Verifying Belief-based Programs via Symbolic Dynamic Programming

ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230429

Verifying Belief-Based Programs via Symbolic Dynamic Programming

Daxin Liu [a,b;*], **Qinfei Huang**[b], **Vaishak Belle** [a] **and Gerhard Lakemeyer** [b]

[a]University of Edinburgh, the United Kingdom
[b]RWTH-Aachen University, Germany

Abstract. Belief-based programming is a probabilistic extension of the Golog programming language family, where every action and sensing could be noisy and every test refers to the subjective beliefs of the agent. Such characteristics make it rather suitable for robot control in a partial-observable uncertain environment. Recently, efforts have been made in providing formal semantics for belief programs and investigating the hardness of verifying belief programs. Nevertheless, a general algorithm that actually conducts the verification is missing. In this paper, we propose an algorithm based on symbolic dynamic programming to verify belief programs, an approach that generalizes the dynamic programming technique for solving (partially observable) Markov decision processes, i.e. (PO)MDP, by exploiting the symbolic structure in the solution of first-order (PO)MDPs induced by belief program execution.

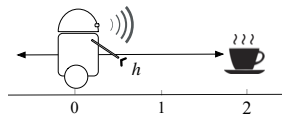
1 Introduction

The action programming language GOLOG [26], short for *alGOl in LOGic*, is a powerful tool to express high-level agent behavior. GOLOG is based on the situation calculus [34], a first-order language to model how the world changes as a result of actions. Belief programs, introduced by Belle and Levesque [8] in the ALLEGRO system, are probabilistic extensions of the GOLOG programs where every action and sensing could be noisy and tests refer to the agent's subjective degree of belief. These characteristics, amongst others, make belief programs rather suitable for robot control in a partially observable uncertain environment.

Before deployment, it is desirable to verify if a program meets certain properties. As a running example of a belief program, consider a variant of the coffee robot from [31, 28]. The robot in a one-dimensional world tries to search the coffee. Initially, the horizontal position h of the robot is 0 and coffee is at 2. The robot has a knowledge base about her location (usually a belief distribution). The robot might perform sensing $sencfe$ to detect if the coffee is at her current position and action $east$ to move a unit east. Note both $sencfe$ and $east$ could be noisy. E.g. $east$ might end up moving 1 or 2 units with, respectively, likelihood 0.8 and 0.2. A possible belief program is as

```
1 while B(h = 2) < 1 do
2     east|sencfe;
3 endWhile
```

and works as follows: while the robot does not fully believe that she reached the coffee (Line 1), she non-deterministically selects the action $east$ or sensing $sencfe$ to execute (Line 2). The program is an online program and its execution depends on the outcome of sensing as input. An interesting property of the program could be: whether the probability that *eventually* the robot fully *believes* she reaches the coffee is lower than 0.1.



To perform verification, Liu and Lakemeyer [31, 28] reconsider the proposal by Belle and Levesque [8] based on a probabilistic modal logic of belief and actions, called $\mathcal{DS}p$ [29]. Amongst others, their proposal allows specifying the properties of programs in a variant of PCTL logic [19]. For example, the above property can be expressed as $\mathbf{P}_{\leq 0.1}[\mathbf{F}\boldsymbol{B}(h = 2 \colon 1)]$. They also show that the verification problem is closely related to model-checking infinite horizon partially observable Markov decision process (POMDP), therefore undecidable.

Despite many theoretical results that were achieved, a general algorithm that actually conducts the verification is missing. In this paper, we propose an algorithm based on *symbolic dynamic programming* to verify belief programs. Symbolic dynamic programming (SDP) [36] is a generalization of the dynamic programming technique for solving (partially observable) Markov decision processes, i.e. (PO)MDP, that exploits the symbolic structure in the solution of relational and first-order(FO) logical MDPs. Such a technique suits us well in the sense that the execution of belief programs yields multiple ground POMDPs which can be captured by a single FO-POMDP.

The rest of the paper is arranged as follows: in Section 2, we review background knowledge about the logic of belief and action, i.e. the logic $\mathcal{DS}p$, the formal semantics of belief programs, and the verification problem. The symbolic dynamic programming algorithm is presented in Section 3. Section 4 introduces and evaluates an implementation of the algorithm. We end with a discussion on related works and future directions in Section 5.

2 Preliminary

2.1 The logic $\mathcal{DS}p$

We use the nullary fragment of the logic $\mathcal{DS}p$, a probabilistic modal logic of beliefs and actions, to model the agent's subjective belief. For simplicity, we only have two sorts: *number* and *action*, and by number, we mean *algebraic real number* [20] which includes irrational numbers such as $\sqrt{2}$. The language features a finite set of

* Corresponding Author. Email:daxin.liu@ed.ac.uk

nullary fluents (of type number) and a fixed countable domain with the unique name assumption: the set of *standard names* $\mathcal{N} = \mathbb{D} \cup \mathcal{N}_a$ where $\mathbb{D}$ are number names and $\mathcal{N}_a$ are action names. There is a modal operator $\boldsymbol{B}$ to express the agent's subjective belief.

Syntax Formally, the vocabulary of the language is taken from the following. Logical symbols include equality = and

- standard names $\{n, n', \ldots\}$ and variables $\{x, y, \ldots\}$;
- rigid function symbols $+$, $\times$, $east(1)$, $sencfe(1)$, etc.;
- finitely many nullary fluent functions $\{h_1, h_2, \ldots h_k\}$.
- a special unary fluent l that takes an action as its argument and returns the action's likelihood.

Logical symbols include connectives $\{\wedge, \forall, \neg\}$. We treat $\{\exists, \equiv, \supset\}$ as syntactic abbreviations. For simplicity, no predicates are considered. However, we will use "$\leq$" (defined on numbers) by assuming there is a rigid function to simulate it. *Terms* are the smallest set of expressions such that: 1) every variable and standard name (or constant) is a term; 2) if $t_1, \ldots t_k$ are terms and f is k-ary function symbol, then $f(t_1, \ldots t_k)$ is a term. *Well-formed formulas* are constructed as usual in first-order logic with equality. They can further be in the context of belief and action modalities. *Ground terms* are terms without free variables and *primitive terms* $\mathcal{P}_t$ are terms of the form $f(n_1 \ldots n_m)$, where $n_i \in \mathbb{D}$. Additionally, we assume $\mathcal{N}_a$ is just the set of action primitive terms.

The logic has an epistemic operator: $\boldsymbol{B}(\alpha\colon x)$ is to be read as "α is believed with a probability x" where x is a term of the number sort. We use $\boldsymbol{K}(\alpha)$ as abbreviation for $\boldsymbol{B}(\alpha\colon 1)$, read as "$\alpha$ is known". There are two action modalities $[a]$, $\Box$ in that if α is a formula, then so are $[a]\alpha$ (read:"α holds after a") and $\Box\alpha$ (read: "α holds after any sequence of actions"). For $z = a_1 \cdots a_k$, we write $[z]\alpha$ to mean $[a_1] \cdots [a_k]\alpha$. We use TRUE to denote truth, which is taken as an abbreviation for, say, $\forall x(x = x)$, and FALSE for its negation. For α, we use α_t^x to denote the formula obtained by substituting free variable x in α with term t. A formula without $\boldsymbol{B}$ is called *objective*, without $[a], \Box$ is called *static*, and without fluents, $[a], \Box$ outside $\boldsymbol{B}$ is called *subjective*.

Semantics The semantics is given in terms of possible worlds. In a dynamic setting, such worlds are defined to interpret not only the current state of affairs but also how that changes over actions. Let $\mathcal{Z} = (\mathcal{N}_a)^*$ be the set of all finite sequences of actions including $\langle\rangle$, the empty sequence. Then a *world* maps $\mathcal{P}_t \times \mathcal{Z}$ to $\mathcal{N}$ of the right sort satisfying the rigidity constraint: if $t \in \mathcal{P}_t$ is rigid , $w[t, z] = w'[t, z']$ for all w, w', z, and z'. Additionally, $+, \times, \leq$ are rigid and interpreted in the usual sense: E.g. $w[1 + 1, z] = 2$ for all w, z. This amounts to setting the theory of real as a background theory that admits quantifier elimination [40].

Let $\mathcal{W}$ be the set of all possible worlds. The *denotation* of ground terms (wrt a world w, action sequence z) is defined inductively. If $t \in \mathcal{N}$, then $\|t\|_{w,z} = t$; If $t = f(t_1, \ldots t_m)$ then $\|t\|_{w,z} = w[f(r_1, \ldots r_m), z]$ where $r_i = \|t_i\|_{w,z}$. If t is rigid, we write $\|t\|$.

An *epistemic state* e is a set of distributions d (weighted functions) that maps $\mathcal{W}$ to $\mathbb{R}^{\geq 0}$. By a model, we mean a triple $\langle e, w, z\rangle$.

