# Combining Logic and Markov Decision Processes Using Hybrid ASP

**Alex Brik, Jeffrey Remmel**
Department of Mathematics, UC San Diego, USA

## Abstract

This paper shows how the extension of ASP called Hybrid ASP introduced by the authors in [4] can be used to combine logical reasoning and Markov Decision Processes (MDPs) with transition densities finitely supported in infinite domains. The paper shows how a computer language H-ASP# based on Hybrid ASP can be used to create efficient, robust representations of dynamic domains and to compute optimal finite horizon policies for the agents acting in such domains. The complexity of computing optimal policies is EXP-complete in the length of the input program. The paper extends [5] where the authors discuss using Hybrid ASP to combine logical reasoning and MDPs with transition densities in finite domains.

## 1 Introduction

Hybrid Answer Set Programming (Hybrid ASP or H-ASP) introduced by the authors in [4], is an extension of Answer Set Programming (ASP) that allows users to combine ASP type rules and numerical algorithms. In [5] the authors have shown that H-ASP can be used to create efficient, robust representations of dynamic domains with transition densities over finite sets and to compute optimal finite horizon policies for the agents acting in the domains. This paper extends [5] by demonstrating that H-ASP can be used to create representations of dynamic domains with transition densities which are finitely supported over infinite sets. The paper discusses a computer language H-ASP# introduced in [5], [3] based on H-ASP, which is used to obtain computational solutions.

The process for computing an optimal finite horizon policy from H-ASP# program is as follows.

**1.** Use H-ASP# program to generate a flat representation of the underlying MDP. This is done by computing the successor states and state transitions of the initial state. The step is then repeated for all the newly computed successor states to compute the set of states removed from the initial state by two steps. The step is repeated again until no more successor states can be computed.

**2.** Use the Dynamic Programming algorithm on the flat representation of MDP to compute an optimal finite horizon policy.

H-ASP is a general formalism for combining ASP type rules and numerical algorithms. H-ASP is applicable to a wide range of problems. Thus, the particular problem of computing optimal finite horizon policies considered in this paper is just one example of possible applications of H-ASP. In H-ASP rules act as input-output devices for the algorithms. This feature of H-ASP allows it to be effectively used for reasoning about domains with transition densities which are finitely supported over infinite sets.

Markov Decision Processes (MDPs), first described in [2] are widely used to model decision-making problems. At a specific time, a decision maker observes the state of the system and decides which action to perform. Upon performing an action, the decision maker incurs a reward that depends on the state of the system and the action chosen. The system then nondeterministically moves to a new state with a known probability, at which time a decision maker again observes the state of the system and decides which action to perform. The goal is to determine a policy that maximizes a cumulative expected reward. In the discrete finite horizon case, which is the case that we will be considering, the problem is usually solved numerically using the Dynamic Programming algorithm (DP algorithm for short). Thus one could encode a probabilistic model of the domain from the description, and then run the DP algorithm with the probabilistic model encoded as an input to generate a finite horizon optimal

policy. We will call this approach the ad hoc approach.

It is then reasonable to ask whether solving the problem using H-ASP provides an advantage over the ad hoc approach? We will argue in this paper that solving the problem using H-ASP provides a number of advantages over the ad hoc approach. One advantage is that a H-ASP formulation of the problem allows the user to easily modify the underlying search space by imposing logical constraints on the system in much the same way that an ASP programmer can impose constraints on the intended set of stable models of the program.

There are various approaches for combining ASP and probability that are described in the literature. In [10], Saad shows that normal hybrid probabilistic logic programs with answer set semantics introduced in [11] can be used to describe possible finite trajectories of MDPs and to compute finite horizon optimal policies. We will discuss this and other related work in the conclusion. Generally, the approaches for combining ASP and probability do not handle transition densities finitely supported over infinite sets. The main advantage of H-ASP over these methods is in the ability of H-ASP to handle transition densities finitely supported over infinite sets.

First, we describe a dynamic domain that will be used as the main example in this paper. The dynamic domain and the associated problem will be called the Secret Agent Problem. It is a typical example of a problem for the use of DP algorithm.

<u>The Secret Agent Problem</u>.

*Secret agent 00111 needs to cross a rectangular lake by a boat (see figure 1). The opposite shore of the lake has only one location where the secret agent can safely exit. At all the other locations on the opposite shore the evil agents are waiting in an ambush. The lake is divided into 4 segments parallel to the shores. Each segment has a water current with water flowing parallel to the shores. The speed of the water can change suddenly in one of two predetermined ways with equal probability (the speed can take both positive and negative values). To simplify the problem we will assume that the water speed changes in 1 second intervals. Every 1 second the agent can choose to steer the boat at the same angle as before, or choose to increase the steering angle by $\Delta_{angle}$ or choose to decrease the steering angle by $\Delta_{angle}$ if doing so does not steer the boat back towards the starting shore. The boat is assumed to travel at a constant velocity. What is an optimal choice of steering angles allowing the agent to arrive at the opposite shore as fast as possible while avoiding an ambush?*

A few comments about the problem description should be made. First, the problem description specifies the

