Computing a Finite Horizon Optimal Strategy Using Hybrid ASP

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

Abstract

In this paper we shall show how the extension of ASP called Hybrid ASP introduced by the authors in (Brik and Remmel 2011) can be used to combine logical and probabilistic reasoning. In particular, we shall show how Hybrid ASP can be used to create efficient, robust representations of dynamic domains whose stable models compute optimal finite horizon policies for the agents acting in such domains. We discuss a prototype implementation and show that the complexity of computing optimal policies is EXP-complete in the length of the input program.

Introduction

Hybrid Answer Set Programming (H-ASP) is an extension of Answer Set Programming (ASP) introduced by the authors in (Brik and Remmel 2011) that allows users to combine ASP type rules and numerical algorithms. The goal of this paper is to show how H-ASP can be used to create efficient, robust representations of dynamic domains whose stable models compute optimal finite horizon policies for the agents acting in the domains.

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. One of the properties of H-ASP is that while allowing the programmer to use both logical reasoning and numerical algorithms, the formalism keeps the numerical processing as separate from logic as possible. Thus the rules act as input-output devices for the algorithms. This feature of H-ASP is extremely conducive to a practical integration of logic rules and numerical processing.

Markov Decision Processes (MDPs) are widely used to model decision-making problems and were first described in (Bellman 1957). At a specific time, a decision maker observes the state of the system and decides upon 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 non-deterministically moves to a new state with a known probability, at which time a decision maker again observes the state of the system and decides upon which action to perform. The goal is to determine a policy that maximizes a cumulative expected reward.

There are many applications of MDPs, for instance: in communication networks (Altman 2000), in computer science (Strehl, Li, and Littman 2009), in finance (Trench et al. 2003) to name a few. Finding an optimal policy for an MDP is a well-studied topic in control theory (see (Puterman 1994) for an overview). 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 strategy. 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 that 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 logic and probability that are described in the literature. In (Saad 2008), Saad shows that normal hybrid probabilistic logic programs with answer set semantics introduced in (Saad and Pontelli 2006) 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.

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 Merchant Problem. It is a typical example of a problem for the DP algorithm.

The Merchant Problem.

Suppose that a certain merchant Q plans to travel from one coastal city C1 to another coastal city C5 by sea. On the way he plans to visit 3 more coastal cities C2, C3, C4 one after another, so that he will visit C1, C2, C3, C4, C5

in the specified sequence. Q owns a small cargo ship with a cargo capacity of 6 containers. Q has an expertise in trading three goods: flour, wine and wool. The prices of these goods under normal conditions in each city are known. However, the conditions in each city can change. First, the weather can change from normal to stormy. If that happens then the prices of the goods at a city will change in a specified way. Second, a city may choose to enact a tax on any subset of the three goods. This will also change the prices of goods in a specified way. The probability of a storm or of a particular tax for each city is known. The question is what should Q buy and sell in each city to maximize his total profit. Of course Q's action in each city will depend on the prices of goods at the city when Q arrives there and on the content of Q's cargo. Moreover we will assume that every transaction will result in buying or selling an integer number of containers of goods and that Q will not buy more goods than he can store in the ship's cargo.

A few comments about the problem description should be made. First, the problem description specifies the 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 Merchant 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 millions. 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 problem description of the Merchant Problem is changed by adding a fourth good that Q trades. 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 dynamic 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 Merchant 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 introduce a new language closely related to H-ASP called H-ASP#. We have implemented a prototype H-ASP# solver, created a H-ASP# program for the Merchant Problem, and have used the solver with the program to solve the problem.

In this paper we present a number of theoretical results. Because of the space constraints we are not able to include the proofs. However, the proofs can be found in (Brik 2012).

The rest of the paper is structured as follows. In section 2, we will review the DP algorithm. In section 3, we will review H-ASP. In section 4, we will show how the Merchant Problem can be solved using H-ASP. Section 5 will discuss the implementation related issues and the language H-ASP#. The same section will discuss the computational complexity of H-ASP# programs. We end with our conclusion and a discussion of related work.