Truth for objective sentences is given as:

- $e, w, z \models t_1 = t_2$ iff $\|t_1\|_{w,z} = \|t_2\|_{w,z}$;
- $e, w, z \models \neg\alpha$ iff $e, w, z \not\models \alpha$;
- $e, w, z \models \alpha \wedge \beta$ iff $e, w, z \models \alpha$ and $e, w, z \models \beta$;
- $e, w, z \models \forall x.\alpha$ iff $e, w, z \models \alpha_n^x$ for all $n \in \mathcal{N}$ for the right sort;
- $e, w, z \models [a]\alpha$ iff $e, w, z \cdot n \models \alpha$ where $n = \|a\|_{w,z}$;
- $e, w, z \models \Box\alpha$ iff $e, w, z \cdot z' \models \alpha$ for all $z' \in \mathcal{Z}$.

To account for stochastic actions, $\mathcal{DS}p$ uses a notion called *observational indistinguishability* among actions. The idea is that instead of saying stochastic actions have non-deterministic effects, $\mathcal{DS}p$ says stochastic actions have non-deterministic alternatives which are mutually observationally indistinguishable from the agent's perspective and each of which has a deterministic effect. In the coffee robot example, to express that action $east$ might end up in moving 1 or 2 units east non-deterministically, $\mathcal{DS}p$ uses two actions $east(1)$ and $east(2)$, and they are interpreted in that the robot intends to move a unit east but nature may select 1 or 2 as outcomes.

More formally (for simplicity, we assume actions have at most 1 argument), arguments of stochastic action $a(y)$ are *uncontrollable* and *unobservable* while arguments of sensing $sen(x)$ are *observable* yet *uncontrollable* to the agent. Under this convention, deterministic actions are just actions without arguments. For example, $east(1)$, $east(2)$ have uncontrollable and unobservable arguments 1 and 2, while $sencfe(1)$ has an uncontrollable yet observable argument 1 indicating the sensor receives a reading of 1.

Besides, we define action sequence *observational indistinguishability* as follows:

Definition 1 *We define $z \approx z'$: 1. $\langle\rangle \approx z'$ iff $z' = \langle\rangle$; 2. $z \cdot r \approx z'$ iff $z' = z^* \cdot r$, $z \approx z^*$ and r is a deterministic action or sensing action; 3. $z \cdot a(n) \approx z'$ iff $z' = z^* \cdot a(n')$ for some n' and $z \approx z^*$.*

For example, $east(1) \cdot east(2) \approx east(2) \cdot east(3)$. Lastly, we define the likelihood of action sequences in a world:

Definition 2 *We define $l^* : \mathcal{W} \times \mathcal{Z} \mapsto \mathbb{R}^{\geq 0}$:*

- $l^*(w, \langle\rangle) = 1$ *for all* $w \in \mathcal{W}$;
- $l^*(w, z \cdot r) = l^*(w, z) \times n$ *where* $w[l, z](r) = n$.

Given e, w, z and formula α, let $\|\alpha\|_{e,w,z} := \{\langle w', z'\rangle\colon z' \approx z, e, w', z' \models \alpha\}$. $\|\alpha\|_{e,w,z}$ is the set of all alternative worlds and actions that might result in α. For a distribution d, we define $\text{NORM}(d, \|\alpha\|_{\{d\},w,z}, n)$ if $n = \frac{1}{\eta} \times \sum_{\|\alpha\|_{\{d\},w,z}} d(w') \times l^*(w', z')$,[1] where η is a normalizer with the same expression as the numerator but replacing α to TRUE. Namely, the set of pairs of worlds and actions that result in α has proportioned summed weights (or probability) n. Now, we are ready to give truth for $\boldsymbol{B}$. Supposing r is a rigid term,

- $e, w, z \models \boldsymbol{B}(\alpha\colon r)$ iff for all $d \in e$, $\text{NORM}(d, \|\alpha\|_{\{d\},w,z}, \|r\|)$;

We freely use $\boldsymbol{B}(\alpha) \leq r$ (or "$<$") as formulas, they should be understood as syntactic abbreviations for $\exists x.\boldsymbol{B}(\alpha\colon x) \wedge x \leq r$.

For a sentence α, we write $e, w \models \alpha$ to mean $e, w, \langle\rangle \models \alpha$. When Σ is a set of sentences and α is a sentence, we write $\Sigma \models \alpha$ (read: Σ logically entails α) to mean that for all e and w, if $e, w \models \alpha'$ for every $\alpha' \in \Sigma$, then $e, w \models \alpha$. Satisfiability and validity are defined in the usual way. If α is an objective formula, we write $w \models \alpha$ instead of $e, w \models \alpha$. Similarly, we write $e \models \alpha$ instead of $e, w \models \alpha$ if α is subjective.

Basic action theory To infer belief after actions, $\mathcal{DS}p$ encompasses an action theory $\mathcal{D}_{dyn}$, including *successor state axioms* that incorporate Reiter's solution to the frame problem [34] and *likelihood axioms*, to specify the dynamics of a domain.

- **Successor state axioms (SSAs):** There is one such axiom for each fluent h: $\Box[a]h = x \equiv \Phi(a, x)$ [2] where $\Phi(a, x)$ is a formula with free variables among a, x;

[1] Since $\mathcal{W}$ is uncountable, NORM just requires the sum here to be "well-defined". See also [6] for details.

[2] Free variables are implicitly universally quantified. The $\Box$ has lower syntactic precedence than the connectives, and $[\cdot]$ has the highest priority.

- **Likelihood axioms (LAs):** We assume there are only finitely many action symbols, and finitely many such axioms, one for each action symbol. Particularly, LAs are of the form (respectively for stochastic actions and sensing)

$$\Box l(a(y)) = q \equiv \bigvee\nolimits_i y = n_i \wedge q = n_i^*$$

$$\Box l(sen(x)) = q \equiv \bigvee\nolimits_{i,j} x = n_i' \wedge \phi_j \wedge q = n_{i,j}^*$$

where ϕ_j are mutually exclusive and complete sentences. Namely, $\phi_j \models \neg\phi_{j'}$ and $\models \bigvee_j \phi_j \equiv$ TRUE. Essentially, the LAs say that both stochastic actions and sensing have finite outcomes. Moreover, the likelihood of sensing might depend on the context (ϕ_j) but the likelihood of stochastic actions is fixed. [3] Besides, all actions have finitely many outcomes with non-zero likelihood.

Example 1 *In the robot example, we might have SSAs and LAs as:*

$$\Box[a]h = x \equiv \exists y.a = east(y) \wedge x = y + h \vee \forall y.a \neq east(y) \wedge x = h$$

$$\Box l(east(y)) = x' \equiv y = 1 \wedge x' = 0.8 \vee y = 2 \wedge x' = 0.2 \vee y \notin \{1,2\} \wedge x' = 0$$

$$\Box l(sencfe(x)) = x' \equiv x = 1 \wedge h = 2 \wedge x' = 1 \vee x = 0 \wedge h \neq 2 \wedge x' = 1 \vee x = 1 \wedge h \neq 2 \wedge x' = 0 \vee x = 0 \wedge h = 2 \wedge x' = 0$$

That is, the robot's new location x is determined by its current location h and the actual moved distance y. Stochastic action $east(y)$ might end up moving 1 (0.8 likelihood) or 2 units (0.2 likelihood). Besides, sensing action $sencfe$ is accurate: if the robot is at the coffee's position, i.e. $h = 2$, executing $sencfe$ receives a reading 1 with 100% likelihood and a reading 0 with zero likelihood.

Regression & progression The *regression* of a formula α through an action a is another α' that holds prior to a being performed iff $[a]\alpha$ holds. The idea is to recursively replace formulas of the form $[t]h = n$ by the RHS of h's SSA with substitutions. E.g. $\Re([east(2)]h = 2, \mathcal{D}_{dyn}) := (\exists y.east(2) = east(y) \wedge 2 = y + h \vee \forall y.a \neq east(y) \wedge x = h) \equiv (h = 0)$, given $\mathcal{D}_{dyn}$ as Example 1.