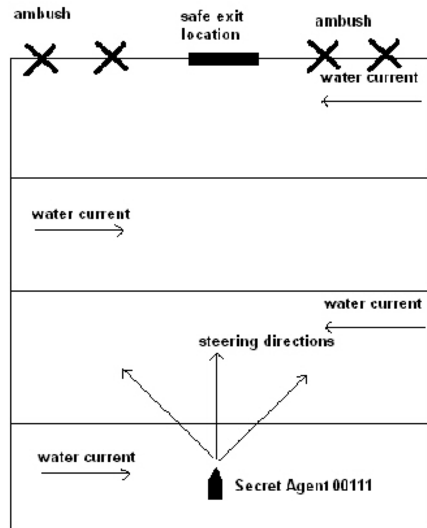


Figure 1: Secret Agent domain

laws of the domain. Then the probabilistic model, i.e. the probabilities of moving from one state to another under a particular action, needs to be computed using the laws specified in the description. While the laws in the Secret Agent Problem are simple and easily yield the probabilistic model, one can see that the laws could be more complicated and that creating a probabilistic model from the description could be a challenging problem in itself. Second, the number of states and state transitions will be on the order of hundreds of thousands. Thus to be practical, the software that generates and analyzes the probabilistic model needs to be efficient. Third, even small changes to the problem description can significantly change the probabilistic model. This is because adding, changing or removing a law can affect all the states. For instance suppose that the water speed in each segment can take 3 different values with unequal probability. This will significantly change the probabilistic model and the state space.

Our first comment justifies the idea that it is desirable to produce a probabilistic model of a domain by encoding a domain description in a language that has syntactic constructs for specifying laws of the domain. H-ASP rules are syntactic constructs that allow specifying laws of domains. Our third comment justifies the need for a general approach. In this paper we will describe a general approach based on H-ASP. The H-ASP formulation of the problem allows the user to change the laws of the underlying system with little restructuring of the rest of the H-ASP program.

The paper discusses one way of solving the Secret Agent Problem using H-ASP. However, the techniques

we use are general and can be applied to many other problems. For the implementation purposes we will discuss a language closely related to H-ASP called H-ASP#. We have implemented a prototype H-ASP# solver, created a H-ASP# program for the Secret Agent Problem, and have used the solver with the program to solve the problem.

In this paper due to space constraint we have to omit many technical details. For the interested reader, the details can be found in [3].

Solving the Secret Agent Problem is accomplished by modeling the Secret Agent Domain as a H-ASP# program $P\#$ and then using a H-ASP# solver software to find an optimal policy. The semantics of H-ASP# program $P\#$ is defined using stable model semantics of certain H-ASP program derived from $P\#$.

The rest of the paper is structured as follows. In section 2, we will review the MDPs and DP algorithm. In section 3, we will review ASP and H-ASP. In section 4, we will discuss H-ASP#. In section 5 we will discuss H-ASP# program modeling the Secret Agent Problem. We end with a conclusion and a discussion of the related work.

## 2 Markov Decision Processes

A MDP $M$ is a tuple $M = \langle S, S_0, A, T, r \rangle$ where $S$ is a set of states, $S_0 \subseteq S$ is the set of initial states, $A$ is the set of actions, $T : S \times A \times S \rightarrow [0,1]$ is the stationary transition function such that for all $s \in S$ and $a \in A$, $T(s,a,\cdot)$ is the probability distribution over $S$, i.e. $T(s,a,s')$ is the probability of moving to a state $s'$ when action $a$ is executed in state $s$. $r : S \times A \rightarrow \mathbb{R}$ is the reward function, i.e. $r(s,a)$ is the reward gained by taking action $a$ in state $s$.

A trajectory $\theta$ is a sequence of states $\theta_1, \theta_2, \ldots, \theta_m$ for some $m \geq 0$, where $\theta_1, \theta_2, \ldots, \theta_m \in S$. A policy $\pi$ is a map $S \times \mathbb{Z}^+ \rightarrow A$ where $\mathbb{Z}^+$ is the set of all the positive integer numbers, that maps each state $s \in S$ and time $i \in \mathbb{Z}^+$ to an action $\pi(s,i)$. Let an MDP $M$, and a finite horizon $\kappa$ be given. $\kappa$ is the maximum length of the trajectories that we will be considering. The probability $\text{Prob}_\pi(\theta)$ of a trajectory $\theta$ under a policy $\pi$ is $\text{Prob}_\pi(\theta) = \prod_{i=1}^{|\theta|-1} T(\theta_i, \pi(\theta_i, i), \theta_{i+1})$ where $|\theta|$ is the length of the trajectory. The reward $R_\pi^n(\theta)$ of a trajectory $\theta$ at time $n$ under a policy $\pi$ is $R_\pi^n(\theta) = \sum_{i=1}^{|\theta|} r(\theta_i, \pi(\theta_i, n+i-1))$.

Let $\Theta(s,i)$ be the set of all the trajectories of length $i$ starting at state $s \in S$. Note that $\Theta(s,1) = (s)$. The performance of a policy $\pi$ at time $i$ with initial state $s$ is the expected sum of rewards received on the next $\kappa + 1 - i$ steps by following the policy $\pi$. That is $R_\pi(s,i) = \sum_{\theta \in \Theta(s,\kappa+1-i)} \text{Prob}_\pi(\theta) \cdot R_\pi^i(\theta)$.

