Reinforcement Learning in Partially Observable Markov Decision Processes using Hybrid Probabilistic Logic Programs

Emad Saad  
Department of Computer Science  
Gulf University for Science and Technology  
Mishref, Kuwait  
saad.e@gust.edu.kw

Abstract—We present a probabilistic logic programming framework to reinforcement learning, by integrating reinforcement learning in POMDP environments, with normal hybrid probabilistic logic programs with probabilistic answer set semantics, that is capable of representing domain-specific knowledge. We formally prove the correctness of our approach. We show that the complexity of finding a policy for a reinforcement learning problem in our approach is NP-complete. In addition, we show that any reinforcement learning problem can be encoded as a classical logic program with answer set semantics. We also show that a reinforcement learning problem can be encoded as a SAT problem. We present a new high level action description language that allows the factored representation of POMDP. Moreover, we modify the original model of POMDP so that it be able to distinguish between knowledge producing actions and actions that change the environment.

I. INTRODUCTION

Reinforcement learning is the problem of learning to act by trial and error interaction in dynamic environments. Reinforcement learning problems can be represented as Markov Decision Processes (MDP), under the assumption that accurate and complete model of the environment is known. This assumption requires the agent to have perfect sensing and observation abilities.

However, complete and perfect observability is unrealistic for many real-world reinforcement learning applications, although necessary for learning optimal policies in MDP environments. Therefore, different model is needed to represent and solve reinforcement learning problems with partial observability. This model is Partially Observable Markov Decision Processes (POMDP). Similar to MDP, POMDP requires the model of the environment to be known, however states of the world are not completely known. Consequently, the agent performs actions to make observations about the states of the worlds. These observations can be noisy due to imperfect agent’s sensors. Similar to MDP, dynamic programming methods, by value iteration, has been used to learn the optimal policy for a reinforcement learning problem in POMDP environment.

A logical framework to reinforcement learning in MDP environment has been developed in [30], which relies on techniques from probabilistic reasoning and knowledge representation by normal hybrid probabilistic logic programs (NHPLP) to solve reinforcement learning problems in MDP environment is based on that; NHPLP is nonmonotonic, therefore more suitable for knowledge representation and reasoning under uncertainty; NHPLP subsumes classical normal logic programs with classical answer set semantics [7], a rich knowl-edge representation and reasoning framework, and inherits its knowledge representation and reasoning capabilities including the ability to represent and reason about domain-specific knowledge; NHPLP has been shown applicable to a variety of fundamental probabilistic reasoning problems including probabilistic planning [28], contingent probabilistic planning [31], the most probable explanation in belief networks, the most likely trajectory in probabilistic planning, and Bayesian reasoning [29].

In this view, we integrate reinforcement learning in POMDP environment with NHPLP, providing a logical framework that overcomes the representational limitations of dynamic programming method to reinforcement learning in POMDP and is capable of representing its domain-specific knowledge. In addition, the proposed framework extends the logical framework of reinforcement learning in MDP of [30] with partial observability. We show that any reinforcement learning problem in POMDP environment can be encoded as a SAT
II. Syntax and Semantics of NHPLP

We introduce a class of NHPLP [34], namely NHPLP, that is sufficient to represent POMDP.

A. The Language of NHPLP

Let \( \mathcal{L} \) be a first-order language with finitely many predicate symbols, constants, and infinitely many variables. The Herbrand base of \( \mathcal{L} \) is denoted by \( \mathcal{B}_\mathcal{F} \). Probabilities are assigned to atoms in \( \mathcal{B}_\mathcal{F} \) as values from \([0,1]\). A normal probabilistic logic program (np-program) in NHPLP is a pair \( P = (R, \tau) \), where \( R \) is a finite set of normal probabilistic rules (np-rules) and \( \tau \) is a mapping \( \tau : \mathcal{B}_\mathcal{L} \rightarrow S_{\text{disj}} \) where \( S_{\text{disj}} \) is a set of disjunctive probabilistic strategies (p-strategies) whose composition functions \( c \) are mappings \( c : [0,1] \times [0,1] \rightarrow [0,1] \). A composition function of a disjunctive p-strategy returns the probability of a disjunction of two events given the probability values of its components.

An np-rule is an expression of the form
\[
A : \mu \leftarrow A_1 : \mu_1, \ldots, A_n : \mu_n,
\]
where \( A, A_1, \ldots, A_n, B_1, \ldots, B_m \) are atoms and \( \mu, \mu_1, \ldots, \mu_n \) are annotations. Let \( \alpha_1, \alpha_2 \in [0,1] \). Then we say that \( \alpha_1 < \alpha_2 \) iff \( \alpha_1 < \alpha_2 \). A normal probabilistic logic program (np-program) in NHPLP is a pair \( P = (R, \tau) \), where \( R \) is a finite set of normal probabilistic rules (np-rules) and \( \tau \) is a mapping \( \tau : \mathcal{B}_\mathcal{L} \rightarrow S_{\text{disj}} \) where \( S_{\text{disj}} \) is a set of disjunctive probabilistic strategies (p-strategies) whose composition functions \( c \) are mappings \( c : [0,1] \times [0,1] \rightarrow [0,1] \). A composition function of a disjunctive p-strategy returns the probability of a disjunction of two events given the probability values of its components.

A. Probabilistic Answer Set Semantics of NHPLP

A probabilistic interpretation (p-interpretation), \( \pi \), is a mapping from \( \mathcal{B}_\mathcal{L} \) to \([0,1] \). Let \( P = (R, \tau) \) be a ground np-program, \( h \) be a p-interpretation, and \( r \) be an np-rule as above. Then, we say
- \( h \) satisfies \( A_1 : \mu_i \) iff \( \mu_i \leq h(A_i) \).
- \( h \) satisfies not \( (B_j : \beta_j) \) iff \( \beta_j \leq h(B_j) \).
- \( h \) satisfies Body \( \equiv A_1 : \mu_1, \ldots, A_n : \mu_n, \) not \( (B_1 : \beta_1), \ldots, \) not \( (B_m : \beta_m) \) iff \( \forall (1 \leq i \leq n) \), \( h \) satisfies \( A_i : \mu_i \), and \( \forall (1 \leq j \leq m) \), \( h \) satisfies not \( (B_j : \beta_j) \).
- \( h \) satisfies \( A : \mu \leftarrow \) Body iff \( h \) satisfies \( A : \mu \) or \( h \) does not satisfy Body.
- \( h \) satisfies \( P \) iff \( h \) satisfies every np-rule in \( R \) and for every atom \( A \in \mathcal{B}_\mathcal{L} \), we have \( c_{\epsilon(A)} \{ \mu \} A : \mu \leftarrow \) Body \( \in R \) such that \( h \models \text{Body} \) \( \leq_i h(A) \).