[30] also proposed a regression operator for belief formulas such as $[east(1)]\boldsymbol{B}(h = 2\colon 0.2)$ if an initial *belief distribution* is given. A belief distribution is a formula of the form $\forall x.\boldsymbol{B}(h = x\colon f(x))$ specifying a distribution of the random variable h where f is a rigid mathematical function (we write $\boldsymbol{B}^f$ for short). A similar formula exists in case there are multiple random variables. For simplicity, we only consider one random variable and initial belief distribution that only finitely many points have a non-zero degree of belief, i.e. $\forall x.\boldsymbol{B}(h = x\colon f(x)) \equiv \bigwedge_j \boldsymbol{B}(h = n_j\colon r_j)$. For example, a possible initial belief distribution for the robot coffee could be $\boldsymbol{B}^{f_0} \equiv \boldsymbol{B}(h = 0\colon 1)$. Namely, the robot fully believes she is at location 0. With an initial belief distribution, the regression of beliefs is defined as the evaluation against the initial beliefs. For example:

$$\Re([east(2)]\boldsymbol{B}(h = 2\colon 0.2), \mathcal{D}_{dyn}, \boldsymbol{B}^{f_0}) := \sum_{n_j}\sum_{n_i} r_j \times l(east(n_i)) \begin{cases} 1 & h = n_i \supset \Re([east(n_i)]h = 2) \\ 0 & o.w. \end{cases}$$

$$\equiv 0.2 = 1 \times 0.2$$

which is then evaluated to TRUE.[4]

[3] It is possible to allow likelihood of stochastic actions to vary as well, for simplicity, we only consider rigid likelihood.

[4] The assumption that only finite action outcomes have non-zero likelihood and finite values of random variables have a non-zero degree of belief ensures summation here is finite and hence can be replaced by a finite plus.

The inverse of regression is *progression*, which takes a formula ψ and action a and represents logical consequences in a *static formula* ψ' (formula without $[a]$ and $\Box$). Unlike regression, restrictions must be imposed to ensure ψ' is first-order [27]. Nevertheless, [29] showed that if the input is a belief distribution $\boldsymbol{B}^f$, then the progression is always first-order and in the form of another belief distribution $\boldsymbol{B}^{f'}$, i.e. $\boldsymbol{B}^f \wedge \boldsymbol{K}\mathcal{D}_{dyn} \models [a](\boldsymbol{B}^{f'} \wedge \boldsymbol{K}\mathcal{D}_{dyn})$. For example, the progression of $\boldsymbol{B}^{f_0}$ wrt action $east(1)$ is $\boldsymbol{B}^{f_1}$, and the progression of $\boldsymbol{B}^{f_1}$ wrt action $sencfe(1)$ is $\boldsymbol{B}^{f_2}$, where $f_1(x)$ and $f_2(x)$ are as:

$$f_1(x) = \begin{cases} 0.8 & x = 1 \\ 0.2 & x = 2 \\ 0 & o.w. \end{cases} \quad \text{and } f_2(x) = \begin{cases} 1 & x = 2 \\ 0 & o.w. \end{cases}$$

We comment that both of these results are significant improvements and advancements over classical regression and progression [34] as well as the epistemic non-probabilistic regression and progression by Scherl and Levesque [37] or Liu and Wen [32].

2.2 Verification of belief programs

Belief programs are probabilistic extensions of GOLOG, where every action and sensing could be noisy, and every test is referring to the agent's subjective beliefs.

Belief program The basic ingredient of belief programs is the so-called *primitive programs* which are actions that suppress their uncontrollable parameters. For example, for the action $sencfe(1)$ and $east(2)$, their respective primitive programs are $sencfe$ and $east$. A primitive program ϱ is instantiated by a ground action t_a, i.e. $\varrho \to t_a$ iff $\models \exists y.t_a = \varrho[y]$, where $\varrho[y]$ is the action that restores its suppressed parameters by y.

A *program expression* can be a primitive program ϱ, a test $\alpha?$ where α is a static subjective formula, a sequence $\delta;\delta$, a nondeterministic choice $(\delta|\delta)$, and a non-deterministic iteration δ^* of programs. Formally, the program expression can be defined as:

$$\delta := \varrho|\alpha?|(\delta;\delta)|(\delta|\delta)|\delta^*$$

The **if**-statements and **while**-loops are then defined as abbreviations:

$$\textbf{if } \alpha \textbf{ then } \delta_1 \textbf{ else } \delta_2 \textbf{ endIf} := [\alpha?;\delta_1]|[\neg\alpha?;\delta_2]$$

$$\textbf{while } \alpha \textbf{ do } \delta \textbf{ endWhile} := [\alpha?;\delta]^*;\neg\alpha$$

An example is the coffee robot program δ_{cfe} in the introduction. Given a set of formulas without modality $\mathcal{D}_0$ that describes the initial world state, an initial belief distribution $\boldsymbol{B}^f$, a BAT $\mathcal{D}_{dyn}$, and a program expression δ, a belief program $\mathcal{BP}$ is a pair of the form:

$$\mathcal{BP} = (\mathcal{D}_0 \cup \mathcal{D}_{dyn} \cup \boldsymbol{B}^f \cup \boldsymbol{K}\mathcal{D}_{dyn}, \delta).$$

For the coffee robot, a belief program could be

$$\mathcal{BP}_{cfe} = (\{h \leq 0\} \cup \mathcal{D}_{dyn} \cup \boldsymbol{B}^{f_0} \cup \boldsymbol{K}\mathcal{D}_{dyn}, \delta_{cfe})$$

where $\mathcal{D}_{dyn}$ is as in Example 1. Namely, the robot initially is at a non-positive position $h \leq 0$ yet she fully believes that she is at position 0, i.e $\boldsymbol{B}^{f_0}$. To handle termination and failure, we reserve two nullary fluents $Final$ and $Fail$. Moreover, $\Box[a]Final = u \equiv a = \epsilon \wedge u = 1 \vee Final = u$ (likewise for $Fail$ with action ϝ) are implicitly assumed to be part of $\mathcal{D}_{dyn}$. Additionally, $\mathcal{D}_0 \models Final = 0 \wedge Fail = 0$, and actions ϵ, ϝ do not occur in δ.

Program semantics & the verification problem A *configuration* $\langle z, \delta\rangle$ of a program consists of an action sequence z and a program expression δ, where z is an action history and δ is the remaining program to be executed. Given $\mathcal{BP} = (\mathcal{D}_0 \cup \mathcal{D}_{dyn} \cup \boldsymbol{B}^f \cup \boldsymbol{K}\mathcal{D}_{dyn}, \delta)$, the transition relation $\xrightarrow{e}$ among configurations given e is defined inductively as:(See [31, 28] for a full list):

1. $\langle z, \varrho\rangle \xrightarrow{e} \langle z \cdot t, \langle\rangle\rangle$, if $\varrho \to t$;
2. $\langle z, \delta_1; \delta_2\rangle \xrightarrow{e} \langle z \cdot t, \delta'; \delta_2\rangle$, if $\langle z, \delta_1\rangle \xrightarrow{e} \langle z \cdot t, \delta'\rangle$;
3. $\langle z, \delta^*\rangle \xrightarrow{e} \langle z \cdot t, \delta'; \delta^*\rangle$, if $\langle z, \delta\rangle \xrightarrow{e} \langle z \cdot t, \delta'\rangle$.

The set of *final configurations* wrt e, i.e. $Fin(e)$, is defined as the smallest set such that:

1. $\langle z, \langle\rangle\rangle \in \mathrm{Fin}(e)$, $\langle z, \delta^*\rangle \in \mathrm{Fin}(e)$;
2. $\langle z, \alpha?\rangle \in \mathrm{Fin}(e)$ if $e, w, z \models \alpha$;
3. $\langle z, \delta_1; \delta_2\rangle \in \mathrm{Fin}(e)$ if $\langle z, \delta_1\rangle \in \mathrm{Fin}(e)$ and $\langle z, \delta_2\rangle \in \mathrm{Fin}(e)$;
4. $\langle z, \delta_1 | \delta_2\rangle \in \mathrm{Fin}(e)$ if $\langle z, \delta_1\rangle \in \mathrm{Fin}(e)$ or $\langle z, \delta_2\rangle \in \mathrm{Fin}(e)$;

The set of *failing configuration* is given by: $Fail(e) = \{\langle z, \delta\rangle | \langle z, \delta\rangle \notin Fin(e), \neg\exists\langle z \cdot t, \delta'\rangle . \langle z, \delta\rangle \xrightarrow{e} \langle z \cdot t, \delta'\rangle\}$.