The Dynamic Programming Algorithm

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, $T : S \times A \times S \to [0, 1]$ 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. $r : S \times A \to \mathbb{R}$ is the reward function, i.e. $r(s, a)$ is the reward gained by taking action a in state s.

A trajectory θ 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 π is a map $S \times \mathbb{Z}^+ \to A$ 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 κ be given. κ is the maximum length of the trajectories that we will be considering. The probability $\mathrm{Prob}_\pi(\theta)$ of a trajectory θ under a policy π is $\mathrm{Prob}_\pi(\theta) = \prod_{i=1}^{|\theta|-1} T(\theta_i, \pi(\theta_i, i), \theta_{i+1})$ where $|\theta|$ is the length of the trajectory. The reward $R^n_\pi(\theta)$ of a trajectory θ at time n under a policy π is $R^n_\pi(\theta) = \sum_{i=1}^{|\theta|} r(\theta_i, \pi(\theta_i, n + i - 1))$.

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 π 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 π. That is $R_\pi(s, i) = \sum_{\theta \in \Theta(s, \kappa+1-i)} \mathrm{Prob}_\pi(\theta) \cdot R^i_\pi(\theta)$.

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

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

The performance for a policy π 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 π^* such that for all $s \in S$ and for all the policies π $R_{\pi^*}(s, 1) \geq R_\pi(s, 1)$. π^* can be constructed as follows. First define $\pi^*(s, \kappa) = \mathrm{argmax}_{a \in A} r(s, a)$. 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 (Mundhenk et al. 2000).

Hybrid ASP

In this section we give a brief review of H-ASP.

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 $\widehat{M} = \{\mathbf{x} \in S : (\exists a \in At)((a,\mathbf{p}) \in M)\}$. A block B is an object of the form $B = a_1,\ldots,a_n,\ not\ b_1,\ldots,\ not\ b_m$ where $a_1,\ldots,a_n, b_1,\ldots,b_m \in At$. Given $M \subseteq At \times S$, $B = a_1,\ldots,a_n, not\ b_1,\ldots,not\ b_m$, 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 $B^- = not\ b_1,\ldots,not\ b_m$.

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 $a_1,\ldots, a_n, not\ b_1,\ldots, not\ b_m$ 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 $a_1,\ldots, a_n, not\ b_1,\ldots, not\ b_m$ 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.

The idea is that 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 is defined as follows. Let $M \subseteq At \times S$, let $I \in S$. Then we form the Gelfond-Lifschitz reduct of P over M and I, $P^{M,I}$ as follows. 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.
(1) Eliminate all rules with 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 $t(\mathbf{p}) = t$.

Given $M \subseteq At \times S$ and $\mathbf{p} \in S$, we define $W_M(\mathbf{p})$ - *the state at* $\mathbf{p}$ to be $W_M(\mathbf{p}) = \{a \mid (a, \mathbf{p}) \in M\}$. A *hybrid state* for $\mathbf{p}$ is a pair $\langle W_M(\mathbf{p}), \mathbf{p}\rangle$.

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 a $\mathbf{q}' \in A(\mathbf{p})$ such that $y(\mathbf{q}') = y'$ and $\mathbf{q}$ and $\mathbf{q}'$ are identical in all the parameter values except possibly y.

Solving the Merchant Problem using H-ASP

In this section we will describe our formalization of the Merchant Problem as a H-ASP program. Details about the H-ASP solver used to implement the program will be given in section 5. The construction of the desired H-ASP program will proceed in 3 steps. In step 1, we will discuss a H-ASP program P such that the single trajectory stable models of P describe the possible trajectories of the Merchant Problem. In step 2, we will describe a transform of P to a H-ASP program P'. P' has the property that its unique maximal stable model $M'_{\max}$ captures all the possible single trajectory stable models of P. $M'_{\max}$ can be computed by the Local Algorithm, also introduced in step 2. Finally in step 3, we will show how P' can be modified to a H-ASP program P'' that uses the DP algorithm to compute the optimal policy π^*.

Step 1: Modeling the Merchant Problem.