The probabilistic reduct \( P^h \) of \( h \) w.r.t. \( h \) is an np-program without negation, \( P^h = \langle R^h, \tau \rangle \), where:
\[
A : \mu \leftarrow A_1 : \mu_1, \ldots, A_n : \mu_n, \not(B_1 : \beta_1), \ldots, \not(B_m : \beta_m) \in R
\]
and \( \forall (1 \leq j \leq m), \beta_j \leq h(B_j) \). A probabilistic model (p-model) of an np-program \( P \) is a p-interpretation of \( P \) that satisfies \( P \). We say that a p-interpretation \( h \) of \( P \) is a probabilistic answer set of \( P \) if \( h \) is the minimal p-model of the probabilistic reduct, \( P^h \), of \( P \) w.r.t. \( h \).

III. Partially Observable Markov Decision Processes

We review finite-horizon POMDP [12] with stationary transition functions, stationary bounded reward functions, and stationary policies.

A. POMDP Definition

POMDP is a tuple of the form \( M = (S, S_0, A, T, \lambda, R, \Omega, O) \) where: \( S \) is a finite set of states; \( S_0 \) is the initial state distribution; \( A \) is a finite set of stochastic actions; \( T \) is stationary transition function \( T : S \times A \times S \rightarrow [0,1] \), where for any \( s \in S \) and \( a \in A \), \( \sum_{s' \in S} T(s, a, s') = 1 \); \( \lambda \in [0,1] \) is the discount factor; \( R : S \times A \times S \rightarrow \mathbb{R} \) is a stationary bounded reward function; \( \Omega \) is a finite set of observations that the agent observes in the environment; and \( O \) is observation function \( O : S \times A \times S \rightarrow [0,1] \), where for any \( s \in S \) and \( a \in A \) where \( \sum_{o \in \Omega} O(s, a, o) = 1 \). A stationary policy is a mapping from states to actions of the form \( \pi : S \rightarrow A \). The value function of a policy \( \pi \) with respect to an initial state \( s_0 \in S_0 \), with finite horizon of \( n \) steps remaining, \( V^\pi_n (s_0) \), is calculated by
\[
V^\pi_n (s_0) = \sum_{s_1 \in S} T(s_0, \pi(s_0), s_1) \sum_{o_1 \in \Omega} O(s_1, \pi(s_0), o_1) [R(s_0, \pi(s_0), s_1) + \lambda V^\pi_{n-1} (s_1)]
\]
which determines the expected sum of discounted rewards resulting from executing the policy \( \pi \) starting from \( s_0 \). Because of the agent is unable to completely observe the states of the world and with reliability, it keeps what is called a belief state. An agent's belief state is a probability distribution over the possible world states the agent may think it is in. Therefore, an action causes a transition from a belief state to another belief state. Given \( b \) is a believe state and \( a \) is an action, then executing \( a \) in the belief state \( b \) results a new belief state \( b' \), where the probability of a state, \( s' \), in \( b' \) and the value function of executing a policy \( \pi \) in \( b \) are given by:
\[
b'(s') = \frac{O(s', a, o) \sum_{s \in S} T(s, a, s') h(s)}{Pr(o|a, b)}
\]
\[
V^\pi_n (b) = \sum_{s \in S} b(s) V^\pi_{n-1} (s).
\]
The optimal policy over the agent's belief states can be constructed from the optimal value function over the agent's belief states which is given by $V^*(b) = \max_{\pi} V_{\pi}^*(b)$.

B. Discussion

The original model of POMDP does not distinguish between knowledge producing (sensing) actions and actions that affect and change the environment (non-sensing actions). This means that it treats sensing and non-sensing actions equally in the sense that, like non-sensing actions, a sensing action affects and changes the environment as well as producing knowledge resulting from observing the environment. However, [36] proved that sensing actions produce knowledge (make observations) and does not change the state of the world. Therefore, actions that change the state of the world are different from the knowledge producing actions. In addition, the value function described above makes the agent observing the environment at every step of its life with each action it takes. However, this is not necessary to be always the case, since it is possible for the agent to start with observing the environment then performing a sequence of actions, or the agent could start with performing a sequence of actions then observing the environment. To overcome these limitations, we define the value function of n-step finite horizon POMDP with respect to an initial state $s_0 \in S_o$ as:

- If $\pi(s_0)$ is a non-sensing action
  
  $$V^*_n(s_0) = \sum_{s_1 \in S} T(s_0, \pi(s_0), s_1)[R(s_0, \pi(s_0), s_1) + \lambda V^*_{n-1}(s_1)]$$

- If $\pi(s_0)$ is sensing action
  
  $$V^*_n(s_0) = \sum_{s_1 \in S} O(s_0, \pi(s_0), s_1)[R(s_0, \pi(s_0), s_1) + \lambda V^*_{n-1}(s_1)]$$

where $O(s_0, \pi(s_0), s_1)$ is the probability of observing the state $s_1$, where for some $o \in \Omega$, $o$ is observed in $s_1$. Notice that $O$ is treated as a mapping $O : S \times A \times S \rightarrow [0, 1]$, where $A$ is the set of sensing actions. For any $s \in S$ and $a \in A$, $O(s, a, a')$ is the probability distribution over states resulting from executing $a$ in $s$, such that $\sum_{s' \in S} O(s, a, s') = 1$. As in the original model of POMDP, $T$ is a mapping $T : S \times A \times S \rightarrow [0, 1]$, where $A$ is the set of non-sensing actions. Extension to infinite horizon POMDP can be achieved in a similar manner. This definition of POMDP distinguishes between knowledge producing actions and actions that change the environment. In this view, the optimal policy $V^*_n$ is given by: $V^*_n(s_0) = \max_{\pi} V_{\pi}^*(s_0)$

IV. $\mathcal{A}_{DC}$ AN ACTION LANGUAGE FOR POMDP

We introduce an action language for POMDP, $\mathcal{A}_{DC}$. The proposed action language extends both the action language, $\mathcal{A}_{MDP}$, [30] for representing and reasoning about MDP, and the action language, $\mathcal{A}$, [31] for representing and reasoning about imperfect sensing actions with probabilistic outcomes. An action theory in $\mathcal{A}_{DC}$ is capable of representing the initial state distribution, the executability conditions of actions, the discount factor, the reward received from executing actions in states, and makes it clear the distinction between sensing and non-sensing actions.

