the approach is semantical, and no succinctness or complexity questions are addressed.

2.1.2. Online Execution and Belief Tracking in Contingent Planning

Most of the research in planning consists of developing generic algorithms both for offline plan generation and for online plan execution. We do not consider plan generation here: we make the assumption that the plan has been already written by an expert or (semi-)automatically computed.

As far as online execution is concerned, the literature on planning considers two different contexts: either action selection is done online without any plan being computed offline (which is referred to as "online planning"), or a plan has been precomputed offline and expressed in some format, and action selection consists of retrieving, from this plan, the next action to perform at any stage of the execution. As the latter context is also ours, we now discuss it in more detail.

How the evaluation of branching conditions is implemented depends on the choice made for the representation of belief states. There have been proposals to represent belief states by action-observation sequences, with the use of a SAT solver for recognizing equal belief states [40], by literals conditioned on the initial state [1, 19], by logical formulas [70], or by binary decision diagrams [12].

When executing a plan, we have to keep track of beliefs in one way or another. The current belief state can be computed from the previous belief state and the last action performed and observation received; while this works in theory, in practice this is less easy, as this belief state may quickly become very large (in the worst case, it is exponential in the number of propositional variables). However, as argued in several papers [18, 70, 20], in order to execute a plan, we do not need to store these belief states explicitly; it is rather sufficient

to be able to answer these three types of queries: given an execution history, (1) has the goal been achieved? (2) which actions are executable, that is, have their precondition satisfied? (3) if a given executable action is performed, which observations are possible?

These tasks are referred to as *belief tracking*. Although they are all untractable in the worst case, still, it is possible to keep track of the beliefs necessary for solving them in time and space exponential in a width parameter associated with the problem, and which measures the degree of interaction between variables with respect to the set of available actions. When this width is too large for an exact computation of belief states to be tractable, a solution is to use approximate belief tracking algorithms [17].

Belief tracking is orthogonal to our work. We focus on the representation of policies, while belief tracking focuses on their execution. While we prove worst-case complexity results about the execution of knowledge-based programs in Section 6, such execution needs the current branching condition to be evaluated and thus amounts to a belief tracking problem, namely: given the current sequence of actions performed and observations gathered, is the current branching condition satisfied? Results about polynomial time and space complexity parameterized by the width of the problem, and algorithms for approximate belief tracking, apply, and can be used to make the execution of a KBP easier. We will return to that in Section 6.

2.1.3. Partially Observable Markov Decision Processes

Partially Observable Markov Decision Processes (POMDPs) can be seen as a probabilistic counterpart of contingent planning problems, with probabilistic transition functions and a probabilistic observation model. The study of POMDPs dates back to the 1960s [4, 68], and they have become a dominant model for AI planning under partial observability since the 1990s [41] .

Representation issues, both for the input of the problem (action effects and observations), and the output (policies) are of course of primary importance, as in contingent planning. Representing policies succinctly is far more difficult than in contingent planning, first of all because the set of belief states is infinite. A well-known way of representing policies in a finite way consists in using the fact that value functions are piecewise linear and convex [68]; this yields a representation by α-*vectors*, which can be seen as compact representations of sets of belief states, although arguably much less readable than epistemic branching conditions as used by knowledge-based programs. On the other hand, *point-based methods* allow to consider a smaller number of belief states [59]. Anyway, the typical representation of policies in this domain is by finite-state controllers, that is, finite-state automata branching on the observations received, possibly further restricted to have a bounded number of states [60].

2.2. Related Work in Logic

Knowledge-based programs rely on epistemic logic, and were initially introduced for protocol specification [32]. Hence there is an important amount of work related to ours in the logic community.

2.2.1. Golog and the Situation Calculus

Golog is a high-level agent programming language, built on the situation calculus, that uses actions, knowledge, and several program constructs. Knowledge-based Golog programs have been considered first by Reiter [61], while Claßen and Lakemeyer [28], and later Claßen and Neuss [27] implement knowledge-based programs on modal variants of the situation calculus.

The situation calculus is a first-order language which is much more expressive than action languages based on propositional logic. But this high expressivity comes with a huge price: evaluating branching conditions is undecidable. Therefore, not only knowledge-based programs cannot be verified in finite time, but they can even not be executed. To some extent, our knowledge-based programs can be seen as a decidable, propositional fragment of Golog with knowledge-based programs.

The least we can require for a fragment of Golog to be acceptable for building knowledge-based programs on it is that it is decidable, so that branching conditions can be evaluated and a knowledge-based program can always be executed. A few attempts to identify decidable classes of epistemic situation calculus theories have been made [36, 30, 31, 74], but without addressing knowledge-based programming.

We know of only one work where knowledge-based programs are built over a decidable fragment of Golog: Zarrieß and Claßen [73] define a framework for knowledge-based programs where knowledge and actions are represented in an epistemic extension of the basic description logic DL. As DL is half-way between propositional logic and first-order logic, there is a hope for decidability, but the authors prove that it is not the case: evaluating branching conditions is undecidable in the general case (due to nondeterministic action choices). However, they identify two nested decidable fragments, for which evaluating branching conditions is respectively EXPTIME-complete and PSPACE-complete, with knowledge-based program verification being in 2-EXPSPACE. These results are very valuable because, as the authors say, they obtain decidability for a language that goes far beyond propositional logic. On the other hand, our choice to restrict to propositional logic admittedly makes programs less expressive, but also computationally *much easier* (and yet already difficult). Also, succinctness is not investigated by Zarrieß and Claßen [73].

2.2.2. Propositional Languages for Planning with Knowledge

The idea of using explicit knowledge preconditions for actions and plans comes back to work by Moore [54] and Morgenstern [55], and was developed further by Brafman et al. [21]. Propositional knowledge-based programs in a planning framework were first considered by Son and Baral [69]: their formalism is based on a specific language for expressing action effects (based on causal rules), and the syntax of their branching conditions is restricted to epistemic formulas of the form $\mathbf{K}\varphi$. Herzig et al. [39] go further and give a language for specifying partially observable planning with knowledge-based programs, which does not depend on a specific action representation language and allows any epistemic formula in branching conditions. Petrick and Bacchus [58] use

a simplified epistemic language for plans, where for each propositional variable x, two extra propositional symbols $\mathbf{K}x$ and $\mathbf{K}\neg x$ are used, expressing that x is known to be true (resp. false); in the same vein, Albore et al. [1] use a more general language with symbols $\mathbf{K}x/t$ and $\mathbf{K}\neg x/t$, meaning "it is known that if t is true in the initial situation, x is true (resp. false)".

2.2.3. Protocol Synthesis

Knowledge-based programs can be seen as a way of *specifying* concrete programs, rather than building them. This view is taken by several authors [21, 71, 9]. In this view, the underlying question is to compute a *protocol* (in our words, a *policy*), if any, which realizes a given specification, possibly with restrictions on the class of protocols and their operational semantics. Our work can be considered as complementary to these: we consider from the start a precise operational semantics for knowledge-based programs (see our Section 4.3), so that there is no question about how our KBPs can be implemented: they are by definition realized by this operational semantics. Said otherwise, we do not see KBPs as a *specification* language, but as a language in which we *describe* policies which can readily be executed (or automatically compiled into other executable forms).

2.2.4. Other Related Work

There exist variants of knowledge-based programming where the agent does not have knowledge, but *beliefs*, and which allow for uncertain action effects and noisy observations: probabilistic beliefs [10], and more qualitative beliefs, based on ordinal conditional functions (also known as "kappa functions") [48, 49]. Succinctness and plan verification are not investigated there.

Lastly, let us briefly mention a (more loosely) related stream of work, which concerns planning for *dynamic epistemic logic* (DEL) [51, 3, 5, 6]. In this context, actions are rather *events* with explicit effects on the knowledge of the agents (in addition to possible ontic effects). Nevertheless, the focus of epistemic planning in the literature is on the multi-agent setting, on the one hand, and on the other hand on the computation of sequential plans. Hence, maybe quite counterintuitively, "epistemic planning" refers to a line of work which (so far at least) has objectives different from ours.

3. Background

We define partially observable domains and contingent planning problems (Sections 3.1 and 3.2), and then the useful notions of epistemic logic (Section 3.3).

Since we deal with planning problems in factored representations, we assume a finite set X of propositional variables, and we use a standard propositional language built from atoms $x \in X$, the constants $\bot, \top$, and the connectives $\neg, \vee, \wedge, \rightarrow, \leftrightarrow, \oplus$. Assignments are denoted compactly; for instance, $x_1\overline{x_2}x_3$ denotes the assignment of $\top$ to x_1, x_3 and $\bot$ to x_2. We write $\mathcal{P}(X)$ for the set of

all assignments to all the variables in X. For an assignment $\mu \in \mathcal{P}(X)$ and a propositional formula φ over X, we write $\mu \models \varphi$ if μ satisfies φ, and we write $\mathrm{Sat}(\varphi)$ for the set of all assignments to X which satisfy φ (called *satisfying assignments* of φ).

3.1. Partially Observable Domains

Our model is a standard one for planning [35]; it can also be seen as the qualitative counterpart of (stochastic) partially observable Markov Decision Processes (POMDPs) [41].

Given a finite set of variables X, we call *state* any assignment s to all the variables in X, so that the *state space* of a planning domain is $\mathcal{P}(X)$, also denoted by $\mathcal{S}$. *Actions* from a finite set A (possibly) modify the current state and yield *observations* from a finite set O, through a partial transition function $\delta : \mathcal{S} \times A \to \mathcal{P}(O \times \mathcal{S}) \setminus \{\emptyset\}$. We also write $s \xrightarrow{a|o} s'$ for $(o, s') \in \delta(s, a)$. This means that each time action a is taken in state s, it may be the case (according to a nondeterministic choice by the environment) that the current state is modified from s to s' and that the agent receives observation o. Observe that both modifications of the state and observations are nondeterministic in general.

Following Brafman and Shani [20], we assume the following compact representation of the transition and observation function. A deterministic action a is defined to be a triple $\langle pre_a, \mathit{eff}_a, obs_a \rangle$, where

- pre_a is a propositional formula over X, denoting the precondition of a,
- eff_a is a set of ordered pairs $\langle cond_{a,\ell}, \ell \rangle$, at most one per literal ℓ over X, where $cond_{a,\ell}$ is a propositional formula over X denoting the condition under which a makes ℓ become true, and where for given a, ℓ, $cond_{a,\ell}$ and $cond_{a,\bar{\ell}}$ are mutually inconsistent,
- obs_a is a set of ordered pairs $\langle when_{a,o}, o \rangle$, at most one per observation $o \in O$, where $when_{a,o}$ is a propositional formula over X denoting the condition (over the outcome state) under which o is observed, and where for given a, formulas $when_{a,o}$ are mutually inconsistent and jointly exhaustive, that is, each state satisfies $when_{a,o}$ for exactly one $o \in O$.

We also write $cond_\ell$ and $when_o$ when the action is clear from the context.

More formally, the semantics is given by

$$s \xrightarrow{a|o} s' \iff \begin{cases} s \models pre_a \\ \text{for all literals } \ell, \quad \big(s' \models \ell \ \text{ iff } \ s \models (cond_{a,\ell} \vee (\ell \wedge \neg cond_{a,\bar{\ell}}))\big) \\ s' \models when_{a,o} \end{cases}$$

where the second line states that literals ℓ with $s \models cond_{a,\ell}$ are made true in s', and others are left unchanged (tacitly assuming $cond_{a,\ell} = \bot$ if no condition is given for ℓ).

Now a *nondeterministic* action a is simply modelled as a set $\{a_1, \ldots, a_n\}$ of deterministic actions all with the same precondition, one of which is nondeterministically selected when a is taken, that is,

$$s \xrightarrow{\{a_1,\ldots,a_n\}|o} s' \iff \exists i \in \{1,\ldots,n\}, s \xrightarrow{a_i|o} s'$$

An action is called *purely ontic* if it always yields the same observation, and *purely epistemic* if it never changes the state.

Definition 3 (partially observable domain)**.** *A* partially observable domain *is a triple $M = \langle X, A, O\rangle$, where $X = \{x_1, \ldots, x_n\}$ is a finite set of propositional* variables*, $A = \{a_1, \ldots, a_k\}$ is a nonempty finite set of nondeterministic actions over X and O, represented compactly, and $O = \{o_1, \ldots, o_p\}$ is a nonempty finite set of* observations*.*

Example 4 (continued)**.** *The domain for the Minesweeper (Example 1), on an $H \times W$ board, is formally defined as follows. The set of variables X_{ms} is defined to be $\{m_{i,j}, c_{i,j} \mid i = 1, \ldots, H, j = 1, \ldots, W\}$. Variable $m_{i,j}$ encodes that there is a mine at $\langle i, j\rangle$, variable $c_{i,j}$ encodes that $\langle i, j\rangle$ has already been cleared. The set of actions A_{ms} is defined to be $\{\text{click}_{i,j} \mid i = 1, \ldots, H, j = 1, \ldots, W\}$, and the set of observations to be $O_{\text{ms}} = \{o_0, \ldots, o_8, o_{\text{lost}}\}$, where o_n is observed when the agent clicks a position not containing a mine and having n mines among its neighbours, and o_{lost} is observed when a mine is detonated.*

For a state s and a position $\langle i, j\rangle$, let us write $s \cup c_{i,j}$ for the state equal to s except for $(s \cup c_{i,j})(c_{i,j}) = \top$, and $\text{N}(i, j)$ for the set of positions neighbouring $\langle i, j\rangle$. We can now define the transition function δ_{ms} by $\delta_{\text{ms}}(s, \text{click}_{i,j}) = \{(o_{\text{lost}}, s \cup c_{i,j})\}$ for $s(m_{i,j}) = \top$ and $\delta_{\text{ms}}(s, \text{click}_{i,j}) = \{(o_n, s \cup c_{i,j})\}$ for $s(m_{i,j}) = \bot$, with n the number of mines neighbouring $\langle i, j\rangle$.

For the factored form, two example conditions describing $\text{click}_{i,j}$ *are $\text{cond}_{c_{i,j}} = \top$, meaning that $c_{i,j}$ is always set to true by the action, and $\text{when}_{o_{\text{lost}}} = m_{i,j}$, meaning that o_{lost} is observed when $m_{i,j}$ is true.*

When an agent takes actions repeatedly in an environment governed by a partially observable domain M, the system evolves as follows. At each timestep t, the system is in some state s^t. Then the agent executes one of its actions a^t. If s^t does not satisfy pre_a, then the whole execution fails. Otherwise, a pair (o^t, s^{t+1}) is picked by the environment, o^t is perceived by the agent, and the system evolves to state s^{t+1}. We emphasize that the observation o^t is the only piece of information available to the agent: the agent does not observe the new state s^{t+1}.