Let κ be a policy horizon. By a trajectory of the Merchant Problem we mean a finite sequence of the form $(s_1, a_1, s_2, a_2, \ldots, s_n, a_n)$ where $s_1, \ldots, s_n$ are states of the Merchant Problem domain, $a_1, \ldots, a_n$ are actions chosen by the merchant Q in the states $s_1, \ldots, s_n$ correspondingly. We will make the following assumptions about the H-ASP program P whose single trajectory stable models describe the trajectories of the Merchant Problem.

(1) Every initial condition I is such that $t(I) = \gamma$ for some fixed γ.

(2) All the advancing rules have the form $\frac{B:A,O}{a}$ where $\gamma \leq t(\mathbf{q}) =\leq \gamma + 2\kappa\Delta$ if $\mathbf{p} \in O$. Here $\Delta > 0$ is a program dependent constant.

(3) All the stationary rules have the form $\frac{B_1;B_2:H,O^*}{a}$ or $\frac{B:H,O}{a}$ where $t(\mathbf{p}) \geq \gamma$ if $\mathbf{p} \in O$ or $(\mathbf{p}, \mathbf{q}) \in O^*$.

(4) There exists $\Delta > 0$ such that if A is an advancing algorithm and $\mathbf{q} \in A(\mathbf{p})$, then $t(\mathbf{q}) = t(\mathbf{p}) + \Delta$.

(5) For every stationary rule $\frac{B_1;B_2:H,O}{a}$, $t(\mathbf{p}) + \Delta = t(\mathbf{q})$ if $(\mathbf{p}, \mathbf{q}) \in O$.

These conditions are designed to allow us to prove a number of useful properties of the stable models of P. In particular, if M is a single trajectory stable model of P, then every hybrid state $(U, \mathbf{q})$ depends on at most one other state $(V, \mathbf{p})$ with the property that $t(\mathbf{p}) + \Delta = t(\mathbf{q})$. Since in MDPs, every state depends only on the previous state, P will be able to describe an MDP.

We will need the following parameters: CARGO, PRICE, TAXES, PROFIT. With these parameters we will describe cargo contents, the price of goods at the current city, the taxes at the current city, and the immediate profit of the last action, respectively. We will also need the parameters Act, Prob, LEVEL. The values of the parameter CARGO will be triples with each triple specifying the number of cases of goods: flour, wool and wine in the cargo. The values of the parameter PRICE will be triples specifying the prices of goods in the current city. The values of the parameter TAXES will be triples specifying whether taxes on goods are enacted. The values of PROFIT will be real numbers. The meaning of Act, Prob, LEVEL will be explained further in this section.

The set of atoms At will contain atoms: STORM, BUY, SELL, MOVE, FAIL. The atom STORM will model the presence or the absence of a storm at the current city. The atoms BUY, SELL, MOVE will specify whether the next action is to buy, to sell or to move to the next city respectively. FAIL will be used for constraints as in $\frac{not\ \text{FAIL}}{\text{FAIL}}$ and thus will never be part of a stable model.

The merchant will first choose an action of the type $(\text{buy}, b_1, b_2, b_3)$ where b_1, b_2, b_3 are the numbers of cases of 3 goods that he will buy. At the next time step he will move to the next city by choosing an action (move). Then the merchant will sell a certain amount of goods by choosing an action of the form $(\text{sell}, x_1, x_2, x_3)$ where x_1, x_2, x_3 are the amounts of goods that he wants to sell. Then the cycle repeats with the merchant buying goods and moving to the next city. The process will stop after 12 steps as merchant sells some of his goods at the last city.

P will have to derive action choices, derive consequences of actions and derive the transition probabilities. To achieve this, P will be partitioned into three disjoint sets of rules $P = P_A \cup P_C \cup P_P$ where P_A are the rules for deriving action choices, P_C are the rules for deriving the consequences of actions and P_P are the rules for deriving the probabilities. The parameter Act will be used to model the actions. To model probabilities, we will use the parameter Prob.