A. Language syntax

A fluent is a predicate, which may contain variables. Given that $\mathcal{F}$ is a set of fluents and $\mathcal{A}$ is a set of actions that can contain variables, a fluent literal is either a fluent $F \in \mathcal{F}$ or $\neg F$. A conjunction of fluent literals of the form $l_1 \wedge \ldots \wedge l_n$ is conjunctive fluent formula, where $l_1, \ldots, l_n$ are fluent literals. Sometimes we abuse the notation and refer to a conjunctive fluent formula as a set of fluent literals ($\emptyset$ denotes true). An action theory, $\mathbf{PT}$, in $\mathcal{A}_{DC}$ is a tuple $\mathbf{PT} = \langle S_0, D, \lambda \rangle$, where $S_0$ is a proposition of the form (1), $D$ is a set of propositions from (2)-(4), and $0 \leq \lambda < 1$ is a discount factor as follows:

| Initially | Executable A if \psi | \begin{align*} \alpha & \text{ causes } \phi_i : p_i : r_i \text{ if } \psi, \quad 1 \leq i \leq n \\ \alpha & \text{ observes } \psi_i : p_i : r_i \text{ if } \psi, \quad 1 \leq i \leq n \end{align*} |}

| Initially | Executable A if \psi | \begin{align*} \alpha & \text{ causes } \phi_i : p_i : r_i \text{ if } \psi, \quad 1 \leq i \leq n \\ \alpha & \text{ observes } \psi_i : p_i : r_i \text{ if } \psi, \quad 1 \leq i \leq n \end{align*} |

Example 1: Consider the tiger domain from [20]. A tiger is behind left (l) or tight (−l) door with equal probability 0.5. If left door is opened and (l), punishment of -100 is received, but a reward of 10 is received if (−l) and the other way around. The sensing action listen used for hearing the tiger behind left door (htl), a correlated property to tl, then it reports tl with 0.85 and erroneously reports −tl with 0.15. Similarly for listening to the right door. This is represented by the action theory $\mathbf{PT} = \langle S_0, D, \lambda \rangle$, where

$S_0 = \begin{cases} \{tl, htl\} : 0.5 \\ \{−tl, −htl\} : 0.5 \end{cases}$
B. Semantics

A set of ground literals $\phi$ is consistent if it does not contain a pair of complementary literals. If a literal $l$ belongs to $\phi$, then we say $l$ is true in $\phi$, and $\lnot l$ is false in $\phi$ if $l$ is in $\phi$. A set of literals $\phi$ is true in $s$ if $\phi$ is contained in $s$. A state is a complete and consistent set of literals that describes the world at a certain time point.

**Definition 2:** Let $PT = \{S_0, D, \lambda\}$ be a ground action theory in $A_{PT}$, $s$ be a state, $a_i$ causes $\phi_i : p_i : r_i$ if $\psi_i (1 \leq i \leq n)$ be in $D$, and $a = \{a_1, \ldots, a_n\}$ be an action, where each $a_i$ corresponds to $\phi_i, p_i, r_i$, and $\psi_i$ for $1 \leq i \leq n$ (similarly for $a_i$ observes $\phi_i : p_i : r_i$, sensing $\psi_i$). Then, the state resulting from executing $a$ in $s$ $\Phi(a_i, s)$ is:

- $l \in \Phi(a_i, s)$ and $\lnot l \notin \Phi(a_i, s)$ if $l \in a_i$ and $\psi_i \subseteq s$.
- $\lnot l \notin \Phi(a_i, s)$ and $l \notin \Phi(a_i, s)$ if $\lnot l \in a_i$ and $\psi_i \subseteq s$.
- Else $l \in \Phi(a_i, s)$ if $l \in s$ and $\lnot l \notin \Phi(a_i, s)$ if $l \notin s$.

**Definition 2:** Let $s$ be a state, and $a_i$ causes $\phi_i : p_i : r_i$ if $\psi_i (1 \leq i \leq n)$ be in propositions. Then, the transition probability distribution after executing $a$ in $s$ $\Phi(a_i, s)$ is:

$$T(s, a, s') = \begin{cases} p_i & \text{if } s' = \Phi(a_i, s) \\ 0 & \text{otherwise} \end{cases}$$

$$O(s, a', s') = \begin{cases} p'_i & \text{if } s' = \Phi(a'_i, s) \\ 0 & \text{otherwise} \end{cases}$$

The reward received in a state $s$ after executing $a$ in $s$ is $\mathcal{R}(s, a, s') = r_i$ if $s' = \Phi(a_i, s)$, otherwise $\mathcal{R}(s, a', s') = r'_i$ if $s' = \Phi(a'_i, s)$.

**Definition 3:** Let $s_0$ be an initial state, $s, s'$ be states, and $\pi$ be a policy in $PT$. Then, the value function of $n$-step remaining, $V^\pi_n(s_0)$, of $\pi$ is given by:

- If $\pi(s_0)$ is a non-sensing action and $X = T(s_0, \pi(s_0), s_1)$

$$V^\pi_n(s_0) = \mathcal{R}(s_0, \pi(s_0), s_1) + \lambda \sum_{s_{i+1} \in S} V^\pi_{n-1}(s_{i+1})$$

- If $\pi(s)$ is a sensing action and $Y = O(s_0, \pi(s_0), s_1) V^\pi_n(s_0)$

$$V^\pi_n(s_0) = \mathcal{R}(s_{n-1}, \pi(s_{n-1}), s_n)$$

Executing sensing or non-sensing action, $\pi(s)$, in $s$ causes a transition to a set of states, $\sigma = \{s'_1, s'_2, \ldots, s'_m\}$. Let $\pi(\sigma)$ denotes the set of actions $\pi(s'_1), \pi(s'_2), \ldots, \pi(s'_m)$ executed in the states $s'_1, s'_2, \ldots, s'_m$, respectively. Notice that if $\pi(\sigma)$ is a singleton, i.e., the same action is executed in every state in $\sigma$, then this corresponds to executing an action in a belief state $\sigma = \{s'_1, s'_2, \ldots, s'_m\}$. Since executing $\pi(\sigma)$ in $\sigma$ produces another set of states $\sigma'$, then executing $\pi(\sigma)$ causes a transition from a belief state to another belief state.