The finite horizon optimal policy problem is to find a policy $\pi^*$ that would maximize the $\kappa$ step performance, i.e. find $\pi^*$ such that for all the policies $\pi$ for all $s_0 \in S_0$

$$R_{\pi^*}(s_0, 1) \geq R_\pi(s_0, 1)$$

The performance for a policy $\pi$ at time $i$ satisfies the following recursive formula

$$R_\pi(s,i) = r(s, \pi(s,i)) + \sum_{s' \in S} T(s, \pi(s,i), s') \cdot R_\pi(s', i+1) \quad (1)$$

where $R_\pi(s,\kappa) = r(s, \pi(s,\kappa))$. It is the case that there exists $\pi^*$ such that for all $s \in S$ and for all the policies $\pi$ $R_{\pi^*}(s,1) \geq R_\pi(s,1)$. $\pi^*$ can be constructed as follows. First define $\pi^*(s,\kappa) = \text{argmax}_{a \in A} r(s,a)$. Then for $1 \leq i < \kappa$, let

$$\pi^*(s,i) = \text{argmax}_{a \in A} \sum_{s' \in S} T(s,a,s') \cdot R_{\pi^*}(s', i+1) . \quad (2)$$

The above recursive equations defines the DP algorithm. The algorithm proceeds by first computing $\pi^*(s,\kappa)$ and $R_{\pi^*}(s,\kappa)$ for every $s \in S$. Now, assuming that for every $s \in S$ $\pi^*(s,i+1)$ and $R_{\pi^*}(s,i+1)$ are computed, compute $\pi^*(s,i)$ using formula 2 and then $R_{\pi^*}(s,i)$ using formula 1 for all $s \in S$, except for $i = 1$ where $\pi^*(s,1)$ needs to be computed only for the initial states $S_0$ and $R_{\pi^*}(s,1)$ need not be computed. It is easy to see that the DP algorithm is a polynomial time algorithm in $|S|$ and $|A|$.

A *flat representation* of a MDP $\langle S, S_0, A, T, r \rangle$ is a set of $|S| \times |S|$ tables for the transition function - one table for each action and a table for a reward function [7].

## 3 Hybrid ASP

We will now give a brief overview of ASP and then a brief overview of H-ASP.

A *normal propositional logic programming rule* (normal propositional logic programming rule) is an expression of the form

$$C = p \leftarrow q_1, \ldots, q_m, \text{ not } r_1, \ldots, \text{not } r_n \quad (3)$$

where $p, q_1, \ldots, q_m, r_1, \ldots, r_n$ are atoms from a fixed set of atoms $At$. The atom $p$ in the rule above is called the *head* of $C$ ($head(C)$), and the expression

$q_1, \ldots, q_m, \text{not } r_1, \ldots, \text{not } r_n$, with ',' interpreted as the conjunction, is called the *body* of $C$ ($body(C)$). The set $\{q_1, \ldots, q_n\}$ is called the *positive part of the body* of $C$ (posBody($C$)) (or *premises* of $C$) and the set $\{r_1, \ldots, r_m\}$ is called the *negative part of the body* of $C$ (negBody($C$)) (or *constraints* of $C$). Given any set $M \subseteq At$ and atom $a$, we say that $M$ satisfies $a$ (not $a$), written $M \models a$ ($M \models \text{not } a$), if $a \in M$ ($a \notin M$). For a rule $C$ of the form 3 we say that $M$ satisfies the body of $C$ if $M$ satisfies $q_i$ for $i = 1, ..., m$ and $M$ satisfies *not* $r_j$ for $j = 1, ..., n$. We say that $M$ satisfies $C$, written $M \models C$, if whenever $M$ satisfies the body of $C$, then $M$ satisfies the head of $C$. A normal logic program $P$ is set of rules of the form of (3). We say that $M \subseteq At$ is a model of $P$, written $M \models P$, if $M$ satisfies every rule of $P$.

A propositional Horn rule is a logic programming rule of the form $H = p \leftarrow q_1, \ldots, q_m$, where $p, q_1, \ldots, q_m \in At$. Thus in a Horn rule, the negative part of its body is empty. A Horn program $P$ is a set of Horn rules. Each Horn program $P$ has a least model under inclusion relation, $LM_P$, which can be defined using the one-step provability operator $T_P$. For any set $A$, let $\mathcal{P}(A)$ denote the set of all subsets of $A$. The one-step provability operator $T_P : \mathcal{P}(A) \to \mathcal{P}(A)$ associated with the Horn program $P$ [12] is defined by setting:

$$T_P(M) = \{p : \exists C \in P(p = head(C) \wedge M \models body(C))\}$$

for any $M \in \mathcal{P}(A)$. We define $T_P^n(M)$ by induction by setting $T_P^1(M) = T_P(M)$ and $T_P^{n+1}(M) = T_P(T_P^n(M))$. Then the least model $LM_P$ can be computed as $LM_P = T_P(\emptyset) \uparrow \omega = \bigcup_{n \geq 1} T_P^n(\emptyset)$.

If $P$ is a normal logic program and $M \subseteq At$, then the Gelfond-Lifschitz reduct of $P$ with respect to $M$ [6] is the Horn program $P^M$ which results by eliminating those rules $C$ of the form (3) such that $r_i \in M$ for some $i$ and replacing $C$ by $p \leftarrow q_1, \ldots, q_n$ otherwise. We then say that $M$ is a *stable model* for $P$ if $M$ equals the least model of $P^M$.