A *run* for M is a (finite or infinite) sequence $r = s^0a^0o^0s^1a^1o^1s^2\ldots$ satisfying for all timesteps $t = 0, 1, \ldots$: $s^t \in \mathcal{S}$, $a^t \in A$, $o^t \in O$, and $s^t \xrightarrow{a^t|o^t} s^{t+1}$. For all suitable timesteps t, we also write $s^t(r)$ for s^t, $a^t(r)$ for a^t, and $o^t(r)$ for o^t.

We say that r *starts* in state s^0. For a finite run $r = s^0a^0o^0s^1\ldots s^{t-1}a^{t-1}o^{t-1}s^t$, we say that r *ends* in state s^t, and we call t its *length*.

For $r = s^0a^0o^0s^1a^1o^1s^2\ldots$, the information directly available to the agent is the *history* $h(r)$ induced by r, defined to be $h(r) = a^0o^0a^1o^1\ldots$. More generally,

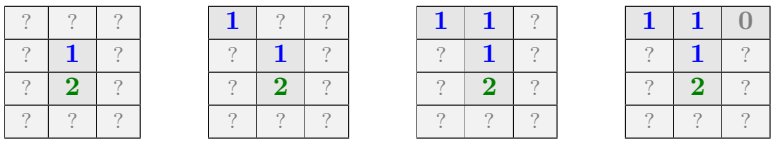

Figure 3: Example of sequence of states and observations for Minesweeper.

a *history* (resp. a *finite history*, a *history of length* t) is such a sequence which is induced by some run (resp. by some finite run, by some run of length t). We write $|h|$ for the length of h, and ϵ for the empty history.

Example 5 (continued)**.** *Consider the sequence of states and observations depicted on Figure 3, where the pictures represent states in the obvious manner (with mines at* $\langle 2,1\rangle$ *and* $\langle 4,3\rangle$*). The word* $r = s_1\,\mathrm{click}_{1,1}\,o_1\,s_2\,\mathrm{click}_{1,2}\,o_1\,s_3\,\mathrm{click}_{1,3}\,o_0\,s_4$ *is a run for our Minesweeper domain. The history induced by* r *is* $h(r) = \mathrm{click}_{1,1}\,o_1\,\mathrm{click}_{1,2}\,o_1\,\mathrm{click}_{1,3}\,o_0$*.*

3.2. Planning Problems

A planning problem takes place in a partially observable domain, but further specifies a set of (possible) *initial states* and a set of *goal states*. *Solving* such a problem means finding a policy for the agent to reach one of the goal states in finite time, starting in any of the possible initial states and whatever the outcome of nondeterministic actions. Still following Brafman and Shani [20], we assume the natural and standard representation of initial and goal states by propositional formulas over X.

Definition 6 (contingent planning instance)**.** *A* (contingent) planning instance *is a triple* $\Pi = \langle M, \varphi^I, \varphi^G\rangle$*, where* M *is a partially observable domain with set of variables* X *and* φ^I, φ^G *are propositional formulas over* X *called the* initial belief state *and the* goal*, respectively.*

When the concrete representation of the initial belief state and the goal does not matter, we sometimes denote a planning instance by $\langle M, I, G\rangle$ for *sets of states* I, G instead of formulas.

Example 7 (continued)**.** *For an* $H \times W$ *board, the contingent planning instance associated to the game of Minesweeper, with no position initially cleared and exactly 2 mines, is the triple* $\Pi_{\mathrm{ms}} = \langle M_{\mathrm{ms}}, \varphi^I_{\mathrm{ms}}, \varphi^G_{\mathrm{ms}}\rangle$ *defined by*

$$\begin{aligned} M_{\mathrm{ms}} &= \langle X_{\mathrm{ms}}, A_{\mathrm{ms}}, O_{\mathrm{ms}}\rangle \text{ as in Example 4} \\ \varphi^I_{\mathrm{ms}} &= \textstyle\bigwedge_{i,j}(\neg c_{i,j}) \wedge \bigvee_{\neq}(m_{i,j} \wedge m_{i',j'}) \wedge \bigwedge_{\neq} \neg(m_{i,j} \wedge m_{i',j'} \wedge m_{i'',j''}) \\ \varphi^G_{\mathrm{ms}} &= \textstyle\bigwedge_{i,j}(c_{i,j} \oplus m_{i,j}) \end{aligned}$$

where $\bigvee_{\neq}, \bigwedge_{\neq}$ *mean that the disjunction (resp. conjunction) is taken on all ordered pairs (resp. triples) of* pairwise different *positions.*

Note that we consider *ontic* goals; we briefly discuss the natural extension of our work to *epistemic* goals in Section 8.1.

Solutions to planning instances Π are given by *policies*. Conceptually, a policy maps finite histories to actions; it tells the agent which action to take depending on what it has done and observed so far.

Definition 8 (policy)**.** *Let $\langle X, A, O\rangle$ be a partially observable domain. A* policy *is a* partial *function from the set of all finite histories to A. A* finite-horizon policy *is a policy for which there exists $t \in \mathbb{N}$ satisfying that all histories h such that $\pi(h)$ is defined have length at most t.*

Observe that we allow policies to be partially defined. When $\pi(h)$ is undefined and h is the current history perceived by the agent, then the agent halts. In particular, a policy may define actions only for those histories which occur in the way from a given initial belief state to a given goal, and/or be defined only for histories up to some length. In general, a policy may define actions for histories of arbitrary length (*e.g.*, the policy defined by $\pi(h) = a$ for all finite histories h), though a finite-horizon policy cannot, by definition.

Example 9 (continued)**.** *Let $(p_t)_t$ be the enumeration $\langle 1,1\rangle, \langle 1,2\rangle, \langle 1,3\rangle \ldots$ of all the positions on the Minesweeper board. The function π_{ms} defined by $\pi_{\mathrm{ms}}(h) := \mathrm{click}_{p_{|h|}}$ is the policy that prescribes to click on the positions in the order specified by this enumeration, independently of what the agent perceives.*

Now write $t(h)$ for the index of the last position clicked in h ($a^{|h|-1}(h) = \mathrm{click}_{p_{t(h)}}$). Then the function π'_{ms} defined by $\pi'_{\mathrm{ms}}(\epsilon) := \mathrm{click}_{p_0}$ and for $h \neq \epsilon$, $\pi'_{\mathrm{ms}}(h) := \mathrm{click}_{p_{t(h)+1}}$ for $o^{|h|-1}(h) = o_0$, and $\pi'_{\mathrm{ms}}(h) := \mathrm{click}_{p_{t(h)+2}}$ otherwise (if defined), is the policy that prescribes to click on the positions in order, except when the last position clicked revealed one or more mines in its neighbourdhood (in which case the next position is skipped over). Both π_{ms} and π'_{ms} are finite-horizon policies (they are defined on histories of length at most $t = H \times W$).

We now define notions pertaining to which runs may occur given that the agent is executing a given policy. We say that a run $r = s^0a^0o^0s^1a^1o^1s^2\ldots$ is: *π-consistent* if for all timesteps $t = 0, 1, \ldots$, the action a^t is $\pi(a^0o^0 \ldots a^{t-1}o^{t-1})$; *$\pi$-maximal* if it is finite and π-consistent, but π is not defined on $h(r)$; and *M-safe* if for all timesteps $t = 0, 1, \ldots$, the state s^t satisfies the precondition pre_{a^t}. We also call a history *π-consistent* if it is induced by some run which is π-consistent.

Definition 10 (valid policy)**.** *A policy π is said to be* valid *for a contingent planning instance $\Pi = \langle M, I, G\rangle$ if the two following conditions are true:*

- *there exists $t \in \mathbb{N}$ such that all π-maximal runs r for M starting in a state $s \in I$ are finite, M-safe, and of length at most t;*
- *all π-maximal runs r for M starting in a state $s \in I$, end in a state $s' \in G$.*

Such a policy is also said to be t-valid.

In words, a valid policy must terminate in finite time on all (consistent) histories, take actions for which the current history *ensures* that the precondition is satisfied, and whatever actually occurred (consistent with the history), it must terminate in a state which satisfies the goal. In particular, if in some run the agent reaches a goal state but continues to act (following its policy), finally stopping in a nongoal state, then we consider that the policy is *not* valid.

Example 11 (continued)**.** *Consider the problem instance* Π_{ms} *given in Example 7, with* $H = 4$ *and* $W = 3$*. Then neither* π_{ms} *nor* π'_{ms} *of Example 9 is valid for* Π_{ms}*. Indeed, assume there are mines at* $\langle 2,1\rangle$ *and* $\langle 4,3\rangle$*, and that no position is initially cleared. Then starting from this state (*s^0*) and with initial belief* φ^I_{ms}*, the unique* π_{ms}*-maximal run consists of the agent clicking all 12 positions in order then stopping, hence this run ends in a state which does not satisfy the conjuncts* $c_{2,1} \oplus m_{2,1}$ *nor* $c_{4,3} \oplus m_{4,3}$ *of* φ^G_{ms}*. Similarly, the unique* π'_{ms}*-maximal run starting in* s^0 *consists of the agent clicking* $\langle 1,1\rangle, \langle 1,3\rangle, \langle 2,1\rangle, \langle 2,3\rangle, \langle 3,1\rangle, \langle 3,3\rangle, \langle 4,2\rangle$*, hence it ends in a state which does not satisfy the conjuncts for* $\langle 1,2\rangle, \langle 2,1\rangle, \langle 2,2\rangle, \langle 3,2\rangle, \langle 4,1\rangle$*.*

3.3. Epistemic Logic

Conditions in KBPs are *subjective* formulas of single-agent epistemic logic [32]. They are Boolean combinations of atoms of the form $\mathbf{K}\varphi$ which are read "the agent knows that the propositional formula φ holds".[5]

Definition 12 (subjective formulas)**.** *Let* X *be a finite set of propositional variables. The language* EL^X *of* subjective formulas *is defined by the following grammar, where* φ *ranges over propositional formulas over* X*:*

$$\Phi \;::=\; \mathbf{K}\varphi \;\mid\; \neg\Phi \;\mid\; \Phi \vee \Phi \;\mid\; \Phi \wedge \Phi$$

We also introduce the dual construction $\widehat{\mathbf{K}}\varphi$ as a shorthand for $\neg\mathbf{K}\neg\varphi$; it is read "the agent considers possible that the propositional formula φ holds".

Example 13 (continued)**.** *The formula*

$$\Phi_{\mathrm{ms}} = \mathbf{K}m_{2,1} \wedge \mathbf{K}(m_{4,1} \vee m_{4,2} \vee m_{4,3}) \wedge \widehat{\mathbf{K}}m_{4,1} \wedge \widehat{\mathbf{K}}m_{4,2} \wedge \neg\widehat{\mathbf{K}}(m_{4,1} \wedge m_{4,2})$$

is read "the agent knows that there is a mine at $\langle 2,1\rangle$*, it knows that there is one at (at least) one of* $\langle 4,1\rangle, \langle 4,2\rangle, \langle 4,3\rangle$*, it considers it possible that there is one at* $\langle 4,1\rangle$ *and possible that there is one at* $\langle 4,2\rangle$*, but not that there is one at both.*

Epistemic formulas are usually interpreted on pointed Kripke structures, but for subjective formulas, there is no loss of generality if they are interpreted on

[5] As the agent is positively and negatively introspective (the agent knows what it knows and what it does not know), there would be no additional expressiveness, were nesting of modalities allowed, so that we do not allow such nesting in the language.

sets of assignments to X. We call such a set $B \subseteq \mathcal{P}(X)$ a *belief state* (over X).[6] Intuitively, in planning, the belief state of an agent is the set of all states s such that the agent considers that the current, actual state may be s.

Definition 14 (semantics of EL)**.** *Let X be a finite set of propositional variables, let B be a belief state over X, and let $\Phi \in \mathsf{EL}^X$ be an epistemic formula. Then B is said to* satisfy *Φ, written $B \models \Phi$, if one of the following conditions holds:*

$$\begin{array}{lll} \Phi = \mathbf{K}\varphi & \textit{and} & \textit{for all states } s \textit{ in } B, \textit{ we have } s \models \varphi, \\ \Phi = \neg\Psi & \textit{and} & B \not\models \Psi, \\ \Phi = \Psi_1 \vee \Psi_2 & \textit{and} & B \models \Psi_1 \textit{ or } B \models \Psi_2, \\ \Phi = \Psi_1 \wedge \Psi_2 & \textit{and} & B \models \Psi_1 \textit{ and } B \models \Psi_2. \end{array}$$

Note that $B \models \widehat{\mathbf{K}}\varphi$ if there exists a state s in B satisyfing $s \models \varphi$.

Example 15 (continued)**.** *Let B_{ms} (resp. B'_{ms}) be the belief state containing all states which are consistent with the situation depicted on the middle left (resp. middle right) of Figure 2 (and the assumption that there are 2 mines). Then B_{ms} satisfies the formula Φ_{ms} of Example 13, but B'_{ms} does not, since it does not satisfy its conjunct $\widehat{\mathbf{K}}m_{4,1}$.*

4. Knowledge-Based Programs

We first formally recall the syntax of knowledge-based programs (KBPs), as introduced by Fagin et al. [32]. We then discuss their semantics as policies for a partially observable model.

4.1. Syntax

KBPs are built from actions using sequence, branching, and iteration. Conditions are subjective epistemic formulas.

Definition 16 (KBP)**.** *Let $M = \langle X, A, O\rangle$ be a partially observable model. A* Knowledge-Based Program *(KBP) is an expression generated by:*

$$\kappa \;::=\; \varepsilon \mid a \mid \kappa \,;\, \kappa \mid \textbf{if } \Phi \textbf{ then } \kappa \textbf{ else } \kappa \textbf{ fi} \mid \textbf{while } \Phi \textbf{ do } \kappa \textbf{ od}$$

where ε is the empty program, a ranges over A, and Φ ranges over EL^X.

We often omit the *else* part (as usual) when the corresponding subprogram is empty, and we sometimes enclose a KBP inside bold brackets ($[\![\ldots]\!]$) to promote readability. We call *while-free* any KBP which does not contain any **while** construct.

The intended interpretation is the straightforward one for all constructs. However, it depends of how branching and continuation conditions are evaluated when a KBP is executed, for what we give a precise definition in Section 4.3.