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 $\text{LEVEL}(\mathbf{p}) = 0$ 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 $t(\widehat{\mathbf{p}}) = t(\mathbf{p}) + \Delta$, $\text{Act}(\widehat{\mathbf{p}}) = a$ and $\text{LEVEL}(\widehat{\mathbf{p}}) = 1$. We will ensure that if $O \subseteq X$ ($O \subseteq X^2$) is a constraint set occurring in $P_C \cup P_P$, then $\text{LEVEL}(\mathbf{p}) = 0$ whenever $\mathbf{p} \in O$ ($(\mathbf{q}, \mathbf{p}) \in O$). On the other hand, if $O \subseteq X$ ($O \subseteq X^2$) is a constraint set occurring in P_A, then $\text{LEVEL}(\mathbf{p}) = 1$ whenever $\mathbf{p} \in O$ ($(\mathbf{q}, \mathbf{p}) \in O$). We will also require that for all the advancing algorithms G in P_A, if $\mathbf{q} \in G(\mathbf{p})$, then $\text{LEVEL}(\mathbf{q}) = 1$ and, for all the advancing algorithms G in a rule in P_C, if $\mathbf{q} \in G(\mathbf{p})$, then $\text{LEVEL}(\mathbf{q}) = 0$.

For example the following advancing rule generates ac-

tion of the form (move):

$$\frac{\mathrm{MOVE}:\ \mathrm{isLevel0},\ \mathrm{actMove}}{\mathrm{MOVE}} \tag{3}$$

Here $\mathrm{isLevel0}(\mathbf{p}) = 1$ iff $\mathrm{LEVEL}(\mathbf{p}) = 0$. $\mathrm{actMove}(\mathbf{p}) = \{\widehat{\mathbf{p}}\}$ where $\widehat{\mathbf{p}}$ is identical to $\mathbf{p}$ except that $t(\widehat{\mathbf{p}}) = t(\mathbf{p}) + \Delta$, $Act(\widehat{\mathbf{p}}) = (\mathrm{move})$ and $\mathrm{LEVEL}(\widehat{\mathbf{p}}) = 1$.

P_P will consist only of the stationary rules of the form $\frac{B_1;B_2,\ not\ \mathrm{FAIL}:H,O}{\mathrm{FAIL}}$. Here for $(\widehat{\mathbf{p}}, \mathbf{q}) \in O$, $H((\widehat{\mathbf{p}}, \mathbf{q})) = 1$ iff the *unnormalized probability* of transitioning from a hybrid state $\langle W, \widehat{\mathbf{p}}\rangle$ satisfying $W \models B_1$ to a hybrid state $\langle U, \mathbf{q}\rangle$ satisfying $U \models B_2$ under action $\mathrm{Act}(\widehat{\mathbf{p}})$ is not $\mathrm{Prob}(\mathbf{q})$. That is, the rule restricts transitions $(\widehat{\mathbf{p}}, \mathbf{q})$ only to those where the value of $\mathrm{Prob}(\mathbf{q})$ is the correct one. Since the rules of P_P will only restrict the generalized positions, every advancing algorithms in P_C will let Prob be free.

Intuitively, the relation between the unnormalized probabilities of transitioning and the (normalized) probabilities of transitioning is as follows. If F is the set of the generalized positions corresponding to all the possible transitions from the generalized position $\widehat{\mathbf{p}}$ and $\mathbf{q} \in F$, then the probability of transitioning from $\widehat{\mathbf{p}}$ to $\mathbf{q}$ is $\mathrm{Prob}(\mathbf{q}) / \sum_{\mathbf{r}\in F} \mathrm{Prob}(\mathbf{r})$. We can use the unnormalized probabilities since the denominator can be computed later in 1 and 2.

Using unnormalized probabilities in the algorithms rather than (normalized) probabilities allows one to easily model a statement such as: a storm is twice as likely as its absence. What is required is that in the event of a storm the unnormalized probability is multiplied by 2.

Due to space restrictions we will not reproduce the program P here. The program can be found in (Brik 2012).

Step 2: The program transform.

To apply the DP algorithm we will transform P to another H-ASP program P'. P' will have a unique maximal stable model $M'_{\max}$ and all the single trajectory stable models of P' will correspond to certain subsets of $M'_{\max}$. Thus $M'_{\max}$ can be used to create a flat representation of the MDP for the Merchant Problem.