An *answer set programming rule* is an expression of the form 3 where $p, q_1, \ldots, q_m, r_1, \ldots, r_n$ are classical literals, i.e., either positive atoms or atoms preceded by the classical negation sign $\neg$. Answer sets are defined in analogy to stable models, but taking into account that atoms may be preceded by classical negation.

A H-ASP program $P$ has an underlying parameter space $S$. Elements of $S$ are of the form $\mathbf{p} = (t, x_1, \ldots, x_m)$ where $t$ is time and $x_i$ are parameter values. We shall let $t(\mathbf{p})$ denote $t$ and $x_i(\mathbf{p})$ denote $x_i$ for $i = 1, \ldots, m$. We refer to the elements of $S$ as *generalized positions*.

Let $At$ be a set of atoms of $P$. Then the universe of $P$ is $At \times S$. For ease of notation, we will often identify an atom and the string representing atom.

If $M \subseteq At \times S$, we let $\widehat{M} = \{\mathbf{p} \in S : (\exists a \in At)((a, \mathbf{p}) \in M)\}$. A block $B$ is an object of the form $B = a_1, \ldots, a_n, \textit{not } b_1, \ldots, \textit{not } b_m$ where $a_1, \ldots, a_n, b_1, \ldots, b_m \in At$. Given $M \subseteq At \times S$, $B = a_1, \ldots, a_n, \textit{not } b_1, \ldots, \textit{not } b_m$, and $\mathbf{p} \in S$, we say that $M$ satisfies $B$ at the generalized position $\mathbf{p}$, written $M \models (B, \mathbf{p})$, if $(a_i, \mathbf{p}) \in M$ for $i = 1, \ldots, n$ and $(b_j, \mathbf{p}) \notin M$ for $j = 1, \ldots, m$. If $B$ is empty, then $M \models (B, \mathbf{p})$ automatically holds. We define $B^- = \textit{not } b_1, \ldots, \textit{not } b_m$.

There are two types of rules in H-ASP.

**Advancing rules** are of the form

$$\frac{B_1; B_2; \ldots; B_r : A, O}{a}$$

where $A$ is an algorithm, each $B_i$ is of the form $a_1, \ldots, a_n, \textit{not } b_1, \ldots, \textit{not } b_m$ where $a_1, \ldots, a_n$, $b_1, \ldots, b_m$, and $a$ are atoms, and $O \subseteq S^r$ is such that if $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O$, then $t(\mathbf{p}_1) < \ldots < t(\mathbf{p}_r)$, $A(\mathbf{p}_1, \ldots, \mathbf{p}_r) \subseteq S$, and for all $\mathbf{q} \in A(\mathbf{p}_1, \ldots, \mathbf{p}_r)$, $t(\mathbf{q}) > t(\mathbf{p}_r)$. Here and in the next rule, we allow $n$ or $m$ to be equal to 0 for any given $i$. Moreover, if $n = m = 0$, then $B_i$ is empty and we automatically assume that $B_i$ is satisfied by any $M \subseteq At \times S$. We shall refer to $O$ as the *constraint set* of the rule and the algorithm $A$ as the *advancing algorithm* of the rule. The idea is that if $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O$ and for each $i$, $B_i$ is satisfied at the generalized position $\mathbf{p}_i$, then the algorithm $A$ can be applied to $(\mathbf{p}_1, \ldots, \mathbf{p}_r)$ to produce a set of generalized positions $O'$ such that if $\mathbf{q} \in O'$, then $t(\mathbf{q}) > t(\mathbf{p}_r)$ and $(a, \mathbf{q})$ holds.

**Stationary rules** are of the form

$$\frac{B_1; B_2; \ldots; B_r : H, O}{a}$$

where each $B_i$ is of the form $a_1, \ldots, a_n, \textit{not } b_1, \ldots, \textit{not } b_m$ where $a_1, \ldots, a_n, b_1, \ldots, b_m$ and $a$ are atoms, $O \subseteq S^r$ is such that if $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O$, then $t(\mathbf{p}_1) < \cdots < t(\mathbf{p}_r)$, and $H$ is a Boolean algorithm defined on $O$. We shall refer to $O$ as the *constraint set* of the rule and the algorithm $H$ as the *Boolean algorithm* of the rule. The idea is that if $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O$ and for each $i$, $B_i$ is satisfied at the generalized position $\mathbf{p}_i$, and $H((\mathbf{p}_1, \ldots, \mathbf{p}_r))$ is true, then $(a, \mathbf{p}_r)$ holds.

In an implemented system, the algorithms in H-ASP rules are allowed to be any sort of algorithms, for instance algorithms for solving differential or integral equations, solving a set of linear equations or linear programming equations, etc.

A *H-ASP Horn program* is a H-ASP program which does not contain any negated atoms in $At$.