[6] The term "knowledge state" would be more appropriate, however, "belief state" is the term most used in the planning literature, especially for POMDPs.

while $\neg \mathbf{K}\varphi^G_{\text{ms}}$ **do**
  if $\mathbf{K}\neg m_{1,1}$ **then** $\text{click}_{1,1}$ **else** ε **fi;**
  if $\mathbf{K}\neg m_{1,2}$ **then** $\text{click}_{1,2}$ **else** ε **fi;**
  ...
  if $\mathbf{K}\neg m_{H,W}$ **then** $\text{click}_{H,W}$ **else** ε **fi**
od

Figure 4: The formal example of the KBP for Minesweeper.

Note that a KBP cannot branch on an objective property φ of the current state, because the current state is not (directly) observable by the agent. This is why epistemic formulas occurring in KBPs are restricted to be *subjective*.

Example 17 (continued)**.** *Figure 4 essentially reproduces our KBP* κ_{ms} *for Minesweeper, with the formal syntax of KBPs.*

Let us also emphasize that we do *not* allow a KBP to use auxiliary variables (as we may want to do for, say, storing the number of mines whose positions we do not know yet in Minesweeper). Definition 16 indeed restricts the epistemic branching conditions to be over the set of variables X which defines the problem. We put this restriction so as to keep KBPs readable, as variables in X typically encode tangible features of the environment.

4.2. Progression

In order to define how KBPs are executed by an agent, we first recall the standard definition of the *progression* of a belief state by an action and an observation [18]. Intuitively, when the agent has a belief state B (representing the states candidate for being the actual one), executes an action, and receives an observation, the progressed belief state represents the knowledge which it has at the new timestep, assuming that it reasons perfectly.

Definition 18 (progressed belief state)**.** *Let* $\langle X, A, O\rangle$ *be a partially observable model,* $B \subseteq \mathcal{P}(X)$ *be a belief state,* $a \in A$ *be an action, and* $o \in O$ *be an observation. The* belief state $\text{Prog}(B, a, o)$ progressed by a and o starting in B *is defined by*

$$\text{Prog}(B, a, o) = \left\{ s' \in \mathcal{S} \mid \exists s \in B, s \xrightarrow{a|o} s' \right\}.$$

For a finite history h*, the* belief state progressed by h starting in B*, written* $\text{Prog}(B, h)$*, is defined by* $\text{Prog}(B, \epsilon) = B$ *and* $\text{Prog}(B, aoh') = \text{Prog}(\text{Prog}(B, a, o), h')$.

Example 19 (continued)**.** *Figure 5 (left) shows a situation together with the corresponding belief state* B *(framed rectangle, where we ignore the value of* $c_{i,j}$*'s). The KBP* κ_{ms} *prescribes action* $\text{click}_{1,1}$*, and as a result the agent receives*

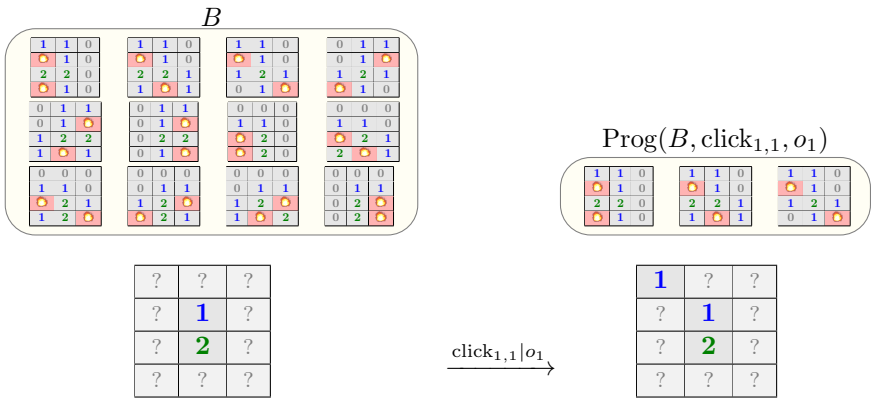

Figure 5: Example of progression for minesweeper in which the agent clicks on the top-left cell ($\text{click}_{1,1}$) and observes that there is one mine around that cell (o_1).

observation o_1. The result of progressing B by this action and observation is depicted on Figure 5 (right).[7]

Observe that progression by an action with nondeterministic (ontic) effects tends to enlarge the belief state (because there are several s' for each s), and that progression by an observation tends to reduce it (because observations rule out some states). For instance, if there are two variables x_1, x_2, the progression of $B^0 = \{\overline{x_1}\overline{x_2}, x_1\overline{x_2}\}$ by a purely ontic action which nondeterministically sets x_2 to the value of x_1 or leaves it unchanged, is $B^1 = \{\overline{x_1}\overline{x_2}, x_1x_2, x_1\overline{x_2}\}$, and further taking a purely epistemic action revealing that x_2 is false yields the belief state $B^2 = \{\overline{x_1}\overline{x_2}, x_1\overline{x_2}\}$. However, this is not always the case, as different effects may yield the same outcome, and observations may reveal nothing in a given belief state. For instance, progressing B^2 by a purely ontic action which nondeterministically sets x_1 to the value of x_2 or to $\bot$, yields $B^3 = \{\overline{x_1}\overline{x_2}\}$, and further observing that x_1 is false yields $B^4 = B^3$.

4.3. Operational Semantics

Before giving the operational semantics, we first get rid of certain KBPs that run into infinite series of tests without taking any action, e.g. [**while** $\top$ **do** ε **od**].

A KBP *takes an action for sure* if it is of the form $[a]$, [**if** Φ **then** κ_1 **else** κ_2 **fi**], where κ_1, κ_2 take an action for sure, or of the form $[\kappa_1$ **;** $\kappa_2]$ where at least one of κ_1, κ_2 takes an action for sure.

Definition 20 (well-formed KBP)**.** *A KBP is said to be* well-formed *if (i) all its subprograms are well-formed and (ii) it is ε, it takes an action for sure,*

[7] An animation of progression of belief states in Minesweeper and other domains is available at `http://hintikkasworld.irisa.fr` [66, 25].

$\dashv$	if $\kappa = \varepsilon$
(a, ε)	if $\kappa = a$
$(\mathrm{Act}_B(\kappa_1), [\mathrm{Cont}_B(\kappa_1)\ ;\ \kappa_2])$	if $\kappa = [\kappa_1\ ;\ \kappa_2]$ and $\llbracket \kappa_1 \rrbracket_B \neq \dashv$
$(\mathrm{Act}_B(\kappa_2), \mathrm{Cont}_B(\kappa_2))$	if $\kappa = [\kappa_1\ ;\ \kappa_2]$ and $\llbracket \kappa_1 \rrbracket_B = \dashv$
$\llbracket \kappa_1 \rrbracket_B$	if $\kappa = [\textbf{if } \Phi \textbf{ then } \kappa_1 \textbf{ else } \kappa_2 \textbf{ fi}]$ and $B \models \Phi$
$\llbracket \kappa_2 \rrbracket_B$	if $\kappa = [\textbf{if } \Phi \textbf{ then } \kappa_1 \textbf{ else } \kappa_2 \textbf{ fi}]$ and $B \not\models \Phi$
$(\mathrm{Act}_B(\kappa_1), [\mathrm{Cont}_B(\kappa_1)\ ;\ \kappa])$	if $\kappa = [\textbf{while } \Phi \textbf{ do } \kappa_1 \textbf{ od}]$ and $B \models \Phi$
$\dashv$	if $\kappa = [\textbf{while } \Phi \textbf{ do } \kappa_1 \textbf{ od}]$ and $B \not\models \Phi$

Figure 6: KBP evaluation $\llbracket \kappa \rrbracket_B$ defined by induction on κ.

it is of the form $[\kappa_1\ ;\ \kappa_2]$ *or* $[\textbf{if } \Phi \textbf{ then } \kappa_1 \textbf{ else } \kappa_2 \textbf{ fi}]$*, or it is of the form* $[\textbf{while } \Phi \textbf{ do } \kappa \textbf{ od}]$*, where* κ *takes an action for sure.*

We are now ready to formally define how well-formed KBPs are executed. At any timestep t, the branching conditions Φ are evaluated with respect to the current belief state B, that is, the agent decides whether $B \models \Phi$ holds. Then the execution amounts to evaluate all branching conditions until reaching an action (or the empty KBP). Hence execution consists of a series of *belief tracking* queries, as we explain in details in Section 6.

Given a well-formed KBP κ and a belief state B, the *KBP evaluation* $\llbracket \kappa \rrbracket_B$ is either (i) the pair $(\mathrm{Act}_B(\kappa), \mathrm{Cont}_B(\kappa))$, where $\mathrm{Act}_B(\kappa)$ is the next action to take given that conditions are evaluated in B, and $\mathrm{Cont}_B(\kappa)$ is the KBP with which to continue execution after taking action $\mathrm{Act}_B(\kappa)$, or (ii) the symbol $\dashv$, meaning that the execution stops because there is no next action anymore. The KBP evaluation is formally defined by induction on κ, as presented on Figure 6.

Since κ is well-formed, the evaluation of $\kappa = [\textbf{while } \Phi \textbf{ do } \kappa_1 \textbf{ od}]$ is well-defined: κ_1 takes an action for sure, thus $\mathrm{Act}_B(\kappa_1)$ is defined.

4.4. Induced Policies

With this in hand, for a given initial belief state, a well-formed KBP induces a policy, formally defined as follows.

Definition 21 (induced policy)**.** *Let* $\langle X, A, O \rangle$ *be a partially observable model,* κ *be a well-formed KBP, and* B *be a belief state. The policy* induced by κ starting in B*, written* $\pi_{\kappa,B}$*, is defined on finite histories as follows.*

- $\pi_{\kappa,B}(\epsilon) = \mathrm{Act}_B(\kappa)$ *if* $\llbracket \kappa \rrbracket_B \neq \dashv$ *holds;*
- $\pi_{\kappa,B}(a^0 o^0 h') = \pi_{\kappa',B'}(h')$*, with* $\kappa' = \mathrm{Cont}_B(\kappa)$ *and* $B' = \mathrm{Prog}(B, a^0, o^0)$*, if* $a^0 = \mathrm{Act}_B(\kappa)$ *and* $\mathrm{Prog}(B, a^0, o^0) \neq \emptyset$ *hold.*

In all other cases, $\pi_{\kappa,B}(h)$ *is undefined.*[8]

[8] In the second case, it might also be undefined because $\pi_{\kappa',B'}(h')$ is itself undefined.

Let us insist that a KBP does not define a policy *per se*, but only relative to an initial belief state (possibly $\top$). Hence KBPs are *generic* in the sense that they induce a policy for all possible initial belief states (though a KBP does not necessarily induce a *valid* policy for all initial belief states, of course).

Example 22 (continued)**.** *Consider again the KBP* κ_{ms} *and assume that it is run in the situation of Figure 5 (left), as adequately captured by a belief state* B*. Then the policy* $\pi_{\kappa_{\mathrm{ms}},B}$ *satisfies* $\pi_{\kappa_{\mathrm{ms}},B}(\epsilon) = \mathrm{click}_{1,1}$*. Indeed, since we have* $B \models \neg\mathbf{K}\varphi^G_{\mathrm{ms}}$*, by Definition 21 and the definition of* $[\![\kappa_{\mathrm{ms}}]\!]_B$*, we have* $\pi_{\kappa_{\mathrm{ms}},B}(\epsilon) = \mathrm{Act}_B(\kappa_{\mathrm{ms}}) = \mathrm{Act}_B(\kappa'_{\mathrm{ms}})$*, where* κ'_{ms} *is the subprogram consisting of the body of the* **while** *loop in* κ_{ms}*, and in turn, since we have* $B \models \mathbf{K}\neg m_{1,1}$*, we have* $\mathrm{Act}_B(\kappa'_{\mathrm{ms}}) = \mathrm{Act}_B(\mathrm{click}_{1,1}) = \mathrm{click}_{1,1}$*.*

Now consider the history $\mathrm{click}_{1,1}\, o_1$*. Again by Definition 21, we have*

$$\pi_{\kappa_{\mathrm{ms}},B}(\mathrm{click}_{1,1}\, o_1) = \mathrm{Act}_{B'}([\mathrm{Cont}_B(\kappa'_{\mathrm{ms}})\ ;\ \kappa_{\mathrm{ms}}])$$

with $B' = \mathrm{Prog}(B, \mathrm{click}_{1,1}, o_1)$ *(which is depicted on the right of Figure 5). Now* $\mathrm{Cont}_B(\kappa'_{\mathrm{ms}})$ *is* [**if** $\mathbf{K}\neg m_{1,2}$ **then** $\mathrm{click}_{1,2}$ **else** ε **fi;** ...] *(the body of the* **while** *loop starting with the second* **if** *subprogram), and since we have* $B' \models \mathbf{K}\neg m_{1,2}$*, we have* $\pi_{\kappa_{\mathrm{ms}},B}(\mathrm{click}_{1,1}\, o_1) = \mathrm{click}_{1,2}$*.*

With a similar reasoning, we get $\pi_{\kappa_{\mathrm{ms}},B}(\mathrm{click}_{1,1}\, o_0) = \mathrm{click}_{1,2}$*. On the other hand,* $\pi_{\kappa_{\mathrm{ms}},B}(\mathrm{click}_{1,1}\, o_2)$ *is not defined, since* $\mathrm{Prog}(B, \mathrm{click}_{1,1}, o_2)$ *is empty (meaning that* $\mathrm{click}_{1,1}\, o_2$ *is not a valid history, as can be seen on Figure 5).*

Importantly, observe that the language of KBPs is robust to syntax in the sense that the operational semantics of, say, [**if** $\Phi_1 \wedge \Phi_2$ **then** κ **else** κ' **fi**] and [**if** Φ_1 **then** [**if** Φ_2 **then** κ **else** κ' **fi**] **else** κ' **fi**] are the same, as is desirable for a natural programming language.

4.5. Expressivity

We conclude this section by observing that while-free KBPs are expressive enough for representing valid finite-horizon policies. This is a well-known result in the planning community, usually stated as the fact that belief states form a sufficient statistics for planning [15]. The intuition is that if there is a policy π which achieves a goal φ^G after a certain history, then there is a KBP which achieves the same goal whenever the *regression* of φ^G through π is known to hold [39]. Another way to see this is that even if policies are formally defined to be mappings from histories to actions, only the knowledge brought about by the history (that is, the current belief state) is relevant to planning ahead, which is essentially due to the fact that the dynamics of our planning problems is Markovian.