The execution of a belief program $\mathcal{P}$ in a pair $\langle e, w\rangle$ such that $e, w \models \mathcal{D}_0 \cup \mathcal{D}_{dyn} \cup \boldsymbol{B}^f \cup \boldsymbol{K}\mathcal{D}_{dyn}$ yields a countably infinite *partially observable Markov decision process* (POMDP) $M_\delta^{e,w} = (S, A, P, O, \Omega, s_0)$ as follows:

- S: the set of configurations reachable from s_0 under $\xrightarrow{e}{}^*$ (transitive and reflexive closure of $\xrightarrow{e}$);
- A: the finite set of primitive programs in δ;
- s_0: the initial state $\langle\langle\rangle, \delta\rangle$;
- O: the observations are the reachable belief distributions $\boldsymbol{B}^{f'}$ from $\boldsymbol{B}^f$, namely, $\{\boldsymbol{B}^{f'} | \exists z \in \mathcal{Z}. \boldsymbol{B}^f \wedge \boldsymbol{K}\mathcal{D}_{dyn} \models [z]\boldsymbol{B}^{f'}\}$.
- Ω: $S \mapsto O$ a deterministic observation mapping as (suppose $s = \langle z, \delta\rangle$) $\Omega(s) = \boldsymbol{B}^{f'}$ iff $\boldsymbol{B}^f \wedge \mathcal{D}_{dyn} \models [z]\boldsymbol{B}^{f'}$.
- P: $S \times A \times S \to \mathbb{R}$ the transition probability $P(\langle z, \delta\rangle, \varrho, \langle z \cdot t_a, \delta'\rangle)$ given by:

$$P(\cdot) = \begin{cases} n & \text{if } \varrho \to t_a, w, z \models l(t_a) = n, \langle z, \delta\rangle \xrightarrow{e} \langle z \cdot t_a, \delta'\rangle \\ 1 & \text{if } \langle z, \delta\rangle \in Fin(e) \text{ and } \varrho = t = \delta' = \epsilon \\ 1 & \text{if } \langle z, \delta\rangle \in Fail(e) \text{ and } \varrho = t = \mathfrak{f}, \delta' = \delta \\ 0 & o.w. \end{cases}$$

We comment that it is a POMDP rather than MDP because we are interested in the set of possible strategies that the robot can perform. The fact that the set of observations is just the set of reachable beliefs distribution means that the robot can only act according to its beliefs about the environment rather than the actual world state, leading to a partially observable setting.

The non-determinism of programs is resolved by a *policy*, which is an action-selection strategy that maps each state to an action, i.e., $\sigma : S \to A$. We will only focus on the *observation-based* policies Σ_{obs}: a policy $\sigma : S \to A$ is observation-based iff for all $s, s' \in S$, if $\Omega(s) = \Omega(s')$ then $\sigma(s) = \sigma(s')$. [5] An infinite path of the form $\pi = s_0 \xrightarrow{\varrho_1} s_1 \xrightarrow{\varrho_2} s_2 \cdots$ is called a σ-*path* if $\sigma(s_j) = \varrho_{j+1}$ for all $j \geq 0$. The j-th state of any such path is denoted by $\pi[j]$. The set of all σ-*paths* starting in s is denoted by $\mathsf{Path}^\sigma(s, M_\delta^{e,w})$.

Every policy σ induces a probability space Pr_s^σ on the set of infinite paths starting in the state s, using the cylinder set construction [42]. For any finite path prefix $\pi_{\mathsf{fin}} = s_0 \xrightarrow{\varrho_1} s_1 \cdots s_n$, the probability measure is defined as: $\mathsf{Pr}_{s_0,\mathsf{fin}}^\sigma = P(s_0, \varrho_1, s_1) \cdots P(s_{n-1}, \varrho_n, s_n)$. This extends to a unique measure Pr_s^σ.

Given a POMDP $M_\delta^{e,w}$, a policy σ, and a static subjective formulas ψ_{goal} as goal, we say that a state $s = \langle z, \delta\rangle$ satisfies ψ_{goal}, i.e. $s \models \alpha$ iff $e, w, z \models \psi_{goal}$, an infinite path π satisfies eventually ψ_{goal}, i.e. $\pi \models \mathbf{F}\psi_{goal}$, iff $\exists k.\pi[k] \models \psi_{goal}$, where $\pi[k]$ is the k-th state in π. The *reachability probability* $\mathsf{Pr}_{s_0}^\sigma(\mathbf{F}\psi_{goal})$ is then defined $\mathsf{Pr}_{s_0}^\sigma(\{\pi | \pi \models \mathbf{F}\psi_{goal}\})$. Given a program $\mathcal{BP}$, a static subjective formula α, and a threshold τ, the *verification problem* ask if

$$\max_{\sigma \in \Sigma_{obs}} \{\mathsf{Pr}_{s_0}^\sigma(\mathbf{F}\psi_{goal})\} \leq \tau$$

for all POMDPs $M_\delta^{e,w}$ with $e, w \models \mathcal{D}_0 \cup \mathcal{D}_{dyn} \cup \boldsymbol{B}^f \cup \boldsymbol{K}\mathcal{D}_{dyn}$. We write $\mathcal{BP} \models \mathsf{Pr}(\mathbf{F}\psi_{goal}) \leq \tau$ if the program satisfies the property. [6]

In the coffee robot example, we might be interested in whether the maximal reachability probability for the robot to fully believe it reaches the coffee is no more than 0.1. Namely, deciding $\mathcal{BP}_{cfe} \models \mathsf{Pr}(\mathbf{F}\boldsymbol{K}h = 2) \leq 0.1$.

3 Verification by symbolic dynamic programming

One way to solve a POMDP is *value iteration*. Since the optimal policy of a POMDP might not be state-based, one has to transfer it into a continuous *belief MDP* so that the optimal policy of the original POMDP corresponds to the optimal state-based policy of the belief MDP [1]. The states B of the belief MDP are called *belief states*,[7] and are distributions over states of the original POMDP.

In value iteration, every belief state $\mathbf{b}$ is assigned a value $V(\mathbf{b})$ representing the maximal reachability probability that could achieve from the belief state $\mathbf{b}$, hence we are interested in the value of the initial belief state $\mathbf{b}_I$ with $\mathbf{b}_I(s_0) = 1$. The value function V is essentially the fixed-pointed of the Bellman backup operator and thus can be computed iteratively [11, 39]: $V(\mathbf{b}) = max_{a \in A} \sum_{\mathbf{b}' \in B} P(\mathbf{b}, a, \mathbf{b}') \times V(\mathbf{b}')$. While the number of belief states B is infinite, it is well-known that the optimal value function for finite horizon H is piece-wise linear and convex, therefore can be approximated arbitrarily closely as the upper envelope of a finite set of "α-vectors" V as $V(\mathbf{b}) = \max_{\alpha \in V} \mathbf{b} \cdot \alpha$. Hence value iteration is transformed to compute V via:

$$V^H = \bigcup_{a \in A} V_a^H \text{ and } V_a^H = \boxplus_{o \in O} V_{a,o}^H \tag{1a}$$

$$V_{a,o}^H = \{\alpha^{a,o} | \alpha \in V^{H-1}\} \tag{1b}$$

$$\alpha^{a,o}(s) = \sum_{s' \in S} [\![\Omega(s') = o]\!] \cdot P(s, a, s') \cdot \alpha(s') \tag{1c}$$

where $V_1 \boxplus V_2 = \{\alpha_1 + \alpha_2 | \alpha_1 \in V_1, \alpha_2 \in V_2\}$, $[\![x]\!]$ is the indicator function and returns 1 if x is true and 0 otherwise, and P is the transition function of the original POMDP.

3.1 FO-POMDP for belief programs

A *first-order partially observable MDP* (FO-POMDP) is a compact representation of (infinite) POMDP which abstractly defines the states S_{FO}, actions A_{FO}, observations O_{FO}, transition probability P_{FO} and observation function Ω_{FO}. It is suitable for our need to abstractly represent the underlying infinite POMDPs generated by the execution of belief programs $M_\delta^{e,w}$.

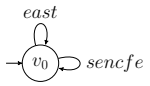
To begin with, we need the notion of *characteristic program graph* G_δ [13] of a program δ. The nodes of G_δ are the reachable subprograms $Sub(\delta)$, each of which is associated with a termination condition $\mathsf{Fin}(\delta)$ (initial node v_0 corresponds to the overall program) and

[5] The definition is slightly different from the classical notion of observation-based policy [33] as theirs are defined in terms of observation history. However, in our case, an observation is a belief distribution, which already encodes a history.

[6] Here, we only consider the upper bound. A dual question is to consider the lower bound, i.e. determining if $\tau \leq \min_{\sigma \in \Sigma_{obs}} \{\mathsf{Pr}_{s_0}^\sigma(\mathbf{F}\psi_{goal})\}$ for all POMDP $M_\delta^{e,w}$ which involves policy synthesis [33]. We leave it for future.

[7] This is in contrast to our model of belief $\boldsymbol{B}$. Here, belief is just a distribution over states, that is, it is a meta-linguistic notion while $\boldsymbol{B}$ is a logical notion.

an edge $\delta_1 \xrightarrow{\varrho/\alpha} \delta_2$ represents a transition from δ_1 to δ_2 by the primitive program ϱ if test condition α holds. Moreover, failure conditions are given by $\mathsf{Fail}(\delta') ::= \neg(\mathsf{Fin}(\delta') \vee \bigvee_{\delta' \xrightarrow{\varrho/\alpha} \delta''} \alpha)$. In the coffee robot program, $G_{\delta_{cfe}}$ has only one node with two self-loops.