P' needs to be different from P because of the following two issues.

(A) For two hybrid states $\langle A_1, \mathbf{p}\rangle$ and $\langle A_2, \mathbf{p}\rangle$ where $A_1 \neq A_2$, $\langle A_1, \mathbf{p}\rangle$ and $\langle A_2, \mathbf{p}\rangle$ cannot both be hybrid states in the same stable model of P. Since it is possible that for some generalized position $\mathbf{p}$, $\langle A_1, \mathbf{p}\rangle$ is a hybrid state in one of the single trajectory stable models of P and $\langle A_2, \mathbf{p}\rangle$ is a hybrid state in another single trajectory stable model of P, it is possible that no stable model of P will capture all the single trajectory stable models of P.

(B) For generalized positions $\mathbf{p}$ and $\mathbf{q}$ with $t(\mathbf{q}) = t(\mathbf{p}) + \Delta$, it is not clear how to determine whether a hybrid state $\langle B, \mathbf{q}\rangle$ is a successor of the hybrid state $\langle A, \mathbf{p}\rangle$. Since this information is needed to construct a flat representation of a MDP, stable models of P cannot be used to construct flat representations of MDPs.

The first problem occurs because the elements of the stable models are pairs $(a, \mathbf{p})$. Hence, at most one hybrid state can correspond to a generalized position $\mathbf{p}$. The second problem occurs because in a stable model, there may be two generalized positions $\mathbf{p}$ and $\mathbf{r}$ with $\mathbf{p} \neq \mathbf{r}$ and $t(\mathbf{p}) = t(\mathbf{r})$. Thus if $t(\mathbf{q}) = t(\mathbf{p}) + \Delta$, $\mathbf{q}$ could be a successor of either $\mathbf{p}$ or $\mathbf{r}$.

We will introduce two parameters AV and prev that will be part of the parameter space X' of P' in addition to the parameters of P. The parameter AV will encode a set of atoms of a hybrid state and the parameter prev will record the information necessary to determine a predecessor generalized position.

The Program Transform is defined as follows.

(I) Define the parameter space X' for P' to be the set of all tuples of the form $\mathbf{p}' = (t, x_1, \ldots, x_{n+2})$ where $\mathbf{p} = (t, x_1, \ldots, x_n) \in X$. The value x_{n+2} denoted $\mathrm{AV}(\mathbf{p}')$ is a bit vector of length $|At|$. The value x_{n+1} denoted $\mathrm{prev}(\mathbf{p}')$ has the form $(t - \Delta, y_1, \ldots, y_n, z)$ where $(t - \Delta, y_1, \ldots, y_n) \in X$ and z is a bit vector of length $|At|$.

(II) Suppose that the initial condition I has the form $(\gamma, x_1, x_2, \ldots, x_n)$. Define $J(I) = (\gamma - \Delta, x_1, \ldots, x_n, \emptyset, 0)$ where 0 denotes the vector of zeros. $J(I)$ will be the initial condition of P'.

(III) For $\mathbf{q} = (t, x_1, \ldots, x_{n+2}) \in X'$ the projection of $\mathbf{q}$ onto X is $\Pi_X(\mathbf{q}) = (t, x_1, \ldots, x_n)$. We need to change the advancing rules of P so that the algorithms and the constraint sets of P' ignore the new parameters AV and prev. That is for every advancing rule $\frac{B:A,O}{a} \in P$, P' has an advancing rule $\frac{B:A',O'}{a}$ where for all $\mathbf{q} = (t, x_1, \ldots, x_{n+2}) \in X'$ $A'(\mathbf{q}) = \{(t + \Delta, y_1, \ldots, y_{n+2}) \mid \exists (t + \Delta, y_1, \ldots, y_n) \in A(\Pi_X(\mathbf{q}))$ and $y_{n+1} = (t, x_1, \ldots, x_n, x_{n+2})$ and $\mathbf{q} \in O' \longleftrightarrow \Pi_X(\mathbf{q}) \in O\}$. Note that A' lets the last parameter AV be free.