Proposition 23. *Let* $\Pi = (M, I, G)$ *be a contingent planning instance. If for some* $t \in \mathbb{N}$*, there is a* t*-valid policy* π *for* Π*, then there is a while-free KBP* κ *such that* $\pi_{\kappa,I}$ *is* t*-valid for* Π*.*

Note that Proposition 23 does *not* show that all policies can be represented by a KBP. Indeed, the agent may receive different observations that yield the

same knowledge update. Though a policy might in general prescribe different actions for these different observations, KBPs cannot make the difference when evaluating epistemic conditions, and hence must prescribe the same action. But of course this is harmless for the validity of the policy.

5. Succinctness

In this section, we demonstrate the exponential gain in succinctness of KBPs with respect to a variety of other classes of representations of policies. We exhibit a family of planning instances with valid policies which (1) can be represented as succinct (while-free) KBPs, but for which (2) there is no valid, succinct policy in these other classes (not even another one than the one represented by a succinct KBP), under a standard complexity-theoretic assumption ($\mathsf{NP} \not\subseteq \mathsf{P/poly}$, see below). Observe that this does not prevent some of these other representations to be exponentially smaller than KBPs for other families of instances. However, this shows that they are at best incomparable to KBPs as concerns succinctness.[9]

The classes of representations which we consider are *all* classes of *reactive* representations, where by "reactive" we mean that the policy can be executed efficiently at each step, as well as maps from belief states to actions.

Definition 24 (reactive)**.** *A class $\mathcal{R}$ of representations of policies is said to be* reactive *if for all policies π, $\pi(h)$ can be computed in polynomial time in the size of the domain, the length of history h, and the size of the representation of π in $\mathcal{R}$.*[10]

Definition 25 (map from belief states to actions)**.** *A representation of a policy π for a contingent planning problem with initial state I is said to be* as a map from belief states to actions *if it consists of a set of ordered pairs (B_i, a_i), where for all i, B_i represents a belief state and a_i is $\pi(h)$ for all histories with* $\mathrm{Prog}(I, h) = B_i$.

Reactive representations cover many standard representations of policies (in particular policy trees and finite-state-controllers), and they correspond to *compact sequential-access representations* for standard planning as formalized by Bäckström and Jonsson [7].

On the other hand, maps from belief states to actions cover many representations of policies which rely on *belief tracking* at execution time [18, 20], and which vary on the representation of belief states: explicitly, for instance as BDDs

[9] The language of KBPs can be easily extended by branching also on the last observation received, and to a more compact representation allowing to share subprograms, in which case it is *always* at least as succinct, and possibly exponentially more so, than finite-state controllers, while all complexity results in this paper remain the same. Branching on the last observation received is studied in a multi-agent setting by [64]. Moreover, by definition, KBPs are always at least as succinct as maps from belief states to actions.

[10] The algorithm is required to return "undefined" if $\pi(h)$ is undefined.

or CNF formulas [70], or implicitly, as the history so far [33, 40]. These are in general *not* reactive, because of the size of the belief state to be maintained at execution, or because of the complexity of matching intensional representations to each other.

Note that KBPs can also be seen as maps from belief states to actions, but more generally they map *sets of belief states* (as captured by epistemic formulas) to actions. Though the idea is implicit in the contingent planning literature, to the best of our knowledge it has never been investigated as a concrete representation of policies, and most often policies are represented as maps from each (implicit or explicit) belief state which is reachable from the initial belief state to actions.

Overall, our results in this section show that (1) when the application at hand does not have too stringent constraints at execution time (like real-time execution), KBPs may have a clear advantage over any reactive representation, namely that of succinctness, in addition to readability,[11], and that (2) summarizing sets of belief states by epistemic formulas also gives a clear advantage over plain (implicit or explicit) belief states, again in terms of succinctness.

5.1. Construction

We build a family of contingent planning instances $(\Pi_n)_{n\in\mathbb{N}}$ of size $poly(n)$, where $poly(n)$ is a notation for any function bounded by a polynomial in n, in such a way that instances Π_n have valid KBPs of size $poly(n)$, but no valid policy with a reactive representation of size $poly(n)$, assuming $\mathsf{NP} \not\subseteq \mathsf{P/poly}$, nor with a representation as a map from belief states to actions of size $poly(n)$. Recall that NP is the class of decision problems decided by a nondeterministic polynomial-time Turing machine. Now $\mathsf{P/poly}$, introduced by Karp and Lipton [44], is the class of decision problems for which there is a deterministic polynomial-time Turing machine, and a family of finite words $(a_n)_{n\in\mathbb{N}}$, called *advice sequence*, such that $|a_n| = poly(n)$ and for all instances w of size n, w is a positive instance of the decision problem if and only if the Turing machine accepts $\langle w, a_n\rangle$. An alternative definition is as the class of decision problems for which there is a nonuniform polynomial-time algorithm, that is, a family of algorithms $(A_n)_{n\in\mathbb{N}}$ such that for all n, A_n decides the instances of size n in time polynomial in n (the advices in the previous definition can be seen as the code of these algorithms). The hypothesis $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ is widely believed to be true; in particular, $\mathsf{NP} \subseteq \mathsf{P/poly}$ would imply that the polynomial hierarchy collapses to the second level [44].

The idea of Π_n is to encode 3-SAT, the problem of deciding whether a 3CNF propositional formula (conjunction of clauses each of size 3) has at least one satisfying assignment [43]. Initially, the agent has no knowledge and it has to first "sense" a 3CNF formula ψ over n variables $x_1, \ldots, x_n$. Then the agent has actions to change the values of $x_1, \ldots, x_n$, enabling it to build a model of ψ, if

[11] Of course, succinctness also favours readability.

possible at all. It can also declare whether ψ is satisfiable or not. Moreover, we constrain a valid plan to reach the goal before some deadline T.

Precisely, let us first define the partially observable domain $M_n = \langle X_n, A_n, O_n \rangle$. Observe that there are $8\binom{n}{3} = \mathit{poly}(n)$ different clauses of length 3 over n variables; we write $\mathsf{3\text{-}Clauses}_n$ for the set of all clauses of length 3 over the variables $x_1, \ldots, x_n$.

The set of variables X_n of M_n contains the following $16\binom{n}{3} + 2n + 5$ variables:

- $x_1, \ldots, x_n$ are the propositional variables appearing in ψ;
- $x_{\gamma \in \psi}$, for all clauses $\gamma \in \mathsf{3\text{-}Clauses}_n$, read as "the 3CNF formula $\psi(x_1, \ldots, x_n)$ contains the clause γ";
- x_{wasTrue}, which is true if and only if ψ was true in the initial state;
- x_{end}, which becomes true when the agent declares whether ψ is satisfiable;
- x_{error}, which becomes true if the agent makes an error in this declaration;
- $x_{\mathrm{t}=0}, \ldots, x_{\mathrm{t}=T}$, encoding the current timestep in unary, where $T = 8\binom{n}{3} + n + 2$ is the last timestep.[12]

To sum up, a state intuitively contains an assignment to the variables $(x_1, \ldots, x_n)$, a 3CNF ψ over these n variables, the value of ψ wrt the initial values of $(x_1, \ldots, x_n)$, Boolean values stating that the agent declared whether ψ is satisfiable or not and encoding whether this declaration was erroneous, and the current timestep.

We introduce actions for "sensing" the presence of a clause in the 3CNF ψ, for setting the values of $x_1, \ldots, x_n$, and for declaring the formula ψ to be satisfiable or to be unsatisfiable. Precisely, the set of actions is

$$A_n = \{\mathrm{sense}_\gamma \mid \gamma \in \mathsf{3\text{-}Clauses}_n\} \cup \{x_i{:=}\top, x_i{:=}\bot \mid i = 1, \ldots, n\} \cup \{\mathrm{sat!}, \mathrm{unsat!}\}\,.$$

The set of observations is $O_n = \{o_{\mathrm{y}}, o_{\mathrm{n}}, o_{\mathrm{void}}\}$, which are read "yes, ψ has the clause sensed by the agent", "no, ψ does not have the clause sensed by the agent", and "void observation", respectively.

Action sense_γ does not change the current state but tells the agent whether γ appears in ψ via observation o_{y} or o_{n}. Action $x_i{:=}\top$ makes x_i true and yields a void observation, and dually for $x_i{:=}\bot$. Action sat! makes x_{end} true, and makes x_{error} true if the values $x_1, \ldots, x_n$ do not satisfy ψ. Dually, action unsat! makes x_{end} true, and makes x_{error} true if ψ was satisfied by the initial values of $x_1, \ldots, x_n$. Finally, all actions have a tautological precondition; no action changes propositions $x_{\gamma \in \psi}$ (that is, the 3CNF ψ is never modified) nor x_{wasTrue}; if x_{error} is true, it remains true; and the timestep is increased by 1 when an action is performed, except if the deadline T is reached. This description is

[12] Since $T = \mathit{poly}(n)$, we would have no benefit in encoding the timestep in binary.

Action	**Effects**	**Observations**
sense_γ	none	$when_{o_\text{y}} = x_{\gamma\in\psi}$ $when_{o_\text{n}} = \neg x_{\gamma\in\psi}$
$x_i{:=}\top$	$cond_{x_i} = \top$	$when_{o_\text{void}} = \top$
$x_i{:=}\bot$	$cond_{\bar{x}_i} = \top$	$when_{o_\text{void}} = \top$
sat!	$cond_{x_\text{end}} = \top$ $cond_{x_\text{error}} = \neg\varphi_\text{isTrue}$	$when_{o_\text{void}} = \top$
unsat!	$cond_{x_\text{end}} = \top$ $cond_{x_\text{error}} = x_\text{wasTrue}$	$when_{o_\text{void}} = \top$
all	$cond_{x_{\text{t}=t+1}} = x_{\text{t}=t} \quad \forall t \le T-1$ $cond_{\bar{x}_{\text{t}=t+1}} = \bar{x}_{\text{t}=t} \quad \forall t \le T-2$	$-$ $-$

Table 1: Description of actions for the construction in Section 5.1

captured by the description in Table 1, where formula φ_isTrue encodes the fact that the current assignment to $x_1, \dots, x_n$ satisfies the formula ψ:

$$\varphi_\text{isTrue} = \bigwedge_{\gamma\in\textsf{3-Clauses}_n} \left[x_{\gamma\in\psi} \to \left(\bigvee_{x_i\in\gamma} x_i \vee \bigvee_{\neg x_i\in\gamma} \neg x_i \right) \right]$$

This completes the description of the domain M_n. Intuitively, since the value of x_wasTrue never changes but is never observed, the agent can take action unsat! without running the risk of making x_error true only if it is sure that x_wasTrue is false, that is, that ψ was not satisfied by the (arbitrary) initial values of $x_1, \dots, x_n$, or, in other words, that ψ is unsatisfiable. Otherwise, its only way to achieve the goal is to use action sat!, and for this it needs to build a model of ψ.

We now define the rest of the instance Π_n. In the initial belief state φ^I, x_wasTrue stores whether ψ is satisfied by the current assignment to $x_1, \dots, x_n$, neither x_end nor x_error are true, and the current timestep is 0:

$$\varphi^I_n = (x_\text{wasTrue} \leftrightarrow \varphi_\text{isTrue}) \wedge \neg x_\text{end} \wedge \neg x_\text{error} \wedge x_{\text{t}=0} \wedge \bigwedge_{t=1}^{T} \neg x_{\text{t}=t}$$

The goal is to reach a state, before the deadline T, in which the agent has declared whether ψ is satisfiable or not, without having made an error:

$$\varphi^G_n = \neg x_{\text{t}=T} \wedge x_\text{end} \wedge \neg x_\text{error}$$

Clearly, the instance $\Pi_n = \langle M_n, \varphi^I_n, \varphi^G_n \rangle$ has size polynomial in n.

Proposition 26. *There is a family $(\kappa_n)_{n\in\mathbb{N}}$ of while-free KBPs such that κ_n is valid for Π_n and has size poly(n).*

Proof. Write $\textsf{3-Clauses}_n = \left\{\gamma_1, \gamma_2, \dots, \gamma_{8\binom{n}{3}}\right\}$, and consider the KBP κ_n given on Figure 7.

sense_{γ_1} ; ... ; $\text{sense}_{\gamma_{8\binom{n}{3}}}$;
if $\widehat{\mathbf{K}}\varphi_{\text{isTrue}}$ **then**
  if $\widehat{\mathbf{K}}(x_1 \wedge \varphi_{\text{isTrue}})$ **then** $x_1{:=}\top$ **else** $x_1{:=}\bot$ **fi;**
  if $\widehat{\mathbf{K}}(x_2 \wedge \varphi_{\text{isTrue}})$ **then** $x_2{:=}\top$ **else** $x_2{:=}\bot$ **fi;**
  ...
  if $\widehat{\mathbf{K}}(x_n \wedge \varphi_{\text{isTrue}})$ **then** $x_n{:=}\top$ **else** $x_n{:=}\bot$ **fi;**
  sat!
else
  unsat!
fi

Figure 7: The KBP κ_n of the proof of Proposition 26

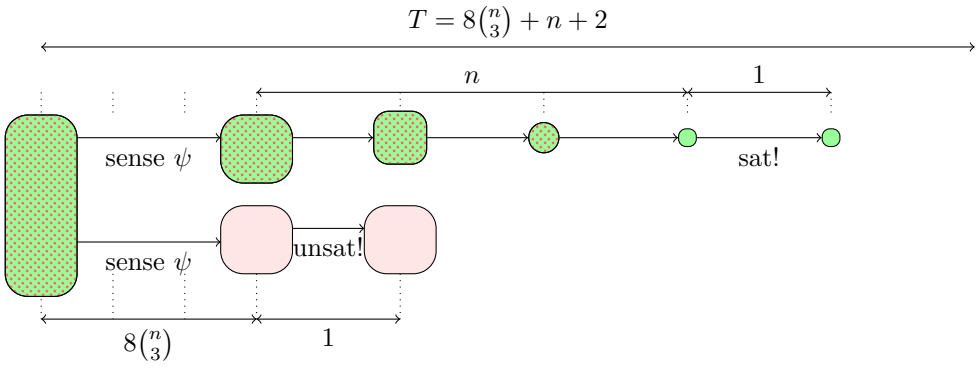

Figure 8: Overview of some executions of Π_n.

To show that κ_n is valid, let r be a κ_n-maximal run starting in φ_n^I, and write ψ for the formula encoded in the first state of r. Figure 8 shows the possible such runs. Clearly, r is finite and consists of at most $T - 1 = 8\binom{n}{3} + n + 1$ steps. Moreover, obviously r is M_n-safe, since all actions have a tautological precondition.