For finite horizon POMDP, a policy $\pi : S \rightarrow A$ can be represented as a set of ordered pairs, starting from the initial belief state $\sigma_0$ (the set of initial states in $S_0$), as $\pi = \{\sigma_0, \pi(\sigma_0), \sigma_1, \pi(\sigma_1), \ldots, \sigma_n, \pi(\sigma_n)\}$ where for $1 \leq i \leq n$, $\sigma_i$ represents a belief state (a set of states) resulting from executing $\pi(\sigma_{i-1})$ in $\sigma_{i-1}$. This set representation of finite horizon policies in POMDP leads to view a policy as a set of trajectories, where each trajectory is $j(n) \equiv s_0, \pi(s_0), s_1, \pi(s_1), \ldots, s_{n-1}, \pi(s_{n-1}), s_n$ where $S_0$ is an initial state in $S_0$ and for all $1 \leq i \leq n$, $s_i = \Phi(s_{i-1}, \pi(s_{i-1}))$. Let $\pi$ be a policy for a finite horizon POMDP and $T_\pi$ be the set of trajectories representation of $\pi$, given the trajectory view of $\pi$, the value function of $\pi$ can be now described as:

$$V^\pi_n(s_0) = \sum_{i(n) \in T_\pi} \sum_{l=0}^{n-1} \lambda^l \prod_{i=0}^{l} X(s_i, \pi(s_i), s_{i+1}) \mathcal{R}(s_l, \pi(s_l))$$

where

$$X(s_i, \pi(s_i), s_{i+1}) = \begin{cases} T(s_i, \pi(s_i), s_{i+1}), \pi(s_i) \text{ is nonsensing} \\ O(s_i, \pi(s_i), s_{i+1}), \pi(s_i) \text{ is sensing} \end{cases}$$

Thus, the optimal policy $V^\pi_n$, the maximum value function among all policies, is given by $V^\pi_n(s_0) = \max_\pi V^\pi_n(s_0)$.

V. REINFORCEMENT LEARNING IN NHPLP

This section uses NHPLP to solve reinforcement learning problems, by encoding an action theory, $PT$ in $A_{PT}$, into an np-program, $\Pi_{PT}$. The probabilistic answer sets of $\Pi_{PT}$ correspond to valid trajectories in $PT$ with associated value function. The np-program encoding of an action theory in $A_{PT}$ follows related encoding described in [30], [31], [37]. We assume that the length of the optimal policy that we are looking for is known and finite. We use the following predicates: holds($L, T$) for literal $L$ holds at time moment $T$, occ($A, T$) for action $A$ executes at time $T$, state($T$) for a state of the world at time $T$, reward($T, r$) for the reward received at time $T$ is $r$, value($T, V$) for the value function of a state at time $T$ is $V$, and factor($\lambda$) for the discount factor $\lambda$. If an atom appears in an np-rule in $R$ with no annotation it is assumed to be associated with the annotation 1. We use $p(\psi)$ to denote $p(l_1), \ldots, p(l_m)$ for $p$ is a predicate and $\psi = \{l_1, \ldots, l_m\}$. Let $\Pi_{PT} = \langle R, \pi \rangle$ be the np-program encoding $PT = \{S_0, D, \lambda\}$, where $R$ is the set of the following np-rules.

- Each action $a = \{a_1, \ldots, a_n\} \in A$, is encoded as for all $1 \leq i \leq n$, action($a_i$) ← (6)

Each fluent $f \in F$ is encoded as a fact of the form

$$\text{fluent}(f) \leftarrow$$

Fluent literals are encoded as

$$\text{literal}(F) \leftarrow \text{fluent}(F)$$

$$\text{literal}(\lnot F) \leftarrow \text{fluent}(F)$$

To specify that fluents $F$ and $\lnot F$ are contrary literals, we use the following np-rules.

- $$\text{contrary}(F, \lnot F) \leftarrow \text{fluent}(F), \text{fluent}(\lnot F)$$
The initial belief state initially \( \{ \psi_i : p_i, 1 \leq i \leq n \} \) is represented in \( R \) as follows. Let \( s_1, s_2, \ldots, s_o \) be the set of possible initial states, where for each \( 1 \leq i \leq n \), \( s_i = \{ l^i_1, \ldots, l^i_m \} \), and the initial probability distribution be \( Pr(s_i) = p_i \). Moreover, let \( s = s_1 \cup s_2 \cup \ldots \cup s_o \), \( s' = s_1 \cap s_2 \cap \ldots \cap s_o \), \( \hat{s} = s - s' \). Let \( s^{report} = \{ l | l \in s_i \text{ and } l \text{ is a sensor report literal} \} \) be the set of all sensor report literals in all \( s_i \). Let \( s'' = \{ l | l \in (\hat{s} - s^{report}) \lor -l \in (\hat{s} - s^{report}) \} \). Intuitively, \( s'' \) is the same as \( \hat{s} \) after excluding the set of sensor report literals \( s^{report} \) from \( \hat{s} \). Let \( s^{sense} \) be the set of all pairs \( (\delta_i, \gamma_i) \), where \( \delta_i \) and \( \gamma_i \) are sets of literals contained in \( s_i \), such that \( \delta_i \) is the set of sensor reading literals and \( \gamma_i \) is the set of sensor report literals appearing in \( s_i \). The set of all possible initial states are generated as follows: for each \( l \in s' \), we include in \( R \)
\[
\text{holds}(l, 0) \leftarrow (11)
\]
which represents a fact that holds in every possible initial state. It says that the literal \( l \) holds at time moment 0. In addition, for each \( l \in s'' \), \( R \) includes
\[
\text{holds}(l, 0) \leftarrow \lnot \text{not holds}(l, 0) (12)
\]
\[
\text{holds}(\lnot l, 0) \leftarrow \lnot \text{not holds}(l, 0) (13)
\]
These np-rules say \( l \) (similarly \( \lnot l \)) holds at time moment 0, if \( \lnot l \) (similarly \( l \)) does not hold at time moment 0. For each \( (\delta, \gamma) \in s^{sense} \), let \( \gamma = \{ l_1, \ldots, l_m \} \), then for each \( 1 < i < m \), \( R \) includes
\[
\text{holds}(l_i, 0) \leftarrow \text{holds}(\delta, 0) (14)
\]
The initial probability distribution over the initial states is encoded as follows, which says that the probability of a state at time 0 is \( p_i \), if \( l_1, \ldots, l_m \) hold at the time 0.
\[
\text{state}(0) : p_i \leftarrow \text{holds}(l_1^i, 0), \ldots, \text{holds}(l_m^i, 0) (15)
\]
• Each executability condition of an action of the form (2) is encoded for each \( 1 \leq i \leq n \) as
\[
\text{exec}(a_i, T) \leftarrow \text{holds}(\psi_i, T) (16)
\]
• For each non-sensing action proposition \( a; \text{causes} \phi_i ; p_i ; r_i \) if \( \psi_i, 1 \leq i \leq n \) in \( D \), let \( \phi_i = \{ l_1^i, \ldots, l_m^i \} \). Then, \( \forall (1 \leq j \leq m), R \) includes
\[
\text{holds}(l_j^i, T + 1) \leftarrow \text{occ}(a_i, T), \text{exec}(a_i, T), \text{holds}(\psi_i, T) (17)
\]
If \( \alpha \) occurs at time \( T \) and \( \psi_i \) holds at the same time moment, then the \( l_j^i \) holds at the time \( T + 1 \). Then, we have
\[
\text{state}(T + 1) : p_i \times U \leftarrow \text{state}(T) : U, \text{occ}(a_i, T), \text{exec}(a_i, T), \text{holds}(\psi_i, T), \text{holds}(\phi_i, T + 1) (18)
\]
where \( U \) is an annotation variable ranging over \( [0, 1] \) acts as a place holder. This np-rule states that if \( \psi_i \) holds in a state at time \( T \), whose probability is \( U \), and in which \( \alpha \) is executable, then the probability of a successor state at time \( T + 1 \) is \( p_i \times U \), in which \( \phi_i \) holds.