(IV) As in (III) above, the stationary rules of P are modified to ignore the new parameters AV and prev except when it is necessary to determine whether $\mathbf{q}$ is a successor of $\mathbf{p}$. That is, for every stationary rule $\frac{B:H,O}{a} \in P$, P' has a stationary rule $\frac{B:H',O'}{a}$ where for all $\mathbf{q} \in X'$, $\mathbf{q} \in O'$ iff $\Pi_X(\mathbf{q}) \in O$ and $H'(\mathbf{q}) = 1$ iff $H(\Pi_X(\mathbf{q})) = 1$. Similarly, for every stationary rule $\frac{B_1;B_2:H,O}{a} \in P$, P' has a stationary rule $\frac{B_1;B_2:H',O'}{a}$ where for all $\mathbf{q}_1 = (t, x_1, \ldots, x_{n+2}) \in X'$ and for all $\mathbf{q}_2 \in X'$, $(\mathbf{q}_1, \mathbf{q}_2) \in O'$ iff $[(\Pi_X(\mathbf{q}_1), \Pi_X(\mathbf{q}_2)) \in O$ & $\mathrm{prev}(\mathbf{q}_2) = (t, x_1, \ldots, x_n, x_{n+2})]$ and $H'((\mathbf{q}_1, \mathbf{q}_2)) = 1$ iff $H((\Pi_X(\mathbf{q}_1), \Pi_X(\mathbf{q}_2))) = 1$.

(V) We will now specify how the parameter AV is used to encode the set of atoms of the hybrid states. Let $(a_1, \ldots, a_{|At|})$ be a fixed enumeration of At, and for $a_j \in At$ define $i(a_j) = j$. Then for each $a \in At$, P' will contain the following two stationary rules $\frac{a,\ not\ \mathrm{FAIL}:H[a],\ X'}{\mathrm{FAIL}}$ and $\frac{not\ a,\ not\ \mathrm{FAIL}:\ \overline{H[a]},\ X'}{\mathrm{FAIL}}$ where $H[a]$ and $\overline{H[a]}$ are Boolean algorithms and for all $\mathbf{q} \in S'$ $H[a](\mathbf{q}) = 1$ if $AV(\mathbf{q})(i(a)) = 0$ and $H[a](\mathbf{q}) = 0$ otherwise. $\overline{H[a]}(\mathbf{q}) = AV(\mathbf{q})(i(a))$. It is easy to see that the rules enforce the following condition on any stable

model M' of P' with the initial condition $J(I)$

$$\forall \mathbf{q} \in \widehat{M'} \cup \{J(I)\}\ \forall a \in At\ (a, \mathbf{q}) \in M' \longleftrightarrow$$

$$AV(\mathbf{q})(i(a)) = 1.$$

Thus for a stable model M' of P' with the initial condition $J(I)$, a generalized position $\mathbf{q} \in \widehat{M'} \cup \{J(I)\}$ contains information necessary to determine the set of atoms $W_{M'}(\mathbf{q})$ corresponding to a hybrid state for $\mathbf{q}$.

(VI) We need a rule that will copy the parameter values of the initial condition $J(I)$ to the time γ. That is we add to P' the advancing rule $\frac{:A_J,O_J}{\text{new}}$ where for $\mathbf{p} \in X' \cup J(I)$ $\mathbf{p} \in O_J \longleftrightarrow t(\mathbf{p}) = \gamma - \Delta$ and for $\mathbf{p} = (\gamma - \Delta, x_1, \ldots, x_{n+2})$ $A_J(\mathbf{p}) = \{(\gamma, x_1, \ldots, x_n, \emptyset, z) |\ z$ is a bit vector of length $|At|\}$.

(VII) Besides the rules in (III)-(VI), P' has no other rules.