Let $P$ be a Horn H-ASP program, let $I \in S$ be an initial condition. Then the one-step provability operator $T_{P,I}$ is defined so that given $M \subseteq At \times S$, $T_{P,I}(M)$ consists of $M$ together with the set of all $(a, J) \in At \times S$ such that

**(1)** there exists a stationary rule $C = \frac{B_1;B_2;\ldots;B_r:H,O}{a}$ and $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O \cap \left(\widehat{M} \cup \{I\}\right)^r$ such that $(a, J) = (a, \mathbf{p}_r)$, $M \models (B_i, \mathbf{p}_i)$ for $i = 1, \ldots, r$, and $H(\mathbf{p}_1, \ldots, \mathbf{p}_r) = 1$ or

**(2)** there exists an advancing rule $C = \frac{B_1;B_2;\ldots;B_r:A,O}{a}$ and $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O \cap \left(\widehat{M} \cup \{I\}\right)^r$ such that $J \in A(\mathbf{p}_1, \ldots, \mathbf{p}_r)$ and $M \models (B_i, \mathbf{p}_i)$ for $i = 1, \ldots, r$.

The stable model semantics for H-ASP programs will now be defined. Let $M \subseteq At \times S$, let $I \in S$. An H-ASP rule $C = \frac{B_1;\ldots,B_r:A,O}{a}$ is *inconsistent* with $(M, I)$ if for all $(\mathbf{p}_1, \ldots, \mathbf{p}_r) \in O \cap \left(\widehat{M} \cup \{I\}\right)^r$, either (i) there is an $i$ such that $M \not\models (B_i^-, \mathbf{p}_i)$ (ii) $A(\mathbf{p}_1, \ldots, \mathbf{p}_r) \cap \widehat{M} = \emptyset$ if $A$ is an advancing algorithm, or (iii) $A(\mathbf{p}_1, \ldots, \mathbf{p}_r) = 0$ if $A$ is a Boolean algorithm. Then we form the Gelfond-Lifschitz reduct of $P$ over $M$ and $I$, $P^{M,I}$ as follows.

**(1)** Eliminate all rules that are inconsistent with $(M, I)$.

**(2)** If the advancing rule $C = \frac{B_1;\ldots,B_r:A,O}{a}$ is not eliminated by (1), then replace it by $\frac{B_1^+;\ldots,B_r^+:A^+,O^+}{a}$ where for each $i$, $B_i^+$ is the result of removing all the negated atoms from $B_i$, $O^+$ is equal to the set of all $(\mathbf{p}_1, \ldots, \mathbf{p}_r)$ in $O \cap \left(\widehat{M} \cup \{I\}\right)^r$ such that $M \models (B_i^-, \mathbf{p}_i)$ for $i = 1, \ldots, r$ and $A(\mathbf{p}_1, \ldots, \mathbf{p}_r) \cap \widehat{M} \neq \emptyset$, and $A^+(\mathbf{p}_1, \ldots, \mathbf{p}_r)$ is defined to be $A(\mathbf{p}_1, \ldots, \mathbf{p}_r) \cap \widehat{M}$.

**(3)** If the stationary rule $C = \frac{B_1;\ldots,B_r:H,O}{a}$ is not eliminated by (1), then replace it by $\frac{B_1^+;\ldots,B_r^+:H|_{O^+},O^+}{a}$ where for each $i$, $B_i^+$ is the result of removing all the negated atoms from $B_i$, $O^+$ is equal to the set of all $(\mathbf{p}_1, \ldots, \mathbf{p}_r)$ in $O \cap \left(\widehat{M} \cup \{I\}\right)^r$ such that $M \models (B_i^-, \mathbf{p}_i)$ for $i = 1, \ldots, r$ and $H(\mathbf{p}_1, \ldots, \mathbf{p}_r) = 1$.

We then say that $M$ is a *stable model of $P$ with initial condition $I$* if $\bigcup_{k=0}^{\infty} T^k_{P^{M,I},I}(\emptyset) = M$.

We say that $M$ is a *single trajectory stable model of $P$ with initial condition $I$* if $M$ is a stable model of $P$ with the initial condition $I$ and for each $t \in \{t(\mathbf{p}) \mid \mathbf{p} \in S\}$ there exists at most one $\mathbf{p} \in \widehat{M} \cup \{I\}$ such that $t(\mathbf{p}) = t$.

We say that an advancing algorithm $A$ lets a parameter $y$ be *free* if the domain of $y$ is $Y$ and for all generalized positions $\mathbf{p}$ and $\mathbf{q}$ and all $y' \in Y$, whenever $\mathbf{q} \in A(\mathbf{p})$, then there exist $\mathbf{q}' \in A(\mathbf{p})$ such that $y(\mathbf{q}') = y'$ and $\mathbf{q}$ and $\mathbf{q}'$ are identical in all the parameter values except possibly $y$.

# 4 H-ASP#

The main reason for introducing H-ASP# is the following. Because advancing algorithms can let parameters be free, it is the case that even for problems of modest size, the number of generalized positions that advancing algorithms produce can be enormous. While most of these generalized positions will not be a part of any stable model, producing them makes implementations impossible. Thus we need to have a mechanism where the values of the parameters which are free are not produced.