• For each sensing action proposition \( a; \text{observes} \phi_i ; p_i ; r_i ; \text{sensing} \psi_i, 1 \leq i \leq n \) in \( D \), let \( \alpha_i = \{ l_1^i, \ldots, l_m^i \} \) and \( \psi_i = \{ l_1^i, \ldots, l_m^i \} \). Then, \( \forall (1 \leq j \leq m), R \) includes
\[
\text{observed}(l_j^i, T) \leftarrow \text{occ}(a_i, T), \text{exec}(a_i, T), \text{holds}(\psi_i, T) (19)
\]
\[
\text{holds}(l_j^i, T + 1) \leftarrow \text{occ}(a_i, T), \text{exec}(a_i, T), \text{observed}(\psi_i, T) (20)
\]
where first p-rule says that executing the sensing action \( \alpha \) at time \( T \) in which \( \psi_i \) holds causes \( \psi_i \) to be observed to be known true at the same moment \( T \), and second p-rule states that if \( \phi \) occurs at time \( T \) and the literals in \( \psi_i \) are observed to be known true at the same moment, then the literals \( l_j^i \) are known to hold at the time moment \( T + 1 \).
\[
\text{state}(T + 1) : p_i \times U \leftarrow \text{state}(T) : U, \text{occ}(a_i, T), \text{exec}(a_i, T), \text{observed}(\psi_i, T), \text{holds}(\alpha_i, T + 1) (21)
\]
The above np-rule says that the probability of a state at time \( T + 1 \) is \( p_i \times U \) if \( \alpha_i \) becomes known true at the same moment, after executing \( \alpha \) in a state at time \( T \), whose probability is \( U \), in which the literals in \( \psi_i \) are observed true.

• The reward \( r \), received at time \( T + 1 \) after executing \( \alpha \) in a state at time \( T \) is encoded as
\[
\text{reward}(r_i, T + 1) \leftarrow \text{occ}(a_i, T), \text{exec}(a_i, T) (22)
\]
• The value function \( T + 1 \) steps away from the initial state given the value function \( T \) steps away from the initial state in a given episode is encoded as
\[
\begin{align*}
\text{-- if } \alpha & \text{ is a non-sensing action} \\
\text{value}(V + \lambda^T \ast U \ast r_i, T + 1) & \leftarrow \text{value}(V, T), \text{factor}(\lambda), \text{state}(T + 1) : U, \text{reward}(r_i, T + 1), \text{occ}(a_i, T), \text{exec}(a_i, T), \text{holds}(\psi_i, T), \\
& \text{holds}(\phi_i, T + 1) (23)
\end{align*}
\]
\[
\begin{align*}
\text{-- if } \alpha & \text{ is a sensing action} \\
\text{value}(V + \lambda^T \ast U \ast r_i, T + 1) & \leftarrow \text{value}(V, T), \text{factor}(\lambda), \text{state}(T + 1) : U, \text{reward}(r_i, T + 1), \\
& \text{occ}(a_i, T), \text{exec}(a_i, T), \text{observed}(\psi_i, T), \\
& \text{holds}(\alpha_i, T + 1) (24)
\end{align*}
\]
where the variables \( V \in \mathbb{R}, \lambda \in [0, 1], U \in [0, 1], \) and \( \text{factor}(\lambda) \) is a fact in \( R \). These np-rules state that the value function at time \( T + 1 \) is equal to the value function at time \( T \) added to the product of the reward \( r \) received in a state at \( T + 1 \) and the probability of a state at time \( T + 1 \) discounted by \( \lambda^T \).

• The following np-rule asserts that a literal \( I \) holds at time \( T + 1 \) if it holds at \( T \) and its contrary does not hold at \( T + 1 \).
\[
\text{holds}(L, T + 1) \leftarrow \text{holds}(L, T), \not \text{not holds}(L', T + 1), \text{contrary}(L, L') (25)
\]
The literal, $A$, and its negation, $\neg A$, cannot hold at the same time, where inconsistent is a literal that does not appear in $PT$.

\[
inconsistent \leftarrow \neg inconsistent, \ holds(A,T), \ holds(\neg A,T) \quad (26)
\]

- Actions are generated once at a time by the np-rules:

\[
occ(AC^i, T) \leftarrow action(AC^i), \ not \ abocc(AC^i, T) \quad (27)
\]

\[
abocc(AC^i, T) \leftarrow action(AC^i), action(AC^{\neg i}),
occ(AC^{\neg i}, T), AC^i \neq AC^{\neg i} \quad (28)
\]

- The goal expression $G = g_1 \wedge \ldots \wedge g_m$ is encoded as

\[
good \leftarrow holds(g_1, T), \ldots, holds(g_m, T) \quad (29)
\]

VI. CORRECTNESS

In this section we prove that the probabilistic answer sets of the np-program encoding of an action theory, $PT$, correspond to trajectories in $PT$, with associated value function. Moreover, we show that the complexity of finding a policy for $PT$ in our approach is NP-complete.