We now show that r ends in a state which satisfies φ_n^G. First, clearly $\neg x_{\mathrm{t}=T}$ holds in the last state of r. Now in case ψ is unsatisfiable, the agent can clearly infer that $\varphi_{\mathrm{isTrue}}$ is not true, and it performs unsat!. Hence x_{end} is true, and x_{error} is false because so is x_{wasTrue}. Dually, in case ψ is satisfiable, the agent can clearly infer $\widehat{\mathbf{K}}(\varphi_{\mathrm{isTrue}})$. Then the agent sets the variables $x_1, \dots x_n$ so as to satisfy ψ. Indeed, at each step, if $\widehat{\mathbf{K}}(x_i \wedge \varphi_{\mathrm{isTrue}})$ is true, this means that it is possible to make ψ true by extending the current partial assignment of $x_1, \dots, x_{i-1}$ with $x_i = \top$; otherwise, since ψ is satisfiable, this is possible with $x_i = \bot$. Hence the agent ends by executing sat!, so that x_{end} becomes true and, as $\varphi_{\mathrm{isTrue}}$ is now true, x_{error} remains false. Hence in all cases, the last state of r satisfies the goal formula φ^G. □

Proposition 27. *Assuming* NP $\not\subseteq$ P/poly*, for any reactive class $\mathcal{R}$, there is no family $(\pi_n)_{n\in\mathbb{N}}$ such that π_n is valid for Π_n and has a poly(n)-size representation in $\mathcal{R}$.*

Proof. Towards contradiction, assume there is such a family $(\pi_n)_{n\in\mathbb{N}}$. We construct a polynomial-time algorithm for 3-SAT which uses $(\pi_n)_{n\in\mathbb{N}}$ as the advice sequence. The algorithm takes a 3CNF formula ψ over n vars as well as (π_n) as an input, and executes the following steps:

1. set $x_1, \dots, x_n$ to be the list of variables appearing in ψ;
2. build an assignment s^0 to X_n with arbitrary values for $x_1, \dots, x_n$, values for $x_{\gamma\in\psi}$'s corresponding to ψ, and values for the other variables as determined by φ_n^I;
3. compute a π_n-maximal run r starting in s^0 by simulating π_n;
4. accept if sat! is taken in some state s^t of r; reject otherwise.

Since the goal requires any valid policy to stop in a polynomial number of steps ($8\binom{n}{3} + n + 1$) and π_n is reactive, computing r requires only polynomial time. Therefore our algorithm runs in polynomial time. Let us prove that it is correct.

If it accepts its input $\langle \psi, \pi_n \rangle$, then action sat! was taken in some state s^t of r. Thus, since x_{error} is false in the last state of r (because π_n achieves the goal), s^t must satisfy $\varphi_{\mathrm{isTrue}}$. Hence the assignment s^t restricted to $x_1, \dots, x_n$ makes ψ true. Hence ψ is satisfiable.

Conversely, if the algorithm rejects its input $\langle \psi, \pi_n \rangle$, then π_n has taken action unsat! in some state s^t (because the goal requires x_{end} to be true). Towards contradiction, assume that ψ has a satisfying assignment μ, and let s^0_μ be a state equal to s^0 except that $x_1, \dots, x_n$ are assigned as in μ and that x_{wasTrue} is true. Clearly, s^0_μ satisfies φ_n^I. Now since the values of the x_i's and x_{wasTrue} have no influence on the observations received by the agent, π_n would take exactly the same actions in a run starting in s^0_μ as in r. In particular, it would take

action unsat! at some point, despite x_{wasTrue} being true, hence making x_{error} true, which contradicts the validity of π_n. Hence ψ is unsatisfiable, as desired.

Hence we have a polynomial-time algorithm for 3-SAT using π_n as an advice. Since π_n has size polynomial in n, it follows that 3-SAT is in $\mathsf{P/poly}$, which contradicts $\mathsf{NP} \not\subseteq \mathsf{P/poly}$ since 3-SAT is NP-complete [43]. □

Intuitively, what Proposition 27 says is that any reactive policy for Π_n would necessarily have an exponential number of branches or paths, or would violate the "deadline" $8\binom{n}{3}+n+1$ of the problem. About this latter point, observe that we could set any deadline polynomial in n (and greater than $8\binom{n}{3}+n$) and have the same results, so that any reactive policy would in fact have an exponential number of branches or paths, *or take an exponential number of actions in some runs*. For instance, an algorithm which explores all possible assignments to the variables in order to find a satisfying one for ψ could be written in a succinct and reactive form (essentially, the DPLL algorithm would be suitable), but it would obviously take an exponential number of actions in general.

Now as concerns representations of policies as maps from belief states to actions, it is also easy to see that they cannot be of polynomial size for the family $(\Pi_n)_{n\in\mathbb{N}}$. Indeed, any valid policy must clearly sense all clauses at some point, and then any two different 3CNF formulas would lead it to a different belief state.

Proposition 28. *There is no family $(\pi_n)_{n\in\mathbb{N}}$ such that π_n is valid for Π_n and has a representation as a map from belief states to actions with poly(n) belief states.*

As another example of a family of policies which is succinct when represented with epistemic branching conditions, but which cannot be succinct when these are not allowed (even with implicit representations of belief states), consider a policy expressing "if the agent knows the value of an even number of variables, then take action a_1, else take action a_2". In the language of KBPs, this can be expressed by the succinct condition

$$[\textbf{if } (\mathbf{K}x_1 \vee \mathbf{K}\neg x_1) \leftrightarrow (\mathbf{K}x_2 \vee \mathbf{K}\neg x_2) \leftrightarrow \cdots \leftrightarrow (\mathbf{K}x_n \vee \mathbf{K}\neg x_n) \textbf{ then } a_1 \textbf{ else } a_2 \textbf{ fi}]$$

while clearly this cannot be expressed succinctly if general epistemic formulas are not allowed.

Finally, as a concrete example of a policy representation which is much less compact (and much less readable) than an equivalent KBP, Figure 9 shows a reactive policy which has exactly the same behaviour as the KBP κ_{ms} of our running example, starting in the situation s_1 depicted on Figure 3. In fact, we can show that Minesweeper essentially exhibits the same behaviour as 3-SAT does: given an instance of Minesweeper, there is a polysize KBP which "says" whether there is a safe position to click (obtained by slightly modifying κ_{ms}), while if there was an equivalent, polysize reactive policy, we would get $\mathsf{coNP} \subseteq \mathsf{P/poly}$ (which is equivalent to $\mathsf{NP} \subseteq \mathsf{P/poly}$) from the fact that this question is a coNP-complete problem [67]. This gives an example of a natural

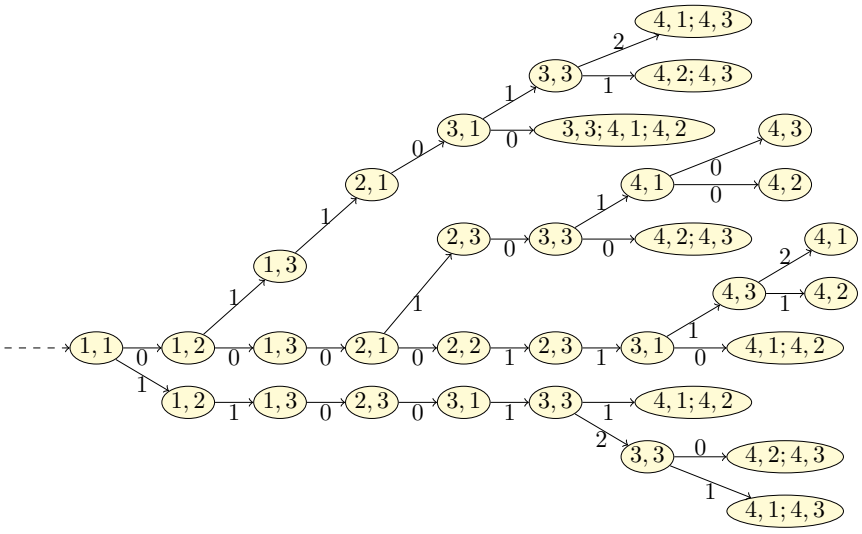

Figure 9: A reactive policy for Minesweeper equivalent to κ_{ms} starting in state s_1 of Figure 3. The nodes are labelled with the positions where to click, and the edges with the uncovered numbers.

problem for which KBPs are exponentially more succinct than *any* reactive class of representations.[13]

6. Complexity of Execution

Informally, the execution problem consists of determining the next action to perform when executing a KBP.

Definition 29 (execution problem for KBPs)**.** *The* execution problem *is the following decision problem.*

- **Input:** *a partially observable domain M, a factored initial belief state φ^I, a KBP κ (inducing the policy $\pi = \pi_{\kappa,\text{Sat}(\varphi^I)}$), a finite π-consistent history h, and an action a.*
- **Output:** *"Yes" if $\pi(h) = a$ holds, that is, a is the action prescribed by κ, starting in φ^I, after the history h; "No" otherwise.*

6.1. Execution as Belief Tracking

Recall that (online) belief tracking is the problem of deciding, given an initial belief state and a history so far, whether a formula (typically, the precondition of

[13]The proof is a little more involved than the one with 3-SAT, but follows essentially the same structure.

an action or the goal) holds in the current belief state [20]; in formulas, whether $\mathrm{Prog}(I, h) \models \mathbf{K}\varphi$ holds for given I, h, φ.

It follows that the execution problem for KBPs is essentially one of belief tracking. However, an important difference is that for KBPs, solving a single execution problem requires to decide a number of atomic conditions of the form $\mathbf{K}\varphi$ in the general case. For instance, in our Minesweeper example, if the kth position is the first one which is safe to click, then the agent needs to evaluate k formulas of the form $\mathbf{K}\neg m_{,i,j}$, in addition to the condition of the **while** loop, before finding the next action to execute. For this reason, the execution problem for KBPs is harder than online belief tracking, namely, Θ^2_{P}-complete (Proposition 32), and this holds even for an empty history.

This being said, it is clear that executing a KBP amounts to solving a number of online belief tracking problems, with the atoms $\mathbf{K}\varphi$ of branching and continuation conditions as queries.[14] Moreover, from membership in Θ^2_{P} (Proposition 30) we get that these problems can be solved in parallel for one instance of the execution problem.

6.2. Complexity Results

Recall that Θ^2_{P} is the class of problems decided by a polynomial-time algorithm that can make *independent* queries to an NP oracle, or, equivalently, a logarithmic number of queries to an NP oracle [38, 24].

Proposition 30. *The execution problem for KBPs is in* Θ^2_{P}.

Proof. We write I for $\mathrm{Sat}(\varphi^I)$, we denote by t the length of h, and by $h^{<u}$ the prefix of h of length u. The algorithm follows the following steps:

1. decide the *independent* questions $\mathrm{Prog}(I, h^{<u}) \models \mathbf{K}\varphi$ for all timesteps $u \leq t$ and for all *atomic* epistemic formulas $\mathbf{K}\varphi$ appearing in all branching conditions of κ,[15]
2. infer the results of the queries $\mathrm{Prog}(I, h^{<u}) \models \Phi$ for all epistemic formulas Φ occurring as a branching condition in κ,
3. execute the KBP κ until timestep t by using the answers to the queries, and deduce whether a is to be executed.

The questions in the first step are in fact online belief tracking queries. The answer to such a query $\mathrm{Prog}(I, h^{<u}) \models \mathbf{K}\varphi$ is negative if and only if there is a sequence of states $s_0, s_1, \ldots, s_u$, with $s_0 \in I$, which is consistent with M and with $h^{<u}$, but is such that s_u does not satisfy φ. For a given sequence, these

[14] Observe that we assume the history given in input to be π-consistent, so that we do *not* need to check that the preconditions of actions held all along the history, as required by *offline* belief tracking [18].

[15] The algorithm needs to evaluate *all* conditions because we formulated the execution problem without assuming anything about what information the agent maintains, so that it must first recover what branch of its KBP it is currently executing; informally, it must "replay" its KBP. However, this has no impact on the complexity of execution, since our hardness result (Proposition 32) already holds for a KBP with only one branching condition.

conditions can clearly be checked in polynomial time, hence each query is a coNP question. Now each query in the second item can be answered in polynomial time given the previous answers, since each Φ is a Boolean combination of atomic epistemic formulas (after rewriting $\widehat{\mathbf{K}}\varphi$ into $\neg\mathbf{K}\neg\varphi$), and similarly, the third step can be performed in polynomial time.

Hence the algorithm runs in polynomial time with independent queries to a coNP oracle or, equivalently, to an NP oracle. It follows that the execution problem is in Θ^2_{P}. □

Hence, as the proof makes clear, one can resort to online belief tracking algorithms for executing KBPs, using one of the numerous techniques developed in the literature: in particular, explicitly maintaining the belief state [70], resorting to an NP oracle [33, 40, 18], using regression [20]. Further, when belief tracking is tractable, this means that all questions in the first step of the proof of Proposition 30 can be answered in polynomial time.

Proposition 31. *Let $\mathcal{C}$ be a class of partially observable models for which the online belief tracking problem is in P. Then the execution problem for KBPs, restricted to models M in $\mathcal{C}$, is in P.*

Hence one can take advantage of special cases identified in the literature, like bounded causal width [18] or contextual width [20].

For the general case however, we now show that the problem is Θ^2_{P}-hard.

Proposition 32. *The execution problem for KBPs is Θ^2_{P}-hard. Hardness holds even for while-free KBPs, the empty history, and the initial belief state $\varphi^I = \top$.*

Proof. We prove hardness by a reduction from the following problem, known to be Θ^2_{P}-complete [72, Theorem 3.2]:[16] given k propositional formulas $\varphi_1, \ldots, \varphi_k$ such that $\mathrm{Sat}(\varphi_i) \subseteq \mathrm{Sat}(\varphi_{i+1})$ for all i, is the smallest j such that φ_j is satisfiable an odd number?

We assume without loss of generality that k is even (otherwise we add $\varphi_{k+1} = \top$). We define the partially observable domain $M = \langle X, A, O\rangle$, with X being the set of variables occurring in the formulas $\varphi_1, \ldots, \varphi_k$ and A being $\{a_{\mathrm{even}}, a_{\mathrm{odd}}\}$ (the set O of observations and the description of the actions are irrelevant). Let us define the following KBP κ:

$$[\textbf{if } (\mathbf{K}\neg\varphi_1 \wedge \widehat{\mathbf{K}}\varphi_2) \vee (\mathbf{K}\neg\varphi_3 \wedge \widehat{\mathbf{K}}\varphi_4) \vee \ldots \vee (\mathbf{K}\neg\varphi_{k-1} \wedge \widehat{\mathbf{K}}\varphi_k) \textbf{ then } a_{\mathrm{even}} \textbf{ else } a_{\mathrm{odd}} \textbf{ fi}]$$

The reduction computes the instance $\langle M, \varphi^I, \kappa, h, a\rangle$ where M and κ are defined above, the initial belief φ^I is $\top$, the history h is the empty history ϵ, and the action a is a_{odd}. The whole construction is polynomial, and it is easy to see that a_{odd} is (the first action) executed if and only if the smallest j such that φ_j is satisfiable is an odd number. □

[16] We thank Ronald de Haan for discussions about this result.