In [4], the authors have suggested an indirect approach by which the advancing algorithms can specify values for only some of the parameters. The approach requires extending the Herbrand base of $P$ by new atoms $S_1$, $S_2$, ..., $S_n$ one for each parameter. Suppose that there is an advancing algorithm $A$ in a rule $\frac{B:A,O}{a}$ that specifies parameters with indexes $i_1$, $i_2$, ..., $i_k$ and lets other parameters be free. Then we add to $P$ rules of the form $\frac{B:A,O}{S_{i_j}}$ for each $j$ from 1 to $k$. This is repeated for every advancing rule of $P$. Then if $M$ is a stable model of $P$ and $\mathbf{p} \in \widehat{M}$, we will require that $\{S_1, \ldots, S_n\} \subseteq W_M(\mathbf{p})$. That is, we will require that every parameter is set at $\mathbf{p}$ by some advancing algorithm. To accomplish this, we add to $P$ the following stationary rules for $i = 1, \ldots, n$ $\frac{not\ S_i,\ not\ \mathrm{FAIL}}{\mathrm{FAIL}}$. We will refer to the mechanism as the Parameter Restriction Mechanism.

A H-ASP# program consists of parameter declarations, algorithm declarations, commands, and rules. Parameter declarations have the form param$(p)$ where $p$ is a parameter to be used by the H-ASP# program. The parameters Act, Prob and Ret describing an action choice, an unnormalized probability and cumulative expected reward respectively are assumed to be used in every H-ASP# program and do not need to be declared. An algorithm declaration has the form program $p(a_1,\ldots,a_k)$ $\{c_1;\ldots;c_m\}$ where $p$ is the name of the algorithm, $a_1,\ldots,a_k$ are the names of the input arguments, $c_1,\ldots,c_m$ are commands. A signature declaration has the form sig$(p, (s_1, \ldots, s_k))$ where $p$ is the algorithm name, $s_1, \ldots, s_k$ are parameter names. A signature states that the algorithm's output is a set of tuples of the form $(y_1, \ldots, y_k)$ where $y_i$ is the value for the parameter $s_i$. The signature declarations are required for the advancing algorithms.

H-ASP# rules have the form $a$:-$B_1$ : $A, O$ or $a$:-$B_1; B_2$ : $A, O$ where $a$ is a string repre-

sentation of an atom, $B_1$, $B_2$ are of the form $c_1, \ldots, c_k, not \quad d_1, \ldots, not \quad d_m$ where $c_1, \ldots, c_k$, $d_1, \ldots, d_m$ are string representations of atoms. $A$ and $O$ are the algorithm names as declared in the algorithm declarations. $O$ is a name of a Boolean algorithm. The algorithms used in rules can be those declared in H-ASP# program or those provided by the solver. Since there are no significant restrictions regarding which algorithms a solver can provide, hypothetically H-ASP# rules can use arbitrary algorithms.

H-ASP# implements the Parameter Restriction Mechanism, with the parameter Prob excepted. That is, for H-ASP# advancing rule, $a : -B : A, O$, the algorithm $A$ is assumed to set the parameters that are specified in its signature declaration and let others be free. Since we require that for every generalized position, every parameter is set by some advancing algorithm, the free parameters are simply not generated by the advancing algorithm. The rules corresponding to $\frac{B:A,O}{S_{i_j}}$ and $\frac{not\ S_i,\ not\ \mathrm{FAIL}}{\mathrm{FAIL}}$ then are not included in H-ASP# programs. It will be an error if at a generalized position a parameter is set by more than one advancing algorithm.

There are two other assumptions used in H-ASP#.

**1.** We assume that if $\mathbf{p}$ is a generalized position and $A$ is an advancing algorithm in a H-ASP# program then for all $\mathbf{q} \in A(\mathbf{p})$ $t(\mathbf{q}) = t(\mathbf{p}) + 1/2$.

**2.** If $O$ is a constraint set algorithm occurring in a H-ASP# program and $O(\mathbf{p}, \mathbf{q}) = 1$ iff $t(\mathbf{p}) + 1/2 = t(\mathbf{q})$

If a signature of an advancing algorithm $A$ is ($Act$) then an advancing rule that uses $A$ is called an *action rule*. If a signature of an advancing algorithm $A$ is (Prob) then an advancing rule that uses $A$ is called a *probability rule*. If an advancing rule is neither an action rule nor a probability rule then it is referred to as simply an advancing rule.

A H-ASP# program has to specify how to derive action choice, how to derive consequences of performing actions and how to derive unnormalized probabilities. Deriving action choice is accomplished by the action rules and stationary rules. Deriving consequences of performing actions is accomplished by the advancing rules and stationary rules. Deriving unnormalized probabilities is accomplished by the probability rules. Deriving action choices and their consequences will occur in two stages. First an action will be chosen. Second, the action's consequences will be derived. To model the two stage process, we will use the parameter LEVEL that will take values 0 or 1. Choosing an action $a$ at a generalized position $\mathbf{p}$ with $\mathrm{LEVEL}(\mathbf{p}) = 0$ is modeled by an appropriate advancing algorithm producing a generalized position $\widehat{\mathbf{p}}$ which is identical to $\mathbf{p}$ in all the parameter values except $t(\widehat{\mathbf{p}}) = t(\mathbf{p}) + 1/2$, $\mathrm{Act}(\widehat{\mathbf{p}}) = a$ and $\mathrm{LEVEL}(\widehat{\mathbf{p}}) = 1$. We will assume that an action rule can only be used in a generalized position $\mathbf{p}$ where $\mathrm{LEVEL}(\mathbf{p}) = 0$ and an advancing rule and a probability rule can be used only when $\mathrm{LEVEL}(\mathbf{p}) = 1$. When generating $\widehat{\mathbf{p}}$ from $\mathbf{p}$ all the atoms derived at $\widehat{\mathbf{p}}$ are copied to $\mathbf{p}$. Thus the states at $\widehat{\mathbf{p}}$ and at $\mathbf{p}$ are nearly identical except for the values of time and Act parameters.