Let the domain of $T$ be $\{0, \ldots, n\}$. Let be a $T$ transition function associated with $PT$, $s_0$ be a possible initial state, and $a_0, \ldots, a_{n-1}$ be a set of actions in $A$. Recall, any action $a_i$ can be represented as $a_i = \{a_1, \ldots, a_n\}$. Therefore, a trajectory $s_0, s_0, s_1, s_1, \ldots, s_{n-1}, s_{n-1}$ in $PT$ can be also represented as $s_0, a_{j_0}, s_1, a_{j_1}, \ldots, s_{n-1}, s_{n-1}$ for $(1 \leq j \leq m)$ and $(0 \leq i \leq n)$, such that $v(0 \leq i \leq n), s_i$ is a state, $a_i$ is an action, $a_{j_i} \in a_i = \{a_1, \ldots, a_n\}, a_{j_i} = \pi(s_i)$, and $s_i = F(a_{j_{i-1}}), s_{i-1}$.

Theorem 1: Let $PT$ be an action theory in $\mathcal{A}_{\mathcal{P}C}'$, $\pi$ be a policy in $PT$, and $T_\pi$ be the set of trajectories in $\pi$. Then, $s_0, s_0, s_1, s_1, \ldots, s_{n-1}, s_{n-1}$ is a trajectory in $T_\pi$ iff $occ(\pi(s_0), 0), \ldots, occ(\pi(s_{n-1}), n - 1)$ is true in a probabilistic answer set of $\Pi_{PT}$. Intuitively, an action theory, $PT$ in $\mathcal{A}_{\mathcal{P}C}'$, can be encoded to an np-program, $\Pi_{PT}$, whose probabilistic answer sets correspond trajectories in $PT$.

Theorem 2: Let $h$ be a probabilistic answer set of $\Pi_{PT}$, $\pi$ be a policy in $PT$, and $T_\pi$ be the set of trajectories in $\pi$. Let $occ$ be a set that contains $h \models \tau = occ(\pi(s_0), 0), \ldots, occ(\pi(s_{n-1}), n - 1)$ iff $s_0, s_0, s_1, s_1, \ldots, s_{n-1}, s_{n-1}, \pi(s_{n-1}), s_n \in T_\pi$. Then, $\sum_n value(v, n) h = =_{\mathcal{O}C} v = V_\pi(s_0)$.

Theorem (2) states that the summation of the values $v$, appearing in $value(v, n)$ that is satisfied by a probabilistic answer set $h$ in which $occ(\pi(s_0), 0), \ldots, occ(\pi(s_{n-1}), n - 1)$ is satisfied is equal to the expected sum of discounted rewards after executing a policy $\pi$ starting from a state $s_0$.

Reinforcement learning problems in POMDP can also be encoded as classical normal logic programs with classical answer set semantics [7]. Excluding the np-rules (15), (18), (21) -- (24) from the np-program encoding, $\Pi_{PT}$, of $PT$, results np-program, denoted by $\Pi_{PT}'$, with only annotations of the form 1. As shown in [34], the syntax and semantics of this class of np-programs is equivalent to classical normal logic programs with classical answer set semantics.

Theorem 5: Let $Pi_{PT}'$ be the normal logic program resulting after deleting the np-rules (15), (18), (21) -- (24) from $\Pi_{PT}$ and $\pi$ be a policy in $PT$. Then, $s_0, s_1, \ldots, s_{n-1}, s_n$ is a trajectory in $\pi$ iff $occ(\pi(s_0), 0), \ldots, occ(\pi(s_{n-1}), n - 1)$ is true in an answer set of $\Pi_{PT}'$.

Theorem 6: Let $\Pi_{PT}'$ be the normal logic program encoding of a reinforcement learning problem in POMDP environment using NHLP\textsubscript{P}, with probabilistic answer set semantics is NP-complete.

VII. REINFORCEMENT LEARNING USING ANSWER SET PROGRAMMING

Reinforcement learning problems in POMDP can be also encoded as classical normal logic programs with classical answer set semantics [7]. Excluding the np-rules (15), (18), (21) -- (24) from $\Pi_{PT}$, of $PT$, results np-program, denoted by $\Pi_{PT}'$, with only annotations of the form 1. As shown in [34], the syntax and semantics of this class of np-programs is equivalent to classical normal logic programs with classical answer set semantics.

Theorem 5: Let $\Pi_{PT}'$ be the normal logic program resulting after deleting the np-rules (15), (18), (21) -- (24) from $\Pi_{PT}$ and $\pi$ be a policy in $PT$. Then, $s_0, s_1, \ldots, s_{n-1}, s_n$ is a trajectory in $\pi$ iff $occ(\pi(s_0), 0), \ldots, occ(\pi(s_{n-1}), n - 1)$ is true in an answer set of $\Pi_{PT}'$.

Theorem 6: Let $\Pi_{PT}'$ be the normal logic program encoding of $PT$, $\pi$ be a policy in $PT$, and $T_\pi$ be the set of trajectories in $\pi$. Then, $\Pi_{PT}'$ can be translated into an equivalent SAT formula, where the models of $\Pi_{PT}'$ correspond to valid trajectories in $PT$.

Theorem 5 shows that classical normal logic programs with answer set semantics can be used to solve reinforcement learning problems in POMDP in two steps. First, a reinforcement learning problem, $PT$, is encoded to a classical normal logic program whose answer sets correspond valid trajectories in $PT$. From the answer sets of the normal logic program encoding of $PT$, we can determine the set of trajectories $T_\pi$ for a policy $\pi$ in $PT$. Second, the value of the policy $\pi$ is calculated using (5). Moreover, any reinforcement learning problem in POMDP environment can be encoded as a SAT problem. Hence, state-of-the-art SAT solvers can be used to solve reinforcement learning problems. Any normal logic program, $\Pi$, can be translated into a SAT formula, $S$, where the models of $S$ are equivalent to the answer sets of $\Pi$ [19]. Therefore, the normal logic program encoding of a reinforcement learning problem $PT$ can be translated into an equivalent SAT formula, where the models of $S$ correspond to valid trajectories in $PT$.

Theorem 6: Let $\Pi_{PT}'$ be the normal logic program encoding of $PT$, $\pi$ be a policy in $PT$, and $T_\pi$ be the set of trajectories in $\pi$. Then, $\Pi_{PT}'$ can be translated into an equivalent SAT formula, where the models of $S$ correspond to valid trajectories in $PT$.

Reinforcement learning problems can be directly encoded to SAT [32]. This is shown by following corollary.
Corollary 1: Let $PT$ be an action theory. Then, $PT$ can be directly encoded as a SAT formula $S$ where the models of $S$ are equivalent to valid trajectories in $PT$.