7. Complexity of Verification

We now turn to the problem of verifying that a knowledge-based program is valid for the planning problem which it intends to solve. We recall that validity means that the program terminates and that each of its possible executions respects the precondition of actions and reaches the goal (then stops).

Definition 33 (verification problem for KBPs)**.** *The* verification problem *for KBPs is the following decision problem.*

- **Input:** *a contingent planning instance* $\Pi = \langle M, \varphi^I, \varphi^G \rangle$ *and a KBP* κ
- **Output:** *"Yes" if* κ *is valid for* Π*; "No" otherwise.*

We start by showing membership to EXPSPACE, and then we prove EXPSPACE-hardness. Inbetween, we construct a KBP whose unique execution path has a doubly exponential length, and which will be used as a clock for the hardness proof. Finally, we show that the restriction to while-free KBPs makes the complexity of verification fall down to the second level of the polynomial hierarchy.

7.1. Verifying KBPs is in EXPSPACE

Proposition 34. *The verification problem for KBPs is in* EXPSPACE*.*

Proof. Let $\langle \Pi, \kappa \rangle$ be an instance of the verification problem. We design a nondeterministic algorithm that decides that κ is *not* valid for Π. Write π for $\pi_{\kappa,\mathrm{Sat}(\varphi^I)}$, the policy induced by κ starting in the initial belief state.

The algorithm iteratively guesses, timestep per timestep, the elements of a run r, and checks that r is indeed a run for M and that it is π-consistent. If at some point an action is taken while its precondition is not true, then the algorithm accepts its input (it has found a—prefix of a—π-maximal run which is not M-safe). Moreover, meanwhile, at each new timestep it increments a counter. If the counter goes beyond a theshold $2^{2^{poly(n)}}$ (precisely, 2^{2^n} times the number of control points in κ, given that there are only 2^{2^n} different belief states), the algorithm again accepts its input (the execution is necessarily in some previously met configuration, and hence π can get stuck in a nonterminating cycle). Otherwise, the run terminates, and the algorithm accepts its input if and only if the final state *does not* satisfy the goal.

This algorithm runs in exponential space because at each timestep, the only information which must be maintained consists of the current state, action, observation, the current belief state (which has exponential size when represented as a set of states), and the counter (which requires only $2^{poly(n)}$ bits). Thus, the verification problem is in coNEXPSPACE. Now by Savitch's theorem [65], we have NEXPSPACE = EXPSPACE, and since EXPSPACE is deterministic, we have coNEXPSPACE = coEXPSPACE = EXPSPACE. □

7.2. A Very Slow KBP

We show how to build a polysize KBP that terminates after a doubly exponential number of steps. This KBP is used as a *clock* in the proof of the EXPSPACE-hardness of KBP verification (Proposition 37).

This construction is of independent interest. Indeed, as it turns out, the KBP which we build uses only *purely ontic* actions. As a consequence, it evolves in a nonobservable environment (with nondeterministic actions), which is the setting of *conformant planning* [2, 50]. Since there are no observations, there is only one possible execution, so that we can restrict to policies which are equivalent to sequences of actions. Since the KBP which we build has a doubly exponential long trace, it is in fact equivalent to a sequence of actions of doubly exponential length. Still, using branching on epistemic conditions, we are able to encode this sequence into a KBP of polynomial size. This can be seen as using epistemic conditions for representing in a very compact form the current timestep in the sequence of actions. We believe that such use of branching for representing sequential policies very compactly has been overlooked in the literature on classical and conformant planning (with the exception of Bäckström et al. [8]).

Overview of the Construction. We write $<$ for the lexicographic order on states. For instance, $\mathcal{P}(\{x_1, x_2, x_3\})$ is ordered by $\overline{x_1x_2x_3} < \overline{x_1x_2}x_3 < \cdots < x_1x_2x_3$. Given a belief state B over a set of variables X and $Y \subseteq X$, we write $B_{|Y}$ for $\{s_{|Y} \mid s \in B\}$, where $s_{|Y}$ denotes the restriction of s to the variables in Y. This allows us to talk about the beliefs of the agent about the variables in Y.

The idea of our construction is to build a partially observable domain M_n, an initial belief state $\varphi_n^{I,\mathrm{clock}}$, and a KBP $\kappa_n^{\mathrm{clock}}$, in such a way that, informally, after t steps of execution, the current belief state is the tth one in a certain enumeration of all belief states. For this, the actions taken by $\kappa_n^{\mathrm{clock}}$ at each execution of the loop will either remove a state from the previous belief state (using a deterministic action) or add one (using a nondeterministic action).

Domain. Precisely, we first build a domain $M = \langle X_n, A_n, O_n\rangle$ (with $O_n = \{o_{\mathrm{void}}\}$). We define X_n to be the set of $4n{+}1$ variables $\{x_i, x_i^{\mathrm{a}}, x_i^{\mathrm{r}}, x_i^{\mathrm{g}} \mid i = 1, \ldots, n\} \cup \{x^{\mathrm{odd}}\}$, and we write $Y = \{x_i \mid i = 1, \ldots, n\}$.[17] For a given belief state B over X_n, we view $B_{|Y}$ as a vector $\vec{b} = b_1b_2 \ldots b_{2^n-1}b_{2^n}$ of 2^n bits, with $b_i = 1$ if and only if the ith state s_i (in the order $<$) is in B. Then our KBP starts with $\vec{b}^0 = 00\ldots01$ (*i.e.*, $B^0 = \{11\ldots1\}$ or, informally, the initial belief state $\mathbf{K}(x_1 \wedge \cdots \wedge x_n)$), and loops until $\vec{b}^{2^{2^n}-1} = 10\ldots00$, or, equivalently, until the current belief state satisfies $\mathbf{K}(\neg x_1 \wedge \cdots \wedge \neg x_n)$. The loop changes the current $\vec{b}^t$ to its successor $\vec{b}^{t+1}$ according to the *Gray code*, which is a way to enumerate all Boolean vectors by changing exactly one bit at a time.

[17] The mnemonics are: x_i^{a} (resp. x_i^{r}, x_i^{g}) takes the value of x_i in the state to be *a*dded to B (resp. in the state to be *r*emoved from B, in the *g*reatest state of B).

Definition 35 (Gray Code)**.** *The successor of a Boolean vector* $\vec{b}$ *according to the* Gray Code *is the Boolean vector obtained from* $\vec{b}$ *as follows:*

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

For instance, the enumeration is $0001, 0011, 0010, 0110 \ldots 1000$ for $n = 2$ (we do not use 0000). In terms of belief states, this is the enumeration

$$\{x_1x_2\}, \{x_1\overline{x_2}, x_1x_2\}, \{x_1\overline{x_2}\}, \{\overline{x_1}x_2, x_1\overline{x_2}\}, \ldots, \{\overline{x_1x_2}\},$$

which indeed passes through all belief states.

Observe that by definition of $\vec{b}$, the greatest i with $b_i = 1$ identifies the greatest state in B (in the order $<$), and flipping b_i amounts to add/remove s_i to/from B.

We now define the set of actions A_n to be $\{x_i^c{:=}\top, x_i^c{:=}\bot \mid i = 1, \ldots, n, c = \mathrm{a}, \mathrm{r}, \mathrm{g}\} \cup \{x_i^{\mathrm{a}}{:=}x_i^{\mathrm{g}}, x_i^{\mathrm{r}}{:=}x_i^{\mathrm{g}} \mid i = 1, \ldots, n\} \cup \{a^{\mathrm{add}}, a^{\mathrm{rem}}, a^{\mathrm{odd}}\}$. Action $x_i^{\mathrm{a}}{:=}\top$ deterministically sets x_i^{a} to $\top$, and similarly for other actions $x_i^c{:=}v$. Action $x_i^{\mathrm{a}}{:=}x_i^{\mathrm{g}}$ (resp. $x_i^{\mathrm{r}}{:=}x_i^{\mathrm{g}}$) deterministically sets x_i^{a} (resp. x_i^{r}) to the current value of x_i^{g}. Action a^{odd} switches the value of x^{odd}.

Now action a^{add} is a simple nondeterministic action, which either does nothing or sets $x_1, \ldots, x_n$ to the values of $x_1^{\mathrm{a}}, \ldots, x_n^{\mathrm{a}}$:

$$a^{\mathrm{add}} = \{a_1^{\mathrm{add}}, a_2^{\mathrm{add}}\} \text{ with } \forall i, \mathit{cond}_{a_2^{\mathrm{add}}, x_i} = x_i^{\mathrm{a}} \text{ and } \mathit{cond}_{a_2^{\mathrm{add}}, \bar{x}_i} = \bar{x}_i^{\mathrm{a}}.$$

It is easy to realize that when the agent takes this action in a belief state B in which the x_i^{a}'s are for sure assigned values $v_1, \ldots, v_n$, the progressed belief state B' satisfies $B'_{|Y} = B_{|Y} \cup \{v_1 \ldots v_n\}$.

Finally, action a^{rem} is a simple deterministic action, which leaves all variables unchanged, except if each x_i is assigned the same value as x_i^{r}, in which case it sets $x_1, \ldots, x_n$ to the values of $x_1^{\mathrm{g}}, \ldots, x_n^{\mathrm{g}}$:

$$\forall i, \mathit{cond}_{x_i} = \big(\bigwedge_{i=1}^{n}(x_i \leftrightarrow x_i^{\mathrm{r}})\big) \wedge x_i^{\mathrm{g}} \text{ and } \mathit{cond}_{\bar{x}_i} = \big(\bigwedge_{i=1}^{n}(x_i \leftrightarrow x_i^{\mathrm{r}})\big) \wedge \neg x_i^{\mathrm{g}}$$

By construction, after taking this action in a belief state B in which the x_i^{r}'s (resp. x_i^{g}'s) are for sure assigned $v_1, \ldots, v_n$ (resp. $v_1^{\mathrm{g}}, \ldots, v_n^{\mathrm{g}}$), if $(v_1, \ldots, v_n) \neq (v_1^{\mathrm{g}}, \ldots, v_n^{\mathrm{g}})$ and $v_1^{\mathrm{g}} \ldots v_n^{\mathrm{g}} \in B_{|Y}$ hold, then the resulting belief state B' satisfies $B'_{|Y} = B_{|Y} \setminus \{v_1 \ldots v_n\}$.

Knowledge-Based Program. Before defining the knowledge-based program $\kappa_n^{\mathrm{clock}}$, we define three subprograms. The first subprogram is written κ^{g}. When κ^{g} is run starting in a belief state B, it ends up assigning to $x_1^{\mathrm{g}}, \ldots, x_n^{\mathrm{g}}$ the values $v_1, \ldots, v_n$ such that $v_1 \ldots v_n$ is the greatest assignment (in the order $<$) in $B_{|Y}$; in words, it copies the greatest *possible* assignment of the x_i's to the x_i^{g}'s. The idea is simply to use a dichotomic search among the assignments to Y:

if $\mathbf{K}(\neg x_1)$ **then** $x_1^{\mathrm{g}}{:=}\bot$ **else** $x_1^{\mathrm{g}}{:=}\top$ **fi;**

if $\mathbf{K}\big((x_1 \leftrightarrow x_1^{\mathrm{g}}) \rightarrow \neg x_2\big)$ **then** $x_2^{\mathrm{g}}{:=}\bot$ **else** $x_2^{\mathrm{g}}{:=}\top$ **fi;**

...

if $\mathbf{K}\Big(\big(\bigwedge_{i=1}^{n-1}(x_i \leftrightarrow x_i^{\mathrm{g}})\big) \rightarrow \neg x_n\Big)$ **then** $x_n^{\mathrm{g}}{:=}\bot$ **else** $x_n^{\mathrm{g}}{:=}\top$ **fi**

The second subprogram is written κ^{r}. When κ^{r} is run starting in a belief state B in which the x_i^{r}'s are assigned values $v_1, \ldots, v_n$ for sure, it ends up in a belief state B' satisfying $B'_{|Y} = B_{|Y} \setminus \{v_1 \ldots v_n\}$. To do so, κ^{r} first ensures that the x_i^{r}'s (resp. x_i^{g}'s) are for sure assigned values $v_1, \ldots, v_n$ (resp. $v_1^{\mathrm{g}}, \ldots, v_n^{\mathrm{g}}$) and $(v_1, \ldots, v_n) \neq (v_1^{\mathrm{g}}, \ldots, v_n^{\mathrm{g}})$ holds, as required for action a^{rem} to serve its purpose. For this, it first assigns to the x_i^{g}'s the values of the greatest assignment to Y in B, by running κ^{g},[18] then, if it turns out that this is the same assignment as that of the x_i^{r}'s ($\mathbf{K}\bigwedge_{i=1}^{n}(x_i^{\mathrm{g}} \leftrightarrow x_i^{\mathrm{r}})$), it runs the dual program of κ^{g} (which considers the smallest instead of the greatest assignment). Then it runs a^{rem}. Obviously, the preprocessing ensures that the x_i^{r}'s and the x_i^{g}'s have different values only if there are at least two assignments in B, so that the greatest and the smallest ones are different; we will ensure this when using κ^{r}.

Finally, the third subprogram is written κ^{d}. When κ^{d} is run starting in a belief state B in which the x_i^{g}'s are assigned values $v_1, \ldots, v_n$ for sure, it ends up with the same belief state, except that at all states the assignment to the x_i^{g}'s has been decremented by 1 (in the order $<$):

if $\mathbf{K}x_n^{\mathrm{g}}$ **then** $x_n^{\mathrm{g}}{:=}\bot$

else if $\mathbf{K}x_{n-1}^{\mathrm{g}}$ **then** $x_{n-1}^{\mathrm{g}}{:=}\bot$ **;** $x_n^{\mathrm{g}}{:=}\top$

...

else if $\mathbf{K}x_1^{\mathrm{g}}$ **then** $x_1^{\mathrm{g}}{:=}\bot$ **;** $x_2^{\mathrm{g}}{:=}\top$ **;** ... **;** $x_n^{\mathrm{g}}{:=}\top$

fi

Importantly, observe that M and $\kappa^{\mathrm{g}}, \kappa^{\mathrm{r}}, \kappa^{\mathrm{d}}$ all have a description of size at most quadratic in n.