Informally, the stable model of the H-ASP# program $W\#$ is the unique maximal stable model of H-ASP program $W$ derived from $W\#$, with the initial condition $J$ derived from the initial condition specified in $W\#$. The definitions of the derivations are omitted due to the considerations of space. However an interested reader can find these in [3].

We prove the following computational result for H-ASP# programs.

**Theorem.** *Let $W\#$ be a H-ASP# program that generates a MDP, and let $I$ be an initial condition. Let the length of $W\#$ be the number of bits required to represent all the statements of $W\#$ plus the number of bits required to encode all the algorithms used in $W\#$ as Turing machines. Suppose that for every advancing algorithm $A$ in $W\#$ and for every input $\mathbf{p}$, the length of the output $|A(\mathbf{p})|$ of $A$ with the input $\mathbf{p}$ is $O(|W\#|^{m_1})$ for some $m_1 \geq 0$. Let the horizon be $\kappa$ which is $O(|W\#|^{m_2})$ for some $m_2 \geq 0$. Suppose that every algorithm used in $W\#$ is a polynomial time algorithm. Then the question of whether there exists a policy with a non-negative performance is EXP-complete in $|W\#|$.*

The proof of the result provides a reduction of the succinct circuit value problem to the problem of non-negative policy existence for an appropriate H-ASP# program and is based on the proof of Theorem 1 [8]. The proof can be found in [3].

# 5 Modeling the Secret Agent Problem using H-ASP#

We will need the following parameters: LOCATION, ANGLE. The values of LOCATION will be $1 \times 2$ vectors describing the locations of the agent. Note that LOCATION is an infinite parameter. The values of ANGLE will be a floating point numbers, with $\mathrm{ANGLE}(\mathbf{p})$ specifying the steering angle of the agent at a state with the generalized position $\mathbf{p}$. With these parameters we will describe the location of the agent and the steering angle. The set of atoms $At$ will contain atoms: DONE, $T$, FAIL. The atom DONE will be

used to signal that the agent has arrived at the opposite shore. $T$ is a placeholder atom. FAIL will be used for constraints as in $\frac{\textit{not}\ \text{FAIL}}{\text{FAIL}}$ and thus will never be part of a stable model.

At every time step the Secret Agent will choose an action of the type (move, $\alpha$) where $\alpha$ is one of the possible steering angles. Once the agent arrives at the opposite shore the only possible action is done which does not change any parameters.

Action rules are used to derive action choices. Below is an action rule deriving actions of the form (move, $\alpha$). The encoding of the algorithms is omitted for brevity.

```
T:- : act_move, isNotDone1

program isNotDone1 (p) {
#
# Check whether the agent has reached
# the opposite shore
}

program act_move (p) {
#
# Generate possible actions of the form
# ("move", a), where a is an angle.
}
sig (act_move, (Act))
```

Below is an advancing rule specifying the effect of actions on the parameter ANGLE.

```
T:- : adv_ANGLE, isActionNotDone

program isActionNotDone (p) {
   Act(p)!="done"
}

program adv_ANGLE (p) {
#
# return the set containing an angle
# specified in Act(p)
}
sig (adv_ANGLE, (ANGLE))
```

Below is a probability rule specifying the unnormalized transition probabilities. As can be seen from the rule all the transitions are assigned equal probability.

```
T:- ; : prob_default

program prob_default (p, q) { 1 }
sig (prob_default, (Prob))
```

The agent's reward at a state is determined as follows. If $\kappa$ is the horizon, then the reward for arriving at a safe exit is $\kappa$ minus the number of 1 second intervals required for the arrival. All other states get a reward of 0.

The full text of the program can be found in http://math.ucsd.edu/~abrik/Secret_Agent/

## 6 Conclusion

The main advantages of using H-ASP# programs over an ad hoc approach for finding an optimal policy is that such programs produce robust and compact representations of dynamic domains which can be modified easily. Implementing even simple changes to an ad hoc model of a dynamic domain may require creating a model from scratch, depending on how the model is constructed. However making changes in a H-ASP# program often requires only changing rules and algorithms that model the changed parts.