VIII. CONCLUSIONS AND RELATED WORK

We described a new high level action language, $A_{POMDP}$, that allows the factored representation of POMDP. Moreover, we presented a new reinforcement learning framework by relating reinforcement learning in POMDP to NHPLP. The translation from an action theory representation of a reinforcement learning problem in $A_{POMDP}$ into an NHPLP program is based on a similar translation from probabilistic planning into NHPLP [28]. The difference between $A_{POMDP}$ and the action languages [1], [2], [5], [11], and [17] is that $A_{POMDP}$ is a high level language and allows the factored specification of POMDP.

The approaches for solving POMDP to find the optimal policies can be categorized into two main approaches: dynamic programming approaches and the search-based approaches (a detailed survey on these approaches can be found in [2]). However, dynamic programming approaches use primitive domain knowledge representation. Moreover, the search-based approaches mainly rely on search heuristics which have limited knowledge representation capabilities to represent and use domain-specific knowledge.

In [22], a logical approach for solving POMDP, for probabilistic contingent planning, has been presented which converts a POMDP specification of a probabilistic contingent planning problem into a stochastic satisfiability problem and solving the stochastic satisfiability problem instead. Our approach is similar in spirit to [22] in the sense that both approaches are logic based approaches. However, it has been shown in [29] that NHPLP is more expressive than stochastic satisfiability from the knowledge representation point of view. In [15], based on first-order logic programs without nonmonotonic negation, a first-order logic representation of MDP has been described. Similar to the first-order representation of MDP in [15], $A_{MDP}$ allows objects and relations. However, unlike $A_{POMDP}$, [15] finds policies in the abstract level. But, NHPLP allows objects and relations. [3] presented a more expressive first-order representation of MDP than [15] that is a probabilistic extension to Reiter’s situation calculus. However, it is more complex than [15].

REFERENCES

[1] C. Baral, N. Tran, L. C. Tuan. Reasoning about actions in a probabilistic setting. In AAAI, 2002.
[2] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: structural assumptions and computational leverage. Journal of AI Research, 11:1–94, 1999.
[3] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In 17th IJCAI, 2001.
[4] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In Proc. of the 2nd International Conference on Artificial Intelligence Planning Systems, pages 31–37, 1994.
[5] T. Eiter and T. Lukasiewicz. Probabilistic reasoning about actions in nonmonotonic causal theories. In 19th Conference on Uncertainty in Artificial Intelligence, 2003.
[6] T. Eiter et al. Declarative problem solving in dlv. In Logic Based Programming.
[7] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. ICLP, 1988, MIT Press.
[8] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. New Generation Computing, 9(3–4):363–385, 1991.
[9] M. Gelfond and V. Lifschitz Representing action and change by logic programs. Journal of Logic Programming, 17:301–321, 1993.
[10] S. Holiholde, E. Karabaev, and O. Skvortsova. FluCap: A heuristic search planner for first-order MDPs. JAIR, 27:419–439, 2006.
[11] L. Iocchi, T. Lukasiewicz, D. Nardi, and R. Rosati. Reasoning about actions with sensing under qualitative and probabilistic uncertainty. In 16th European Conference on Artificial Intelligence, 2004.
[12] L. Kaelbling, M. Littman, and A. Cassandra. Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101: 99–134, 1998.
[13] L. Kaelbling, M. Littman, and A. Moore. Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, 4:237-285, 1996.
[14] H. Kautz and B. Selman. Pushing the envelope: planning, propositional logic, and stochastic search. In 13th National Conference on Artificial Intelligence, 1995.
[15] K. Kersting and L. De Raedt. Logical Markov decision programs and the convergence of logical TD(A). In 14th International Conference on Inductive Logic Programming, 2004.
[16] K. Kersting, M. van Otterlo, and L. De Raedt. Bellman goes relational. In IJCL, 2004.
[17] N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. Artificial Intelligence, 76(1-2):239-286, 1995.
[18] V. Lifschitz. Answer set planning. In ICLP, 1999.
[19] F. Lin and Y. Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. Artificial Intelligence, 157(1-2):115-137, 2004.
[20] M. Littman, A. Cassandra, and L. Kaelbling. Learning policies for partially observable environments: scaling up. In 12th IJCL, 1995.
[21] S. Majercik and M. Littman. MAXPLAN: A new approach to probabilistic planning. In 4th International Conference on Artificial Intelligence Planning, pages 86–93, 1998.
[22] S. Majercik and M. Littman. Contingent planning under uncertainty via stochastic satisfiability. Artificial Intelligence, 147(1-2):119–162, 2003.
[23] M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Complexity of finite-horizon Markov decision process problems. Journal of the ACM, 2000.
[24] I. Niemela and P. Simons. Efficient implementation of the well-founded and stable model semantics. In Joint International Conference and Symposium on Logic Programming, 289-303, 1996.
[25] E. Saad. Incomplete knowledge in hybrid probabilistic logic programs. In 10th European Conference on Logics in Artificial Intelligence, 2006.
[26] E. Saad. Towards the computation of the stable probabilistic model semantics. In 29th Annual German Conference on Artificial Intelligence, June 2006.
[27] E. Saad. A logical approach to qualitative and quantitative reasoning. In 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, 2007.
[28] E. Saad. Probabilistic planning in hybrid probabilistic logic programs. In 1st International Conference on Scalable Uncertainty Management, 2007.
[29] E. Saad. On the relationship between hybrid probabilistic logic programs and stochastic satisfiability. In 10th International Symposium on Artificial Intelligence and Mathematics, 2008.
[30] E. Saad. A logical framework to reinforcement learning using hybrid probabilistic logic programs. In Proceedings of the 2nd International Conference on Scalable Uncertainty Management (SUM’08), 2008.
[31] E. Saad. Probabilistic planning with imperfect sensing actions using hybrid probabilistic logic programs. In SUM, 2009.
[32] E. Saad. Probabilistic reasoning by SAT solvers. In Tenth ECSQARU, 2009.
[33] E. Saad and E. Pontelli. Towards a more practical hybrid probabilistic logic programming framework. In Practical Aspects of Declarative Languages, 2005.
[34] E. Saad and E. Pontelli. A new approach to hybrid probabilistic logic
IX. APPENDIX: Example

Example 2: The np-program encoding of the tiger domain presented in Example 1 is given by $\Pi = (R, \tau)$, where $\tau$ is arbitrary and $R$ consists of the following np-rules, in addition to the np-rules (7), (8), (9), (10), (25), (26), (27), (28):

\[
\begin{align*}
\text{action}(openL_i) & \leftarrow \text{action}(openR_i) \leftarrow \text{action}(listen_j) \leftarrow \\
\text{fluent}(tl) & \leftarrow \text{fluent}(htl) \leftarrow \\
\text{holds}(tl,0) & \leftarrow \text{not holds}(\neg tl,0) \\
\text{holds}(\neg tl,0) & \leftarrow \text{not holds}(tl,0) \\
\text{holds}(tl,0) & \leftarrow \text{holds}(htl,0) \\
\text{holds}(\neg tl,0) & \leftarrow \text{holds}(\neg htl,0)
\end{align*}
\]