Given a belief program $\mathcal{BP}$, we define a FO-POMDP $M_{\mathsf{FO}}^{\mathcal{BP}} = (S_{\mathsf{FO}}, A_{\mathsf{FO}}, P_{\mathsf{FO}}, s_0, O_{\mathsf{FO}}, \Omega_{\mathsf{FO}})$ in the following way:

- S_{FO}: the set of states are of the form $\langle \phi, \delta \rangle$, where ϕ is a first-order $\mathcal{DS}p$ formula. Intuitively, $\langle \phi, \delta \rangle$ represents the set of all possible worlds w, epistemic states e, and program configurations $\langle z, \delta \rangle$ such that $e, w, z \models \phi$;
- A_{FO}: the finite set of primitive programs in δ;
- $s_{\mathsf{FO}}^0 = \langle \mathcal{D}_0, \delta \rangle$: is the initial state.[8]
- O_{FO}: the observations are the same as in $M_\delta^{e,w}$;
- Ω_{FO}: a deterministic observation function $\Omega_{\mathsf{FO}}(\langle \phi, \delta \rangle) = \boldsymbol{B}^{f'}$ if $\phi \models \boldsymbol{B}^{f'}$;
- $P_{\mathsf{FO}}(\langle \phi_1, \delta_1 \rangle, \varrho, \langle \phi_2, \delta_2 \rangle)$ is given as:

$$P_{\mathsf{FO}}(\cdot) = \begin{cases} n & \text{if } \delta_1 \xrightarrow{\varrho/\beta} \delta_2 \in G_\delta \text{ for some } \beta \text{ and} \\ & \exists t_a . \varrho \to t_a \text{ and } \phi_1 \models l(t_a) = n \wedge \beta \\ 1 & \text{if } \phi_1 \models Final = 1 \text{ and } \varrho = \delta_2 = \epsilon \\ 1 & \text{if } \phi_1 \models Fail = 1 \text{ and } \varrho = \delta_2 = \mathfrak{f} \\ 0 & o.w. \end{cases}$$

Case Statement To concisely represent P_{FO} and distributions over states S_{FO} of the FO-POMDP, we introduce a tabular meta-logical device called *case statement* [35]. A case-statement $Case$ defines a distribution over the state $\langle \phi, \delta \rangle \in S_{\mathsf{FO}}$ as

$$Case[\phi_1, \delta_1, t_1; \ldots \phi_k, \delta_k, t_k] \equiv \boxed{\begin{array}{c} \langle \phi_1, \delta_1 \rangle : t_1 \\ \vdots \\ \langle \phi_k, \delta_k \rangle : t_k \end{array}}$$

Additionally, $Case = t$ iff for $1 \le i \le k$, if ϕ_i holds and the remaining program is δ_i, then $t = t_i$. We assume $t_1, \ldots t_k$ are constants for simplicity. For example, the following case statement expresses a distribution that assigns all weights to the state that satisfies $\boldsymbol{B}(h = 2 \colon 1)$ with δ_{cfe} as the remaining program:

$$Case_0 := \boxed{\langle \boldsymbol{K} h = 2, \delta_{cfe} \rangle \colon 1} \tag{2}$$

For a given program $\mathcal{BP}$, we might wish to use some logical operator over its reachable sub-program $Sub(\delta)$ in case statements as shorthand, they are defined as

$$Case[\phi, \delta_1 \vee \delta_2, t] := Case[\phi, \delta_1, t; \phi, \delta_2, t]$$
$$Case[\phi, \neg\delta_1, t] := Case[\phi, \delta_2, t; \ldots \phi, \delta_k, t]$$

where $\delta_2, \ldots \delta_k$ are programs in $Sub(\delta)$ and distinct from δ_1. Following this definition, we use δ_T to mean the disjunctions for all $\delta_i \in Sub(\delta)$ and δ_F for its negation.

Besides, we define sum $\oplus$ (likewise for product $\otimes$) over two case statements as (we might use a scalar value n in $\oplus$ or $\otimes$, they should be understood as $\boxed{\langle \textsc{True}, \delta_{\textsc{True}} \rangle : n}$)

$$\boxed{\begin{array}{c} \langle \phi_1, \delta_1 \rangle : t_1 \\ \langle \phi_2, \delta_2 \rangle : t_2 \end{array}} \oplus \boxed{\begin{array}{c} \langle \phi_1', \delta_1' \rangle : t_1' \\ \langle \phi_2', \delta_2' \rangle : t_2' \end{array}} = \boxed{\begin{array}{c} \langle \phi_1 \wedge \phi_1', \delta_1 \wedge \delta_1' \rangle : t_1 + t_1' \\ \langle \phi_1 \wedge \phi_2', \delta_1 \wedge \delta_2' \rangle : t_1 + t_2' \\ \langle \phi_2 \wedge \phi_1', \delta_2 \wedge \delta_1' \rangle : t_2 + t_1' \\ \langle \phi_2 \wedge \phi_2', \delta_2 \wedge \delta_2' \rangle : t_2 + t_2' \end{array}}$$

Lastly, we need to compute the *preimage* of a case statement wrt a ground action t_a in a program graph G_δ: (supposing $\delta_{i,j} \xrightarrow{\varrho/\beta_{i,j}} \delta_i \in G_\delta$ and $\varrho \to t_a$)

$$Pre(Case[\phi_i, \delta_i, t_i], G_\delta, t_a) := Case[\beta_{i,j} \wedge [t_a]\phi_i, \delta_{i,j}, t_i].$$

If no ambiguity, we write $Pre(Case, t_a)$ or $Pre(Case)$ for short. Essentially, $Pre(Case)$ represents the conditions that prior distributions over S_{FO} should satisfy in order to result in $Case$ after the action t_a. For example, $\boxed{\langle [sencfe(1)]\boldsymbol{K} h = 2, \delta_{cfe} \rangle \colon 1}$ is the preimage of $Case_0$ wrt $G_{\delta_{cfe}}$ and action $sencfe(1)$. This is because $G_{\delta_{cfe}}$ only has self-loops, the program expression previous to the action $sencfe(1)$ can only be δ_{cfe} itself.

Now we show how to represent P_{FO}. First, for each ground action t_a, according to the BAT, $l(t_a)$ can be represented by a case statement: $l(t_a) := Case[\phi_j, \delta_T, n_j]$, where ϕ_j is the likelihood conditions and n_j is the corresponding values. In case t_a is a stochastic action, $\phi_j \equiv \textsc{True}$. Next, recall that P_{FO} in $M_{\mathsf{FO}}^{\mathcal{BP}}$ not only depends on $l(t_a)$ but also the program graph G_δ. Hence, given G_δ, (supposing $\delta_i \xrightarrow{\varrho/\beta_i} \delta_{i'} \in G_\delta$ and $\varrho \to t_a$), we define:

$$P_{\mathsf{FO}}^{G_\delta}(\varrho, t_a) := Case[\beta_i, \delta_i, 1] \otimes l(t_a).$$

Then, we can represent the transition probability P_{FO} by a set of case statements, one for each primitive program and ground action. In the coffee robot program example, P_{FO} contains the following: [9]

$P_{\mathsf{FO}}^{G_\delta}(east, east(1)) := \boxed{\langle \textsc{True}, \delta_{cfe} \rangle : 0.8}$
$P_{\mathsf{FO}}^{G_\delta}(east, east(2)) := \boxed{\langle \textsc{True}, \delta_{cfe} \rangle : 0.2}$
$P_{\mathsf{FO}}^{G_\delta}(sencfe, sencfe(1)) := \boxed{\langle h = 2, \delta_{cfe} \rangle : 1}$
$P_{\mathsf{FO}}^{G_\delta}(sencfe, sencfe(0)) := \boxed{\langle h \neq 2, \delta_{cfe} \rangle : 1}$

3.2 Symbolic dynamic programming

To solve the FO-POMDP for a program $\mathcal{BP}$, we propose a *symbolic dynamic programming* (SDP) algorithm in the form of lifted α-vectors. Lifted α-vectors will be represented as case statements, e.g. $\alpha Case$ and the lifted POMDP value function will be represented as a maximization over a set $V^H = \{\alpha Case\}$:

$$V^H(\mathbf{b}) = \max_{\alpha Case \in V^H} \mathbf{b} \otimes \alpha Case. \tag{3}$$

For a goal ψ_{goal}, we initialize the V^0 by the single case statement $\alpha Case_0 = \boxed{\langle \psi_{goal}, \delta_T \rangle : 1}$. Intuitively, if initially ψ_{goal} already holds, then the reachability would be 1 with the optimal policy saying do nothing. For example, we have $\alpha Case_0 := Case_0$ as in Eq. (2) for the goal $\boldsymbol{K} h = 2$ and program $\mathcal{BP}_{cfe}$.