With this in hand, we define the KBP $\kappa_n^{\mathrm{clock}}$ to be the KBP depicted on Figure 10.

Proposition 36. *Let $\varphi_n^{I,clock}$ be the formula $x_1 \wedge \cdots \wedge x_n \wedge x^{odd}$. The unique maximal run for κ_n^{clock} starting in $\varphi_n^{I,clock}$ has length $2^{2^n} - 1$.*

Proof. All actions used in $\kappa_n^{\mathrm{clock}}$ are purely ontic, therefore there is a unique maximal run starting in $\varphi_n^{I,\mathrm{clock}}$. Now, each step of the execution of $\kappa_n^{\mathrm{clock}}$ simulates the computation of the successor of the current $\vec{b}$ (representing the current belief state as projected onto Y) according to the Gray code. Since the initial belief state is by definition the set of satisfying assignments of $\varphi_n^{I,\mathrm{clock}}$,

[18] The choice of the *greatest* possible assignment is arbitrary.

while $\neg\mathbf{K}(\neg x_1 \wedge \cdots \wedge \neg x_n)$ **do**
  if $\mathbf{K}\neg x^{\mathrm{odd}}$ **do**
    /* even number of 1's, flip b_{2^n} */
    if $\mathbf{K}(\neg x_1 \vee \cdots \vee \neg x_n)$ **do**
      /* $11\ldots1 \notin B$: add it */
      $x_1^{\mathrm{a}}{:=}\top$; $x_1^{\mathrm{a}}{:=}\top$; $\ldots$; $x_n^{\mathrm{a}}{:=}\top$; a^{add}
    else
      /* $11\ldots1 \in B$: remove it */
      $x_1^{\mathrm{r}}{:=}\top$; $x_2^{\mathrm{r}}{:=}\top$; $\ldots$; $x_n^{\mathrm{r}}{:=}\top$; κ^{r}
    fi
  else
    /* odd number of 1's, flip b_{g-1} */
    κ^{g} ; κ^{d} ;
    if $\mathbf{K}\big((x_1 \not\leftrightarrow x_1^{\mathrm{g}}) \vee \cdots \vee (x_n \not\leftrightarrow x_n^{\mathrm{g}})\big)$ **then**
      /* $s_{g-1} \notin B$: add it */
      $x_1^{\mathrm{a}}{:=}x_1^{\mathrm{g}}$; $x_2^{\mathrm{a}}{:=}x_2^{\mathrm{g}}$; $\ldots$; $x_n^{\mathrm{a}}{:=}x_n^{\mathrm{g}}$; a^{add}
    else
      /* $s_{g-1} \in B$: remove it */
      $x_1^{\mathrm{r}}{:=}x_1^{\mathrm{g}}$; $x_2^{\mathrm{r}}{:=}x_2^{\mathrm{g}}$; $\ldots$; $x_n^{\mathrm{r}}{:=}x_n^{\mathrm{g}}$; κ^{r}
    fi
  fi;
  a^{odd}
od

Figure 10: The KBP $\kappa_n^{\mathrm{clock}}$.

corresponding to $\vec{b} = 00\ldots01$, and the KBP stops when its current belief state satisfies $\mathbf{K}(\neg x_1 \wedge \cdots \wedge \neg x_n)$, corresponding to $\vec{b} = 10\ldots00$, all belief states except $\emptyset$ (corresponding to $\vec{b} = 00\ldots00$) are reached exactly once. Therefore the length of the execution path is $2^{2^n} - 1$, as desired. □

7.3. Verifying KBPs is EXPSPACE*-Hard*

We now show that verifying KBPs is EXPSPACE-hard. The proof is based on a reduction from the plan existence problem in unobservable planning (NUP). In our terms, an instance of NUP is a triple $\Pi = \langle M, \varphi^I, \varphi^G \rangle$, where M has only one (void) observation, yielded by all actions in all states; the question is whether there exists a valid policy. The NUP problem was proven to be EXPSPACE-hard by Haslum and Jonsson [37]; another proof was given later by Rintanen [62].[19]

The main difference between an instance of NUP and an instance of KBP verification is that the latter also includes a KBP κ. The key idea of the reduction is to build a KBP which explores all possible plans for an instance of NUP, and which is valid if and only if *none* of them reaches the goal.

Proposition 37. *The verification problem for KBPs is* EXPSPACE*-hard. Hardness holds even for KBPs which are known to terminate.*

Proof. Let $\Pi_1 = \langle M_1, \varphi_1^I, \varphi_1^G \rangle$, with $M_1 = \langle X_1, A_1, O_1 \rangle$, be an instance of NUP. We define a partially observable domain $M_2 = \langle X_2, A_2, O_2 \rangle$, a contingent planning instance $\Pi_2 = \langle M_2, \varphi_2^I, \varphi_2^G \rangle$, and a KBP κ such that there is a valid policy for Π_1 if and only if κ is not valid for Π_2. Since the whole construction is feasible in polynomial time and coEXPSPACE is the same as EXPSPACE, this is enough to conclude.

The idea of κ is essentially to simulate a nondeterministic search for a valid plan for Π_1. By construction, valid plans, and only them, will induce runs on which the KBP will end up fasifying the goal.

Let $n = |X_1|$, and $X_n^{\text{clock}}, A_n^{\text{clock}}$ be the components of the domain defined in Section 7.2. We define the set of variables X_2 to be $X_1 \cup \{x_{\text{error}}, x_{\text{end}}\} \cup X_n^{\text{clock}}$, assuming without loss of generality that the three sets are disjoint. The set of actions A_2 is defined to be $\{a^{\text{choose}}, a^{\text{lose}}, a^G\} \cup A_n^{\text{clock}}$, where a^{choose} is a fresh action which nondeterministically executes one of the actions in A_1, and yields an observation which reveals which one it has picked:[20]

$$a^{\text{choose}} = \{a' \mid a \in A_1\} \text{ with } \forall a \in A_1, \mathit{eff}_{a'} = \mathit{eff}_a \text{ and } \mathit{when}_{a',o_a} = \top,$$

and a^{lose} (resp. a^G) is an action which sets x_{error} (resp. x_{end}) to $\top$ and yields a void observation o_{void}. Accordingly, we define $O_2 = \{o_a \mid a \in A_1\} \cup \{o_{\text{void}}\}$.

[19] The syntax of actions is slightly different from ours, but examination of the proof by Rintanen [62, Theorem 13] shows that this does not change the result

[20] We assume without loss of generality that the actions in A_1 all have a tautological precondition; otherwise, we replace the precondition pre_a with $\top$, we add an effect of the form $\mathit{cond}_{x_{\text{unsafe}}} = \mathit{pre}_a$, and we add $\neg x_{\text{unsafe}}$ to the goal.

Finally, we define φ_2^I to be $\varphi_1^I \wedge \neg x_{\text{error}} \wedge \neg x_{\text{end}} \wedge \varphi_n^{I,\text{clock}}$, where $\varphi_n^{I,\text{clock}}$ is defined in Proposition 36, φ_2^G to be $\neg x_{\text{error}} \wedge x_{\text{end}}$, and the KBP κ to be

while $\neg\mathbf{K}\varphi_1^G \wedge \neg\mathbf{K}\varphi_n^{\text{beep}}$ **do**

a^{choose} **;** $\kappa_n^{\text{clock,inner}}$

od;

if $\mathbf{K}\varphi_1^G$ **then** a^{lose} **else** a^G **fi**

where $\neg\mathbf{K}\varphi_n^{\text{beep}} = \neg\mathbf{K}(\neg x_1 \wedge \cdots \neg x_n)$ is the continuation condition of κ_n^{clock}, and $\kappa_n^{\text{clock,inner}}$ is the body of its **while** loop.

Let π be any policy for Π_1 which is a sequence of actions without any branching and of length at most $2^{2^n} - 2$. Let h_π be the history for M_2 in which after each tth execution of a^{choose}, the observation received is $o_{\pi(t)}$, where $\pi(t)$ is the tth action in π. Then it is easy to realize that at any timestep, the agent's belief state, as progressed by h_π and projected onto the variables in X_1, is the same as it would in the execution of π.

Now if Π_1 admits a valid policy, it admits one without any branching (because Π_1 has only purely ontic actions) and of length at most $2^{2^n} - 2$ [62].[21] Let π be such a policy; then before the clock beeps the agent's belief state, as progressed by h_π, satisfies $\mathbf{K}\varphi_1^G$ and hence, the agent exits the **while** loop and executes a^{lose}, so that κ is not valid for Π_2. Dually, if there is no valid policy for Π_1, then in all histories the agent exits the **while** loop after $2^{2^n} - 2$ rounds (because $\mathbf{K}\varphi_n^{\text{beep}}$ becomes true) without its belief state satisfying $\mathbf{K}\varphi_1^G$, so that it executes a^G, and finally κ is valid for Π_2. This completes the proof. □

7.4. Verifying While-Free KBPs

We now consider the case when the KBP is while-free.

Proposition 38. *The verification problem for while-free KBPs is Π_{P}^2-complete. Hardness holds even if the initial belief state is restricted to be $\top$ and all ontic actions to be deterministic.*

Proof. We first show membership in Π_{P}^2. First observe that for a while-free KBP, all consistent runs have length polynomial in its size. Hence to show that κ is *not* valid for a factored instance $\Pi = \langle M, \varphi^I, \varphi^G \rangle$, we can guess a run r of polynomial length and verify that:

1. its starting state satisfies φ^I;
2. it is indeed a run for M;
3. it is $\pi_{\kappa,\text{Sat}(\varphi^I)}$-consistent and maximal;
4. it is not M-safe, or its last state does not satisfy φ^G.

[21] This is because there are $2^{2^n} - 1$ different, nonempty belief states, and a run containing N actions induces $N + 1$ belief states; so a longer policy would necessarily induce a loop, which can be simplified.

Conditions 1, 2 and 4 can clearly be checked in polynomial time. Condition 3 requires to solve a polynomial number of execution problems; since the execution problem is in Θ^2_{P} (Proposition 30), each of these execution problems can be solved in polynomial time using a polynomial number of NP-oracles; therefore, Condition 3 can be checked in polynomial time using a polynomial number of NP-oracles. Hence the verification problem is in Π^2_{P}.

For hardness, let $\forall x_1^\forall \dots x_p^\forall \exists x_1^\exists \dots x_q^\exists \varphi$ be a QBF formula. Define a factored model in which there is a purely epistemic action a_i, for $i = 1, \dots, p$, which yields observation o_{x_i} or $o_{\neg x_i}$ depending on the value of $x_i^\forall$, and a purely ontic, deterministic action a^G which sets a variable x_{end} to $\top$.

Now define an instance with $\varphi^I = \top$ and $\varphi^G = x_{\text{end}}$, and finally, let κ be the KBP

$$[a_1 \,;\, a_2 \,;\, \dots ;\, a_p \,;\, \textbf{if } \widehat{\mathbf{K}}\varphi \textbf{ then } a^G \textbf{ else } \varepsilon \textbf{ fi}]$$

Clearly, this KBP is valid for the contingent planning instance if and only if for all initial states s^0, after reading the values of $x_1^\forall, \dots, x_p^\forall$ in s^0, the agent considers it possible that φ is true. This in turn is equivalent to $\forall x_1^\forall \dots x_p^\forall \exists x_1^\exists \dots x_q^\exists \varphi$ being valid, which completes the proof since deciding the validity of such a QBF is Π^2_{P}-complete [53]. □

Like for the execution problem (Proposition 31), polynomial cases of the online belief tracking problem can be leveraged for verification of while-free KBPs. Indeed, when queries $\text{Prog}(I, h) \models \mathbf{K}\varphi$ can be answered in polynomial time, it is easy to see that Step 3 of the proof of Proposition 38 can be performed in polynomial time by "replaying" the KBP.

Proposition 39. *Let $\mathcal{C}$ be a class of partially observable models for which the online belief tracking problem is in P. Then the verification problem for while-free KBPs, restricted to models M in $\mathcal{C}$, is in coNP.*

8. Extensions of the Model and the Language

We briefly discuss a few different extensions of our basic model and language, and for each of them, we discuss which of our results would continue to hold.

8.1. Epistemic Goals

Note that despite the fact that we investigate *knowledge-based* approaches to the representation of policies, we do not define *epistemic* goals: a goal is achieved at some timestep depending on the current state, *not* on the agent's current knowledge. The reason why we made this choice is that since we compare knowledge-based policies to other representations of policies for contingent planning, especially along succinctness, we need the models to coincide with the standard ones for contingent planning, which have *ontic* goals.

The difference is more conceptual than technical, though. Below we show that our results would hold as well for positive epistemic goals, expressed by *positive epistemic formulas*. Positive epistemic formulas are defined inductively as follows:

- $\mathbf{K}\varphi$ is a positive epistemic formula;
- if Φ and Ψ are positive epistemic formulas, then $\Phi \wedge \Psi$ and $\Phi \vee \Psi$ are positive epistemic formulas.

Examples of single-agent planning problems with epistemic goals are numerous. For instance, in the epistemic version of Minesweeper, the goal is to know the exact location of the mines; in a diagnosis problem, the goal is to know which components are faulty.

In single-agent settings, it makes little sense to allow epistemic goals that are not positive (while examples can of course be constructed, they have a rather artificial flavour).[22] Now, the plan verification problem for planning instances with positive epistemic goals can be polynomially reduced in a straightforward way to plan verification for ontic goals, and *vice versa*. For one direction, this is trivial: a KBP is valid for a planning problem whose goal is φ^G if it is valid for the corresponding planning problem whose goal is $\mathbf{K}\varphi^G$. For the reverse direction, consider a planning problem with a positive epistemic goal Φ^G. Then κ is valid for this planning problem if and only if the KBP $[\kappa$ **; if** Φ^G **then** $a^G]$ is valid for the goal x_{end}, where a^G is an action that sets x_{end} to true, and an initial state for which x_{end} is known to be false.

Therefore, all complexity results of Section 7 carry on to plan verification for planning problems with positive epistemic goals.

8.2. Different Action Languages

We chose one language for the compact representation of transition functions, namely that of Brafman and Shani [20], because it is simple, yet powerful enough. However, Nebel [57] has shown that most natural representations of actions are equivalent to each other in terms of expressivity and computation. We could have chosen another language from the literature, such as, for instance, expressing actions by propositional action theories (where the effects of an action are described by a formula involving variables typed by t and others by t', for representing the state of the world before and after the action). With such a language, as well as with any other language with polynomial-size translations to and from the language we chose here, all our succintness and complexity results continue to hold. In particular, it can be seen that our membership complexity results (Propositions 30, 34, 38) only use the fact that deciding whether a state is in the initial belief state, satifies the goal, or satisfies the precondition of an action, and whether a transition $s \xrightarrow{a|o} s'$ exists, can be done in polynomial time.