The set of possible initial states are encoded by the p-rules:

\[
\begin{align*}
\text{state}(0) & : 0.5 \leftarrow \text{holds}(tl,0), \text{holds}(htl,0) \\
\text{state}(0) & : 0.5 \leftarrow \text{not holds}(tl,0), \text{not holds}(htl,0)
\end{align*}
\]

The executability conditions of actions are encoded by the following p-rules:

\[
\begin{align*}
\text{exec}(openL_i) & \leftarrow \text{exec}(openR_i) \leftarrow \text{exec}(listen_j) \leftarrow \\
\text{holds}(tl,T + 1) & \leftarrow \text{exec}(openL_1,T), \text{exec}(openL_1,T), \\
\text{exec}(openL_2,T), \text{exec}(openL_2,T), \\
\text{exec}(htl,T) \leftarrow \text{exec}(htl,T)
\end{align*}
\]

The probability distribution resulting from executing the \textit{listen} action is given by

\[
\begin{align*}
\text{state}(T + 1) : 0.85 \times V & \leftarrow \text{exec}(listen_1,T), \text{exec}(listen_1,T), \\
\text{exec}(listen_2,T), \text{exec}(listen_2,T), \\
\text{exec}(listen_3,T), \text{exec}(listen_3,T), \\
\text{exec}(listen_4,T), \text{exec}(listen_4,T)
\end{align*}
\]

The rewards received from executing the actions are encoded by

\[
\begin{align*}
\text{reward}(-100,T + 1) & \leftarrow \text{exec}(openL_1), \text{exec}(openL_2), \\
\text{exec}(openR_1), \text{exec}(openR_2), \\
\text{exec}(openR_1), \text{exec}(openR_2)
\end{align*}
\]

The value function is encoded in SRS by the np-rules:

\[
\begin{align*}
\text{value}(V + \lambda T \ast U \ast -100,T + 1) & \leftarrow \text{value}(V,T), \\
\text{factor}(\lambda), \text{state}(T + 1) : U, \text{reward}(-100,T + 1)
\end{align*}
\]

The effects of the \textit{listen} action are encoded by the p-rules:

\[
\begin{align*}
\text{observed}(htl,T) & \leftarrow \text{exec}(listen_1,T), \text{exec}(listen_1,T), \\
\text{exec}(listen_2,T), \text{exec}(listen_2,T), \\
\text{exec}(listen_3,T), \text{exec}(listen_3,T), \\
\text{exec}(listen_4,T), \text{exec}(listen_4,T)
\end{align*}
\]

The effects of the \textit{listen} action are encoded by the p-rules:

\[
\begin{align*}
\text{observed}(htl,T) & \leftarrow \text{exec}(listen_1,T), \text{exec}(listen_1,T), \\
\text{exec}(listen_2,T), \text{exec}(listen_2,T), \\
\text{exec}(listen_3,T), \text{exec}(listen_3,T), \\
\text{exec}(listen_4,T), \text{exec}(listen_4,T)
\end{align*}
\]

The value function is encoded in SRS by the np-rules:

\[
\begin{align*}
\text{value}(V + \lambda T \ast U \ast -100,T + 1) & \leftarrow \text{value}(V,T), \\
\text{factor}(\lambda), \text{state}(T + 1) : U, \text{reward}(-100,T + 1)
\end{align*}
\]

The effects of the \textit{listen} action are encoded by the p-rules:

\[
\begin{align*}
\text{observed}(htl,T) & \leftarrow \text{exec}(listen_1,T), \text{exec}(listen_1,T), \\
\text{exec}(listen_2,T), \text{exec}(listen_2,T), \\
\text{exec}(listen_3,T), \text{exec}(listen_3,T), \\
\text{exec}(listen_4,T), \text{exec}(listen_4,T)
\end{align*}
\]

The value function is encoded in SRS by the np-rules:

\[
\begin{align*}
\text{value}(V + \lambda T \ast U \ast -100,T + 1) & \leftarrow \text{value}(V,T), \\
\text{factor}(\lambda), \text{state}(T + 1) : U, \text{reward}(-100,T + 1)
\end{align*}
\]
\[ \text{value}(V + \lambda^T * U * 10, T + 1) \leftarrow \text{value}(V, T), \]
\[ \text{factor}(\lambda), \text{state}(T + 1) : U, \text{reward}(10, T + 1), \]
\[ \text{occ}(\text{openR}_2, T), \text{exec}(\text{openR}_2, T), \]
\[ \text{holds}(tl, T), \text{holds}(tl, T + 1) \]
\[ \text{value}(V + \lambda^T * U * -1, T + 1) \leftarrow \text{value}(V, T), \]
\[ \text{factor}(\lambda), \text{state}(T + 1) : U, \text{reward}(-1, T + 1), \]
\[ \text{occ}(\text{listen}_1, T), \text{exec}(\text{listen}_1, T), \]
\[ \text{observed}(\text{htl}, T), \text{holds}(tl, T + 1) \]
\[ \text{value}(V + \lambda^T * U * -1, T + 1) \leftarrow \text{value}(V, T), \]
\[ \text{factor}(\lambda), \text{state}(T + 1) : U, \text{reward}(-1, T + 1), \]
\[ \text{occ}(\text{listen}_2, T), \text{exec}(\text{listen}_2, T), \]
\[ \text{observed}(\text{htl}, T), \text{holds}(tl, T + 1) \]
\[ \text{value}(V + \lambda^T * U * -1, T + 1) \leftarrow \text{value}(V, T), \]
\[ \text{factor}(\lambda), \text{state}(T + 1) : U, \text{reward}(-1, T + 1), \]
\[ \text{occ}(\text{listen}_3, T), \text{exec}(\text{listen}_3, T), \]
\[ \text{observed}(\neg \text{htl}, T), \text{holds}(tl, T + 1) \]
\[ \text{value}(V + \lambda^T * U * -1, T + 1) \leftarrow \text{value}(V, T), \]
\[ \text{factor}(\lambda), \text{state}(T + 1) : U, \text{reward}(-1, T + 1), \]
\[ \text{occ}(\text{listen}_4, T), \text{exec}(\text{listen}_4, T), \]
\[ \text{observed}(\neg \text{htl}, T), \text{holds}(tl, T + 1) \]