We comment that an important concern in the implementation of evaluating $V^H(\mathbf{b})$ for a belief state $\mathbf{b}$ or, more general, computing $\oplus, \otimes$ over case statements is to reduce the number of cases by removing inconsistent cases. We achieve this by an external theorem prover. Moreover, while it is possible to eliminate $[a]$ for objective formulas by regression (wrt BAT $\mathcal{D}_{dyn}$) in order to identify equivalent cases,

[8] The BAT and initial belief distribution $\mathcal{D}_{dyn} \wedge \boldsymbol{B}^f \wedge \boldsymbol{K}\mathcal{D}_{dyn}$ is implicitly included.

[9] We omit cases with 0 values for short.

it is impossible to eliminate $\boldsymbol{B}$ as regression of $\boldsymbol{B}$ is essentially evaluating it against an input belief distribution $\boldsymbol{B}^f$. Nevertheless, such a procedure requires extra effort (we defer a discussion on this to the Implementation section). For the purpose of presentation, we eliminate $[a]$ whenever possible by regression.

Relevant observations Another problem in value iteration for the FO-POMDP is that observations O_{FO} are infinite, hence it is infeasible to perform iterations as in Eq. (1a). The idea here is to identify the set of *relevant observations* at iterations [36]. Given a belief distribution (or observation) $\boldsymbol{B}^f$ and a primitive program ϱ, the set of relevant observations O^ϱ are fixed and determined by the set of ground actions t_a s.t. $\varrho \rightarrow t_a$ (as new observations are the progressions of $\boldsymbol{B}^f$ wrt t_a). We denote the set of all relevant observations O_{Rel} as $O_{Rel} = \bigcup_\varrho O^\varrho$. Additionally, $O^\varrho = \bigcup_{t_i} O^{t_i}$ where $\varrho \rightarrow t_i$.

In our coffee robot example, we have $O_{Rel} = O^{east} \cup O^{sencfe} = O^{east(1)} \cup O^{sencfe(1)} \cup O^{sencfe(0)}$. Note that $O^{east(2)} = O^{east(1)}$ because starting from any belief distribution, action $east(2)$ and $east(1)$ will result in the same belief distribution as the robot has no knowledge about the actual outcome and yields the same beliefs.

We propose an anytime SDP algorithm that runs for a given number of iterations N (N represents policies' maximal horizons in consideration) as follows:

- Step 1: For $H = 0$, set $\alpha Case_0^1 = \boxed{\langle \psi_{goal}, \delta_T \rangle : 1}$, and $V^0 = \{\alpha Case_0^1\}$.
- Step 2: compute $V^H(\mathbf{b}_I)$ via Eq.(3) for the initial belief state $\mathbf{b}_I := \boxed{\langle \mathcal{D}_0, \delta \rangle : 1}$. Regress formulas in $V^H(\mathbf{b}_I)$ wrt $\boldsymbol{B}^f$ and $\mathcal{D}_{dyn}$. If there exists a case $\boxed{\langle \phi_i, \delta_i \rangle : t_i} \in V^H(\mathbf{b}_I)$ such that ϕ_i is satisfiable, $\delta_i = \delta$, and $t_i > \tau$, then return "$\mathcal{BP} \not\models \mathsf{Pr}(\mathbf{F}\psi_{goal}) \leq \tau$", else set $H = H + 1$.
- Step 3: For each primitive program $\varrho \in A_{\mathsf{FO}}$ and each $o \in O_{Rel}$, compute a $\alpha Case_H^{\varrho,o,j}$ for each $\alpha Case_{H-1}^j \in V^{H-1}$ by summing over all nature's choice of outcomes t_i for ϱ:

$$\alpha Case_H^{\varrho,o,j} = \oplus_{t_i} [\![o \in O^{t_i})]\!] \otimes P_{\mathsf{FO}}^{G_\delta}(\varrho, t_i) \otimes Pre(\alpha Case_{H-1}^j, G_\delta, t_i)$$

  where $[\![o \in O^{t_i})]\!]$ returns 1 if $o \in O^{t_i}$ else 0.
- Step 4: Compute the set $V_{\varrho,o}^H$ by:

$$V_{\varrho,o}^H = \{\alpha Case_H^{\varrho,o,j} | \alpha Case_{H-1}^j \in V^{H-1}\}$$

- Step 5: Compute the set V_ϱ^H for primitive program ϱ by summing over all relevant observations in O^ϱ: $V_\varrho^H = \boxplus_{o \in O^\varrho} V_{\varrho,o}^H$.
- Step 6: Union the sets V_ϱ^H for all primitive program ϱ to form V^H: $V^H = \bigcup_\varrho V_a^H$.
- Step 7: If horizon $H < N$ go to step 2, else returns "unkown"

In the robot example, we have:

$V^0(\mathbf{b}_I) = \boxed{\langle h \leq 0, \delta_{cfe} \rangle : 1} \otimes \boxed{\langle \boldsymbol{K}(h = 2), \delta_T \rangle : 1}$

$= \boxed{\langle h \leq 0 \wedge \boldsymbol{K}(h = 2), \delta_{cfe} \rangle : 1}$

After eliminating $\boldsymbol{B}$ by regression (wrt $\boldsymbol{B}^{f_0}$ and $\mathcal{D}_{dyn}$), we have

$V^0(\mathbf{b}_I) = \boxed{\langle h \leq 0 \wedge \textsc{False}, \delta_{cfe} \rangle : 1} = 0.$

This is because $\boldsymbol{B}^{f_0}$ says $\boldsymbol{K}h = 0$ which implies $\neg \boldsymbol{K}h = 2$ and the fact that any inconsistent case statement amounts to 0. Namely, the maximal reachability is 0 for policies with 0 horizons.

When setting $H = 1$, there are three distinct relevant observations O_{Rel} and two primitive programs $\{east, sencfe\}$. Below, we use E, S as shorthand for $east, sencfe$ respectively.

$\alpha Case_1^{E,O^{E(1)},1}$

$= 1 \times P_{\mathsf{FO}}^{G_\delta}(E, E(1)) \otimes Pre(\boxed{\langle \boldsymbol{K}(h = 2), \delta_{cfe} \rangle : 1}, E(1))$

$\oplus 1 \times P_{\mathsf{FO}}^{G_\delta}(E, E(2)) \otimes Pre(\boxed{\langle \boldsymbol{K}(h = 2), \delta_{cfe} \rangle : 1}, E(2))$

$= \boxed{\langle \textsc{True}, \delta_{cfe} \rangle : 0.8} \otimes \boxed{\langle [E(1)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1}$

$\oplus \boxed{\langle \textsc{True}, \delta_{cfe} \rangle : 0.2} \otimes \boxed{\langle [E(2)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1}$

$= \boxed{\langle [E(1)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1}$

The last equality is because $[E(1)]\boldsymbol{K}h = 2$ and $[E(2)]\boldsymbol{K}h = 2$ are logically equivalent: again the robot has no idea about the outcome of $east$. Likewise:

$\alpha Case_1^{S,O^{S(1)},1}$

$= 1 \times P_{\mathsf{FO}}^{G_\delta}(S, S(1)) \otimes Pre(\boxed{\langle \boldsymbol{K}(h = 2), \delta_{cfe} \rangle : 1}, S(1))$

$\oplus 0 \times P_{\mathsf{FO}}^{G_\delta}(S, S(0)) \otimes Pre(\boxed{\langle \boldsymbol{K}(h = 2), \delta_{cfe} \rangle : 1}, S(0))$

$= \boxed{\langle h = 2, \delta_{cfe} \rangle : 1} \otimes \boxed{\langle [S(1)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1}$

$= \boxed{\langle h = 2 \wedge [S(1)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1}$

And $\alpha Case_1^{S,O^{S(0)},1} = \boxed{\langle h \neq 2 \wedge [S(0)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1}$. By Steps 4 and 5, we have $V_E^1 = \{\alpha Case_1^{E,O^{E(1)},1}\}$ and

$V_S^1 = V_{S,S(1)}^1 \boxplus V_{S,S(0)}^1 = \begin{array}{|l} \langle h = 2 \wedge [S(1)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1 \\ \hline \langle h \neq 2 \wedge [S(0)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1 \end{array}$

Thus, $V^1 = V_E^1 \cup V_S^1$ contains two case statements above.