There is extensive literature on combining ASP and probability. However to our knowledge only the work of Saad [10] addresses the problem of computing optimal policies of MDPs. In [10] Saad introduces a Markov Action Language $\mathcal{A}_{MD}$ which allows to describe MDPs. There are three main differences between the present work and [10]. The statements in $\mathcal{A}_{MD}$ can specify exact probabilities, whereas in H-ASP# the probabilities as specified in the values of Prob parameter are unnormalized probabilities. The difference can be significant for modeling. In some cases only the ratio of probabilities is known. For instance we may know that among certain two outcomes the first is twice as likely as the second. This condition can be easily modeled using unnormalized probabilities by multiplying by 2 the unnormalized probability of the state corresponding to the first outcome, and not multiplying the unnormalized probability of the state corresponding to the second outcome. However, specifying the exact probabilities would require the information about all the successor states making probability assignment more difficult. The second difference is that H-ASP# allows the use of algorithms for modeling. For example, in H-ASP#, it is possible to realistically model dynamic domains with parameters over infinite sets where physical processes have to be modeled by the numerical methods. This cannot be achieved in $\mathcal{A}_{MD}$. The third difference has to do with computing an optimal policy. [10] shows how an $\mathcal{A}_{MD}$ program $B$ can be translated into a normal hybrid probabilistic logic program $B_P$. The probabilistic answer sets of $B_P$ correspond to the valid trajectories of the underlying MDP. Saad suggests that optimal policy is found using the flat representation of the underlying MDP. The issue of how to create flat representations of MDPs when the trajectories can be computed is crucial for efficient implementations and

it is not addressed in [10]. For instance the MDP for the Merchant Problem discussed in [5] has over $4 \times 10^{15}$ trajectories. It would be impractical to compute them all. In contrast H-ASP# resolves the issue of computing the optimal policy without explicitly computing all the trajectories.

Baral et al. in [1] have introduced P-log - a declarative language based on ASP that combines logical and probabilistic arguments. Basic probabilistic information in P-log is expressed by probability atoms $pr\left(a\left(t\right)=y|_{c}\ B\right)=v$ where, intuitively, $a$ is caused by factors determined by $B$ with probability $v$. This is causal probability as described in [9]. Thus effect $a$ is independent of all factors except $B$ and the effects of $B$. The semantics of a probabilistic program $\Pi$ is based on the mapping of $\Pi$ to a logic programming $\Pi'$, and is given by the sets of beliefs of a rational agent associated with $\Pi$ together with their probabilities. There are significant differences between P-log and H-ASP#. First, H-ASP# allows the use of parameters over infinite sets and the use of arbitrary algorithms, which allow complex physical models to be created using H-ASP#. This is not the case with P-log. Second difference is that the probabilities in H-ASP# are assigned to states, whereas the probabilities in P-log are assigned to atoms. Assigning probabilities to states allows a somewhat greater flexibility in modeling probabilities, whereas assigning probabilities to atoms can produce somewhat more robust descriptions. Finally, H-ASP# provides a mechanism for constructing flat representations of MDPs whereas such functionality is not explicitly provided by P-log. Some of the similarities of two languages are that both use ASP and that both use unnormalized probabilities.

In this paper we have shown that H-ASP can be used to compute a finite horizon optimal policy for domains with transition densities finitely supported over infinite sets. We have demonstrated our approach using the Secret Agent Problem - a typical example for an application of MDPs. Our solution uses H-ASP# - a computer language based on H-ASP. The stable model of H-ASP# programs $W\#$ is defined as the unique maximal stable model of a H-ASP program $W$ which is derived from $W\#$. Under mild assumptions, the computational complexity of a H-ASP# programs is EXP-complete in the length of the program. We have implemented H-ASP# prototype solver and have create H-ASP# program $P\#$ that solves the Secret Agent Problem. $P\#$ contains 285 lines of code including comments. Computing the stable model of $P\#$ takes 4 minutes and 9 seconds on a 2.0 GHz Intel processor. In the process of the computation, 282921 states and 358120 transitions are generated. This is as expected given that H-ASP# program provides a compact representation of the MDP of the problem domain. Both $P\#$ and the MDP for the Secret Agent Problem can be found at

`http://math.ucsd.edu/~abrik/Secret_Agent/`

## References


[1] Chitta Baral, Michael Gelfond, and J. Nelson Rushton. Probabilistic reasoning with answer sets. In Vladimir Lifschitz and Ilkka Niemelä, editors, *LPNMR*, volume 2923 of *Lecture Notes in Computer Science*, pages 21–33. Springer, 2004.

[2] R. Bellman. *Dynamic programming.* Princeton University Press, 1957.

[3] Alex Brik. *Extensions of Answer Set Programming.* PhD thesis, UC San Diego, 2012.

[4] Alex Brik and Jeffrey B. Remmel. Hybrid ASP. In John P. Gallagher and Michael Gelfond, editors, *ICLP (Technical Communications)*, volume 11 of *LIPIcs*, pages 40–50. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[5] Alex Brik and Jeffrey B. Remmel. Computing a Finite Horizon Optimal Strategy Using Hybrid ASP. In *NMR*, 2012. Accepted.

[6] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080, 1988.

[7] Martin Mundhenk, Judy Goldsmith, Christopher Lusena, and Eric Allender. Complexity of finite-horizon markov decision process problems. *J. ACM*, 47(4):681–720, 2000.

[8] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[9] J. Pearl. *Causality: Models, Reasning and Inference.* Cambridge University Press, 2000.

[10] Emad Saad. A logical framework to reinforcement learning using hybrid probabilistic logic programs. In Sergio Greco and Thomas Lukasiewicz, editors, *SUM*, volume 5291 of *Lecture Notes in Computer Science*, pages 341–355. Springer, 2008.

[11] Emad Saad and Enrico Pontelli. A new approach to hybrid probabilistic logic programs. *Ann. Math. Artif. Intell.*, 48(3-4):187–243, 2006.

[12] Maarten H. van Emden and Robert A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976.