[22] This is completely different in environments with several agents, where an agent may have the goal to know a secret without another agent knowing it.

9. Conclusion and Future Work

We have revisited knowledge-based programs by placing them in the context of AI planning, which they had not been initially designed for. Though similar principles have been explored by the planning community, they had not been studied before as an explicit representation of policies. As such, KBPs turn out to have useful properties: they are succinct, generic (with respect to the initial belief state), and arguably easy to write and understand.[23] This comes with a computational price to pay when executing a knowledge-based program (and to a lesser extent, when verifying its validity for a given planning problem); whether this price should be paid or not depends of course of the specificities of the problem at hand.

Obviously, the first direction for future work is to develop and experiment algorithms for synthesizing (small) valid KBPs for a given planning problem. As preliminary results in this direction, we have settled the complexity of the associated decision problem (is there a — small — valid KBP?) under various restrictions [46]. Of course, in the general case, there exists a valid KBP if and only if there exists a plan at all, so that the problem is 2-EXPTIME-complete [37, 62]; the results by Lang and Zanuttini [46] show that, unsurprisingly, the complexity is much lower when small KBPs are sought for.

Back to the synthesis problem, a natural idea is to use the regression-based algorithm proposed by Herzig et al. [39]. However, this algorithm does not seem to scale up easily,[24] and synthesizing (small) KBPs remains a challenging issue. In some applicative settings, another option would be to first compute reactive, history-based policies using approaches from the literature, and then compact them into KBPs, using for instance online regression [20] along each branch.

An orthogonal direction for future work consists of considering richer settings. A first, natural extension is to probabilistic settings, like POMDPs [41]. First investigations in this direction were made by Belle and Levesque [10] and by Lang and Zanuttini [47]. A second important extension of the model is to collaborative planning, like for Dec-POMDPs [11] or their qualitative counterpart [22]. In such settings, the ability to reason about the other agents' knowledge at execution time is especially important, since the agents typically follow some predefined programs that are common knowledge, but make private observations. First investigations have been made by Saffidine et al. [64], and this direction is also obviously related to epistemic planning [13].

Acknowledgements. The authors wish to thank Alexandre Niveau and Anaëlle Wilczynski for many useful discussions about this work, and for their participation in experimental work related to it. This work was supported by Agence Nationale de la Recherche under the "programme d'investissements d'avenir"

[23] Of course, this claim is not formally provable. See however Example 2: the KBP is arguably very simple to understand and to explain, while a standard policy is arguably not.

[24] We made some preliminary experiments.

ANR-19-P3IA-0001 (PRAIRIE). We also warmly thank the reviewers who considerably helped us to improve our paper.

References

[1] Alexandre Albore, Héctor Palacios, and Hector Geffner. A translation-based approach to contingent planning. In *Proc. IJCAI 2009*, pages 1623–1628, 2009.

[2] Alexandre Albore, Héctor Palacios, and Hector Geffner. Compiling uncertainty away in non-deterministic conformant planning. In *Proc. ECAI 2010*, pages 465–470, 2010.

[3] Mikkel Birkegaard Andersen, Thomas Bolander, and Martin Holm Jensen. Conditional epistemic planning. In *Proc. JELIA 2012*, pages 94–106, 2012.

[4] Karl Johan Åström. Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10 (1):174–205, 1965.

[5] Guillaume Aucher. DEL-sequents for regression and epistemic planning. *Journal of Applied Non-Classical Logics*, 22(4):337–367, 2012.

[6] Guillaume Aucher and Thomas Bolander. Undecidability in epistemic planning. In *Proc. IJCAI 2013*, pages 27–33, 2013.

[7] Christer Bäckström and Peter Jonsson. Algorithms and limits for compact plan representations. *J. Artif. Intell. Res.*, 44:141–177, 2012.

[8] Christer Bäckström, Anders Jonsson, and Peter Jonsson. Automaton plans. *J. Artif. Intell. Res.*, 51:255–291, 2014.

[9] Jorge A. Baier and Sheila A. McIlraith. On planning with programs that sense. In *Proc. KR 2006*, pages 492–502, 2006.

[10] Vaishak Belle and Hector J. Levesque. ALLEGRO: belief-based programming in stochastic dynamical domains. In *Proc. IJCAI 2015*, pages 2762–2769, 2015.

[11] Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The complexity of decentralized control of Markov decision processes. In *Proc. UAI 2000*, pages 32–37, 2000.

[12] Piergiorgio Bertoli, Alessandro Cimatti, Marco Roveri, and Paolo Traverso. Strong planning under partial observability. *Artif. Intell.*, 170(4-5):337–384, 2006.

[13] Thomas Bolander and Mikkel Birkegaard Andersen. Epistemic planning for single and multi-agent systems. *Journal of Applied Non-Classical Logics*, 21(1):9–34, 2011.

[14] Thomas Bolander, Thorsten Engesser, Andreas Herzig, Robert Mattmüller, and Bernhard Nebel. The dynamic logic of policies and contingent planning. In *Proc. JELIA 2019*, pages 659–674, 2019.

[15] Blai Bonet and Hector Geffner. Planning with incomplete information as heuristic search in belief space. In *Proc. AIPS 2000*, pages 52–61, 2000.

[16] Blai Bonet and Hector Geffner. Planning under partial observability by classical replanning: Theory and experiments. In *Proc. IJCAI 2011*, pages 1936–1941, 2011.

[17] Blai Bonet and Hector Geffner. Flexible and scalable partially observable planning with linear translations. In *Proc. AAAI 2014*, pages 2235–2241, 2014.

[18] Blai Bonet and Hector Geffner. Belief tracking for planning with sensing: Width, complexity and approximations. *J. Artif. Intell. Res.*, 50:923–970, 2014.

[19] Ronen I. Brafman and Guy Shani. Replanning in domains with partial information and sensing actions. *J. Artif. Intell. Res.*, 45:565–600, 2012.

[20] Ronen I. Brafman and Guy Shani. Online belief tracking using regression for contingent planning. *Artif. Intell.*, 241:131–152, 2016.

[21] Ronen I. Brafman, Joseph .Y. Halpern, and Yoam Shoham. On the knowledge requirements of tasks. *Artif. Intell.*, 98(1–2):317–350, 1998.

[22] Ronen I. Brafman, Guy Shani, and Shlomo Zilberstein. Qualitative planning under partial observability in multi-agent domains. In *Proc. AAAI 2013*, pages 130–137, 2013.

[23] Daniel Bryce, William Cushing, and Subbarao Kambhampati. State agnostic planning graphs: deterministic, non-deterministic, and probabilistic planning. *Artif. Intell.*, 175(3-4):848–889, 2011.

[24] Samuel R. Buss and Louise Hay. On truth-table reducibility to SAT. *Inf. Comput.*, 91(1):86–102, 1991.

[25] Tristan Charrier, Sébastien Gamblin, Alexandre Niveau, and François Schwarzentruber. Hintikka's world: Scalable higher-order knowledge. In *Proc. IJCAI 2019*, pages 6494–6496, 2019.

[26] Alessandro Cimatti, Marco Pistore, Marco Roveri, and Paolo Traverso. Weak, strong, and strong cyclic planning via symbolic model checking. *Artif. Intell.*, 147(1–2):35–84, 2003.

[27] Jens Claßen and Malte Neuss. Knowledge-based programs with defaults in a modal situation calculus. In *Proc. ECAI 2016*, pages 1309–1317, 2016.

[28] Jörg Claßen and Gerhard Lakemeyer. Foundations for knowledge-based programs using ES. In *Proc. KR 2006*, pages 318–328, 2006.

[29] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *J. Artif. Intell. Res.*, 17:229–264, 2002.

[30] Giuseppe De Giacomo, Yves Lespérance, and Fabio Patrizi. Bounded situation calculus action theories. *Artif. Intell.*, 237:172–203, 2016.

[31] Giuseppe De Giacomo, Yves Lespérance, Fabio Patrizi, and Stavros Vassos. Progression and verification of situation calculus agents with bounded beliefs. *Studia Logica*, 104(4):705–739, 2016.

[32] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[33] Paolo Ferraris and Enrico Giunchiglia. Planning as satisfiability in nondeterministic domains. In *Proc. AAAI 2000*, pages 748–753, 2000.

[34] Hector Geffner and Blai Bonet. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2013.

[35] Malik Ghallab, Dana S. Nau, and Paolo Traverso. *Automated Planning and Acting*. Cambridge University Press, 2016.

[36] Yilan Gu and Mikhail Soutchanski. Decidable reasoning in a modified situation calculus. In *Proc. IJCAI 2007*, pages 1891–1897, 2007.

[37] Peter Haslum and Peter Jonsson. Some results on the complexity of planning with incomplete information. In *Proc. ECP 1999*, pages 308–318, 1999.

[38] Lane A. Hemachandra. The strong exponential hierarchy collapses. *J. Comput. Syst. Sci.*, 39(3):299–322, 1989.

[39] Andreas Herzig, Jérôme Lang, and Pierre Marquis. Action representation and partially observable planning using epistemic logic. In *Proc. IJCAI 2003*, pages 1067–1072, 2003.

[40] Jörg Hoffmann and Ronen I. Brafman. Contingent planning via heuristic forward search with implicit belief states. In *Proc. ICAPS 2005*, pages 71–80, 2005.

[41] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101:99–134, 1998.

[42] Lars Karlsson. Conditional progressive planning under uncertainty. In *Proc. IJCAI 2001*, pages 431–438, 2001.

[43] Richard M. Karp. Reducibility among combinatorial problems. In *Proc. Complexity of Computer Computations*, pages 85–103, 1972.

[44] Richard M. Karp and Richard J. Lipton. Some connections between nonuniform and uniform complexity classes. In *Proc. STOC 1980*, pages 302–309, 1980.

[45] Jérôme Lang and Bruno Zanuttini. Knowledge-based programs as plans – the complexity of plan verification. In *Proc. ECAI-2012*, pages 504–509, 2012.

[46] Jérôme Lang and Bruno Zanuttini. Knowledge-based programs as plans: Succinctness and the complexity of plan existence. In *Proc. TARK 2013*, 2013.

[47] Jérôme Lang and Bruno Zanuttini. Probabilistic belief-based programs. In *Proc. IJCAI-15*, pages 1594–1600, 2015.

[48] Noël Laverny and Jérôme Lang. From knowledge-based programs to graded belief-based programs, part I: On-line reasoning. *Synthese*, 147(2):277–321, 2005.

[49] Noël Laverny and Jérôme Lang. From knowledge-based programs to graded belief-based programs, part II: Off-line reasoning. In *Proc. IJCAI 2005*, pages 497–502, 2005.

[50] Yanjun Li, Quan Yu, and Yanjing Wang. More for free: a dynamic epistemic framework for conformant planning over transition systems. *J. Log. Comput.*, 27(8):2383–2410, 2017.

[51] Benedikt Löwe, Eric Pacuit, and Andreas Witzel. DEL planning and some tractable cases. In *Proc. LORI 2011*, pages 179–192, 2011.

[52] Shlomi Maliah, Ronen I. Brafman, Erez Karpas, and Guy Shani. Partially observable online contingent planning using landmark heuristics. In *Proc. ICAPS 2014*, 2014.

[53] Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proc. 13th Annual Symposium on Switching and Automata Theory*, pages 125–129, 1972.

[54] Robert C. Moore. *A Formal Theory of Knowledge and Action*. PN, 1985.

[55] Leora Morgenstern. Knowledge preconditions for actions and plans. In *Proc. IJCAI 1987*, pages 867–874, 1987.

[56] Christian J. Muise, Vaishak Belle, and Sheila A. McIlraith. Computing contingent plans via fully observable non-deterministic planning. In *Proc. AAAI 2014*, pages 2322–2329, 2014.

[57] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *J. Artif. Intell. Res.*, 12:271–315, 2000.

[58] Ronald Petrick and Fahiem Bacchus. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proc. ICAPS 2004*, pages 2–11, 2004.

[59] Joelle Pineau, Geoffrey J. Gordon, and Sebastian Thrun. Anytime point-based approximations for large POMDPs. *J. Artif. Intell. Res.*, 27:335–380, 2006.

[60] Pascal Poupart and Craig Boutilier. Bounded finite state controllers. In *Proc. NIPS 2004*, pages 823–830, 2004.

[61] Raymond Reiter. *Knowledge in action: logical foundations for specifying and implementing dynamical systems*. MIT press, 2001.

[62] Jussi Rintanen. Complexity of planning with partial observability. In *Proc. ICAPS 2004*, pages 345–354, 2004.

[63] Jussi Rintanen. Conditional planning in the discrete belief space. In *Proc. IJCAI 2005*, pages 1260–1265, 2005.

[64] Abdallah Saffidine, François Schwarzentruber, and Bruno Zanuttini. Knowledge-based policies for qualitative decentralized pomdps. In *Proc. AAAI 2018*, pages 6270–6277, 2018.

[65] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[66] François Schwarzentruber. Hintikka's world: Agents with higher-order knowledge. In *Proc. IJCAI 2018*, pages 5859–5861, 2018.

[67] Allan Scott, Ulrike Stege, and Iris van Rooij. Minesweeper may not be NP-complete but is hard nonetheless. *The Mathematical Intelligencer*, 33: 5–17, 2011.

[68] Richard D. Smallwood and Edward J. Sondik. The optimal control of partially observable Markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973.

[69] Trao Cao Son and Chitta Baral. Formalizing sensing actions: A transition function based approach. *Artif. Intell.*, 125(1-2):19–91, 2001.

[70] Son Thanh To, Tran Cao Son, and Enrico Pontelli. A generic approach to planning in the presence of incomplete information: Theory and implementation. *Artif. Intell.*, 227:1–51, 2015.

[71] Ron van der Meyden and Moshe Y. Vardi. Synthesis from knowledge-based specifications (extended abstract). In *Proc. CONCUR 1998*, volume 1466, pages 34–49, 1998.

[72] Klaus W. Wagner. Bounded query classes. *SIAM J. Comput.*, 19(5):833–846, 1990.

[73] Benjamin Zarrieß and Jens Claßen. Verification of knowledge-based programs over description logic actions. In *Proc. IJCAI 2015*, pages 3278–3284, 2015.

[74] Benjamin Zarrieß and Jens Claßen. Decidable verification of Golog programs over non-local effect actions. In *Proc. AAAI 2016*, pages 1109–1115, 2016.