$V^1(\mathbf{b}_I) = \max\{\boxed{\langle h \leq 0 \wedge [E(1)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1}, \begin{array}{|l} \langle h \leq 0 \wedge h = 2 \wedge [S(1)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1 \\ \hline \langle h \leq 0 \wedge h \neq 2 \wedge [S(0)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1 \end{array}\}$

If we regress cases against $\boldsymbol{B}^{f_0}$ and $\mathcal{D}_{dyn}$, we have

$V^1(\mathbf{b}_I) = \max\{\boxed{\langle h \leq 0 \wedge \textsc{False}, \delta_{cfe} \rangle : 1}, \begin{array}{|l} \langle h \leq 0 \wedge h = 2 \wedge \textsc{True}, \delta_{cfe} \rangle : 1 \\ \hline \langle h < 2 \wedge \textsc{False}, \delta_{cfe} \rangle : 1 \end{array}\} = 0.$ The first case $\boxed{\langle h \leq 0 \wedge \textsc{False}, \delta_{cfe} \rangle : 1}$ is due to that starting from $\boldsymbol{B}^{f_0}$, after the stochastic action $east(1)$, the agent would believe $\boldsymbol{B}(h = 1 \colon 0.8)$ and $\boldsymbol{B}(h = 2 \colon 0.2)$, hence, $[E(1)]\boldsymbol{K}(h = 2)$ does not hold. Likewise $[S(1)]\boldsymbol{K}h = 2 \equiv \textsc{True}$ and $[S(0)]\boldsymbol{K}h = 2 \equiv \textsc{False}$. The result suggests that the maximal reachability is policies with at most 1 horizon is still 0.

For $H = 2$, observing the fact that $[east(1)]\boldsymbol{K}h = 2$ and $[east(0)]\boldsymbol{K}h = 2$ is impossible under $\boldsymbol{B}^{f_0}$ and $\mathcal{D}_{dyn}$, after eliminating inconsistent cases we have

$V_E^2 = \{\begin{array}{|l} \langle h = 1 \wedge [E(1) \cdot S(1)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 0.8 \\ \hline \langle h = 0 \wedge [E(2) \cdot S(1)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 0.2 \end{array}\}$

$V_S^2 = \{\begin{array}{|l} \langle h = 2 \wedge [S(1) \cdot S(1)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1 \\ \hline \langle h \neq 2 \wedge [E(0) \cdot S(0)]\boldsymbol{K}h = 2, \delta_{cfe} \rangle : 1 \end{array}\}$

Therefore the evaluation of $V^2(\mathbf{b}_I)$ results in

$V^2(\mathbf{b}_I) = \max\{0, \begin{array}{|l} \langle h = 0, \delta_{cfe} \rangle : 0.2 \\ \hline \langle h < 0, \delta_{cfe} \rangle : 0 \end{array}\}$. Hence, the maximal reachability is 0.2 if initially $h = 0$ (recall that $\mathcal{D}_0 = \{h \leq 0\}$), and it is obtained by the program (or policy) $east; sencfe$. Since $0.2 > 0.1$, the algorithm terminates with $\mathcal{BP}_{cfe} \not\models \mathsf{Pr}(\mathbf{F}\psi_{goal}) \leq 0.1$.

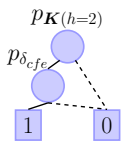
Theorem 1 *Given a belief program $\mathcal{BP}$, a goal ψ_{goal}, a threshold τ, the SDP algorithm above returns "$\mathcal{BP} \not\models \mathsf{Pr}(\mathbf{F}\psi_{goal}) \leq \tau$" iff $\mathcal{BP} \not\models \mathsf{Pr}(\mathbf{F}\psi_{goal}) \leq \tau$.*

Proof Sketch [9] proves the correctness of value iteration for reachability probability in POMDPs. Hence it remains to show that the

FO-POMDP for a belief program is a faithful abstraction of the underlying infinite ground POMDPs, which can be proved based on the facts that 1) every observation-based policy of the FO-POMDP is an observation-based policy of the ground POMDPs and vice versa; 2) every state $\langle \phi, \delta \rangle$ in FO-POMDP abstractly represents a set of program configurations (states of the ground POMDP) which satisfy ϕ.

4 Implementation & Evaluation

The main challenge in implementation is how to implement the case statement and operations over it. For this, we exploit the idea of *first-order algebraic decision diagram* (FOADD) in [35]. The algebraic decision diagram (ADD) [2, 18] is a data structure that compactly represents a Boolean function $\{0,1\}^n \mapsto \mathbb{R}$ using a directed acyclic graph. Nevertheless, the propositional nature prevents us from using ADDs directly since case statements contain general FOL formulas (potentially with modality $[a]$ and $\boldsymbol{B}$) and program expressions. Our solution is to maintain a mapping that maps each first-order sentence to a corresponding proposition (hence the name FOADD). In the same spirit, the mapping also maps each reachable sub-program $\delta_i \in Sub(\delta)$ to a proposition p_{δ_i} (mutually exclusive). Below is an illustration of the FOADD for $Case_0$ in Eq. (2).



With the FOADD, the operators $\oplus$ and $\otimes$ defined over case statements can be implemented by direct use of standard $Apply$ operations such as addition and multiplication for ADDs. A notable exception is the Pre function which requires a program graph G_δ and a ground action t_a as inputs. In practice, we process as follows:

1. updating the FOL-Propositions mapping by adding $[t_a]$ in front of all formulas (excluding $\delta_i \in Sub(\delta)$) in the given FOADD;
2. collecting edges $E := \{\delta_{i,j} \xrightarrow{\varrho/\beta_{i,j}} \delta_i \in G_\delta \text{ and } \varrho \to t_a\}$ whose label ϱ can instantiate the ground action t_a. Let $\Delta = \bigcup_i \{\delta_i\}$ be the set of relevant program expressions (whose predecessors need to be computed);
3. setting propositions $Sub(\delta) - \Delta$ to be FALSE;
4. replacing propositions $\delta_i \in \Delta$ with $\bigvee_j p_{\delta_{i,j}} \wedge p_{\beta_{i,j}}$.

We comment our notion of FOADD goes beyond the one by Sanner [35] as our mapping includes sentences that might contain modalities, even nested modalities.

Reducing space Several tricks can be used to reduce the number of introduced propositions.

1. For objective formulas, one could use regression to eliminate modality $[a]$ and identify potential equivalent formulas. E.g. under the BAT of coffee domain $\mathcal{D}_{dyn}$, the formula $h = 1$ is logically equivalent to $[east(1)]h = 2$;
2. Besides, one can rewrite terms [22] to identify potential equivalent formulas. E.g. we can rewrite $h + 1 + 1$ to $h + 2$;
3. For subjective formulas, since the agent has no knowledge about the actual outcome of stochastic actions, we could ignore their uncontrollable parameters. E.g. since $[east(1)]\boldsymbol{B}(\alpha\colon r)$ is equivalent to $[east(2)]\boldsymbol{B}(\alpha\colon r)$, we simply ignore arguments $\{1, 2\}$.

Although the contribution of the paper is mainly theoretical and conceptual as the primary focus is on converting SDP [36] for verification, we implemented the algorithm (with the above tricks) in a prototype system and evaluate it for the coffee robot domain. We conceive an extended evaluation for variants of domains, programs, and program properties in the future [12]. We exploit PREGO [7] for regression and the Z3 theorem prover [17] for satisfiability checking. Besides, Z3 provides built-in rules for term rewriting. The experiment is conducted on an Intel@ Core™ i5-1135G7 @2.4GHz with 8GB RAM. The table below demonstrates time and space consumption under different horizon H settings. "PRM" stands for the primitive version while "RDC" stands for the version of SDP that applies tricks to reduce space. The result suggests that while the tricks can significantly reduce the number of $\alpha Case$ in the value function (#V) and the number of propositions in ADDs (#Prop), especially when H grows large, it requires significantly more time to process. Unfortunately, both versions run out of memory for $H = 5$. This is not surprising since the set of α-vectors grows exponentially with every iteration [38]. After all, the exact value iteration could not solve the well-known Tiger domain for $H = 2$ with more than 1 door [36].

H	#V		#Prop		Time(in ms)	
	PRM	RDC	PRM	RDC	PRM	RDC
0	1	1	2	2	0	0
1	2	2	6	7	16	31
2	6	6	15	27	**5972**	35
3	**22**	42	**34**	106	**41613**	47
4	**132**	1806	**78**	422	**328472**	500

5 Related Work & Conclusion

We review related works from the perspective of GOLOG program verification and symbolic model-checking.

Our notion of belief programs stems from [31, 28]. A primitive form of belief programs (propositional beliefs) can be found in [25]. The first work on verifying GOLOG program is [15, 16]. Later, verifying variants of GOLOG programs and program properties are studied. E.g. [14] investigated the verification of DT-GOLOG program, a variant with decision theoretical feature, against PCTL-like properties. [43] studied the verification of $\mathcal{ALCOK}$-GOLOG programs, a variant with features of knowledge and sensing, against LTL-like properties [43]. Similarly, [13, 43, 44] resp. studied the verification of CTL*, LTL, and CTL properties of GOLOG programs. On verifying algorithms, a common step of many works above is to abstract the infinite ground transition systems (or MDP, POMDP) induced by program execution to abstract (symbolic) ones, and thereafter perform symbolic model-checking. If the abstractions are finite [43, 44, 14], tools such as PRISM [23] and Storm [21] can be used. Otherwise, one has to devise GOLOG-specific algorithms that, e.g. do similar fixpoint computations as (symbolic) model-checking techniques [13].

On the other hand, symbolic methods are common in model-checking to tackle the state space explosion problem [5]. In this regard, the most relevant work to us is [36] where an SDP algorithm is proposed for value iteration in FO-POMDP, nevertheless, states of FO-POMDP there only contain FOL formulas while ours include modality and program expressions as well, leading to special treatments in computing preimage and evaluating for case statements. Prior to [36], symbolic model-checking can be applied to transition systems [5], Markov chains [4], MDP [3], first-order MDP [35], or relational POMDP [41], stochastic game [24].

Future work includes exploring the possibility of integrating point-based value iteration [38] in verification. Besides, it would be desirable to provide also a lower bound on reachability probability [10].

Acknowledgements

This research was supported partly by a Royal Society University Research Fellowship, UK, partly by a grant from the UKRI Strategic Priorities Fund, UK to the UKRI Research Node on Trustworthy Autonomous Systems Governance and Regulation (EP/V026607/1, 2020–2024), partly by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) RTG 2236/2 'UnRAVeL', and partly by the EU ICT-48 2020 project TAILOR (No. 952215).

References

[1] Karl J Astrom, 'Optimal control of markov decision processes with incomplete state estimation', *J. Math. Anal. Applic.*, **10**, 174–205, (1965).

[2] R Iris Bahar, Erica A Frohm, Charles M Gaona, Gary D Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi, 'Algebric decision diagrams and their applications', *Formal methods in system design*, **10**, 171–206, (1997).

[3] Christel Baier, *On algorithmic verification methods for probabilistic systems*, Ph.D. dissertation, Habilitation thesis, Fakultät für Mathematik & Informatik, Universität Mannheim, 1998.

[4] Christel Baier, Edmund M Clarke, Vasiliki Hartonas-Garmhausen, Marta Kwiatkowska, and Mark Ryan, 'Symbolic model checking for probabilistic processes', in *Automata, Languages and Programming: 24th International Colloquium, ICALP'97 Bologna, Italy, July 7–11, 1997 Proceedings 24*, pp. 430–440. Springer, (1997).

[5] Christel Baier and Joost-Pieter Katoen, *Principles of model checking*, MIT press, 2008.

[6] Vaishak Belle and Gerhard Lakemeyer, 'Reasoning about probabilities in unbounded first-order dynamical domains.', in *IJCAI*, pp. 828–836, (2017).

[7] Vaishak Belle and Hector Levesque, 'Prego: an action language for belief-based cognitive robotics in continuous domains', in *AAAI*, volume 28, (2014).

[8] Vaishak Belle and Hector Levesque, 'Allegro: Belief-based programming in stochastic dynamical domains', in *IJCAI*, (2015).

[9] Alexander Bork, Sebastian Junges, Joost-Pieter Katoen, and Tim Quatmann, 'Verification of indefinite-horizon pomdps', in *ATVA*, pp. 288–304. Springer, (2020).

[10] Alexander Bork, Joost-Pieter Katoen, and Tim Quatmann, 'Under-approximating expected total rewards in pomdps', in *TACAS*, pp. 22–40. Springer, (2022).

[11] Anthony R Cassandra, Michael L Littman, and Nevin Lianwen Zhang, 'Incremental pruning: A simple, fast, exact method for partially observable markov decision processes', *arXiv preprint arXiv:1302.1525*, (2013).

[12] Jens Claßen, 'Symbolic verification of golog programs with first-order bdds', in *KR*, (2018).

[13] Jens Claßen and Gerhard Lakemeyer, 'A logic for non-terminating golog programs.', in *KR*, pp. 589–599, (2008).

[14] Jens Claßen and Benjamin Zarrieß, 'Decidable verification of decision-theoretic golog', in *International Symposium on Frontiers of Combining Systems*, pp. 227–243. Springer, (2017).

[15] Giuseppe De Giacomo, Eugenia Ternovska, and Ray Reiter, 'Non-terminating processes in the situation calculus', in *Proceedings of the AAAI'97 Workshop on Robots, Softbots, Immobots: Theories of Action, Planning and Control*, (1997).

[16] Giuseppe De Giacomo, Eugenia Ternovska, and Ray Reiter, 'Non-terminating processes in the situation calculus', *Annals of Mathematics and Artificial Intelligence*, **88**, 623–640, (2020).

[17] Leonardo De Moura and Nikolaj Bjørner, 'Z3: An efficient smt solver', in *Tools and Algorithms for the Construction and Analysis of Systems: 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings 14*, pp. 337–340. Springer, (2008).

[18] Frederik Gossen, Alnis Murtovi, Philip Zweihoff, and Bernhard Steffen, 'Add-lib: Decision diagrams in practice', *arXiv preprint arXiv:1912.11308*, (2019).

[19] Hans Hansson and Bengt Jonsson, 'A logic for reasoning about time and reliability', *Formal aspects of computing*, **6**, 512–535, (1994).

[20] Godfrey Harold Hardy, Edward Maitland Wright, et al., *An introduction to the theory of numbers*, Oxford university press, 1979.

[21] Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk, 'The probabilistic model checker storm', *International Journal on Software Tools for Technology Transfer*, 1–22, (2021).

[22] Gérard Huet, 'Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems', *Journal of the ACM (JACM)*, **27**(4), 797–821, (1980).

[23] M. Kwiatkowska, G. Norman, and D. Parker, 'PRISM 4.0: Verification of probabilistic real-time systems', in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, eds., G. Gopalakrishnan and S. Qadeer, volume 6806 of *LNCS*, pp. 585–591. Springer, (2011).

[24] Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos, 'Symbolic verification and strategy synthesis for turn-based stochastic games', in *Principles of Systems Design: Essays Dedicated to Thomas A. Henzinger on the Occasion of His 60th Birthday*, 388–406, Springer, (2022).

[25] Jérôme Lang and Bruno Zanuttini, 'Probabilistic knowledge-based programs', in *IJCAI*, (2015).

[26] Hector J Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin, and Richard B Scherl, 'Golog: A logic programming language for dynamic domains', *The Journal of Logic Programming*, **31**(1-3), 59–83, (1997).

[27] Fangzhen Lin and Ray Reiter, 'How to progress a database', *Artificial Intelligence*, **92**(1-2), 131–167, (1997).

[28] Daxin Liu, *Projection in a probabilistic epistemic logic and its application to belief-based program verification*, PhD dissertation, RWTH Aachen University, 2023.

[29] Daxin Liu and Qihui Feng, 'On the progression of belief', in *KR*, volume 18, pp. 465–474, (2021).

[30] Daxin Liu and Gerhard Lakemeyer, 'Reasoning about beliefs and meta-beliefs by regression in an expressive probabilistic action logic', in *IJCAI*, (2021).

[31] Daxin Liu and Gerhard Lakemeyer, 'On the verification of belief programs', *arXiv preprint arXiv:2204.12562*, (2022).

[32] Yongmei Liu and Ximing Wen, 'On the progression of knowledge in the situation calculus', in *IJCAI*, volume 22, p. 976, (2011).

[33] Gethin Norman, David Parker, and Xueyi Zou, 'Verification and control of partially observable probabilistic systems', *Real-Time Systems*, **53**, 354–402, (2017).

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

[35] Scott Sanner and Craig Boutilier, 'Practical solution techniques for first-order mdps', *Artificial Intelligence*, **173**(5-6), 748–788, (2009).

[36] Scott Sanner and Kristian Kersting, 'Symbolic dynamic programming for first-order pomdps', in *AAAI*, volume 24, pp. 1140–1146, (2010).

[37] Richard B Scherl and Hector J Levesque, 'Knowledge, action, and the frame problem', *Artificial Intelligence*, **144**(1-2), 1–39, (2003).

[38] Guy Shani, Joelle Pineau, and Robert Kaplow, 'A survey of point-based pomdp solvers', *Autonomous Agents and Multi-Agent Systems*, **27**, 1–51, (2013).

[39] Edward J Sondik, 'The optimal control of partially observable markov processes over the infinite horizon: Discounted costs', *Operations research*, **26**(2), 282–304, (1978).

[40] Alfred Tarski, *A decision method for elementary algebra and geometry*, Springer, 1998.

[41] Chenggang Wang and Roni Khardon, 'Relational partially observable mdps', in *AAAI*, volume 24, pp. 1153–1158, (2010).

[42] Wolfgang Woess, *Denumerable Markov chains*, Springer, 2009.

[43] Benjamin Zarrieß and Jens Claßen, 'Verification of knowledge-based programs over description logic actions.', in *IJCAI*, pp. 3278–3284, (2015).

[44] Benjamin Zarrieß and Jens Claßen, 'Decidable verification of golog programs over non-local effect actions.', in *AAAI*, pp. 1109–1115, (2016).