Thus the transform of rule 3 is $\frac{\text{MOVE: isLevel0}', \text{ actMove}'}{\text{MOVE}}$ where $\text{isLevel0}'(\mathbf{p}) = 1$ iff $\text{LEVEL}(\mathbf{p}) = 0$. $\text{actMove}'(\mathbf{p}) = \{\widehat{\mathbf{p}}\}$ where $\widehat{\mathbf{p}}$ is identical to $\mathbf{p}$ except that $t(\widehat{\mathbf{p}}) = t(\mathbf{p}) + \Delta$, $Act(\widehat{\mathbf{p}}) = (\text{move})$, $\text{LEVEL}(\widehat{\mathbf{p}}) = 1$ and if $\mathbf{p} = (x_1, ..., x_{n+2})$ then $\text{prev}(\widehat{\mathbf{p}}) = (x_1, ..., x_n, x_{n+2})$.

Theorem *There exists a bijection between the set of the single trajectory stable models of P with the initial condition I and the set of the single trajectory stable models of P' with the initial condition $J(I)$.*

The proof shows how to construct such a bijection and is omitted due to the considerations of space. The theorem shows that the single trajectory stable models of P' with the initial condition $J(I)$ describe the trajectories of the MDP for the Merchant Problem.

Step 3: Computing the optimal policy.

Given the H-ASP program P' which is the transform of P we want to be able to construct a flat representation of the MDP for the Merchant Problem. To do so, for every hybrid state $\langle A, \mathbf{p}\rangle$ of a single trajectory stable model of P' with the initial condition $J(I)$ we need to be able to compute all the possible successor hybrid states of $\langle A, \mathbf{p}\rangle$. That is, we want to be able to compute the set $L(A, \mathbf{p})$ of hybrid states where for each $\langle B, \mathbf{q}\rangle \in L(A, \mathbf{p})$ there exists a single trajectory stable model M' of P' with the initial condition $J(I)$ and $\langle A, \mathbf{p}\rangle$ and $\langle B, \mathbf{q}\rangle$ are both hybrid states in M' and $t(\mathbf{p}) + \Delta = t(\mathbf{q})$. Also, for each single trajectory stable model M' of P' with the initial condition $J(I)$ if $\langle A, \mathbf{p}\rangle$, $\langle C, \mathbf{r}\rangle$ are hybrid states of M' and $t(\mathbf{r}) = t(\mathbf{p}) + \Delta$ then $\langle C, \mathbf{r}\rangle \in L(A, \mathbf{p})$.

This can be accomplished by the Local Algorithm. We will state the Local Algorithm and then we will show that its output defines the unique maximal stable model $M'_{\max}$ of P' with the initial condition $J(I)$.

Let $K \subseteq At \times S'$ and let $\mathbf{p} \in X'$. Define $\text{Adv}(K, \mathbf{p}) = \{(a, \mathbf{q}) |$ there exists an advancing rule $\frac{B:A',O'}{a} \in P'$ and $\mathbf{p} \in O'$ and $K \models (B, \mathbf{p})$ and $\mathbf{q} \in A'(\mathbf{p})\}$.

The Local Algorithm produces a sequence $\{L_i\}_{i=-1}^{\infty}$ where $L_i \subseteq At \times X'$ for $i = -1, 0, \ldots$

Define $L_{-1} = \emptyset$.

Assume that L_i is defined. Let $\mathbf{p} \in \widehat{L_i}$ if $i \geq 0$ and $\mathbf{p} = J(I)$ if $i = -1$. For $\mathbf{q} \in \widehat{Adv(L_i, \mathbf{p})}$, $A \subseteq At$ define $P^A(\mathbf{p}, \mathbf{q})$ as follows.

(1) For a stationary rule $\frac{B:H',O'}{a} \in P'$, if $\mathbf{q} \in O'$, $H'(\mathbf{q}) = 1$, and $A \models B^-$, then $\frac{B^+}{a} \in P^A(\mathbf{p}, \mathbf{q})$.

(2) For a stationary rule $\frac{B_1;B_2:H',O'}{a} \in P'$, if $(\mathbf{p}, \mathbf{q}) \in O'$, $H'((\mathbf{p}, \mathbf{q})) = 1$, $L_i \models (B_1, \mathbf{p})$, and $A \models B_2^-$, then $\frac{B_2^+}{a} \in P^A(\mathbf{p}, \mathbf{q})$.

Define $U_i(A, \mathbf{p}, \mathbf{q}) = \bigcup_{k=0}^{\infty} T^k_{P^A(\mathbf{p},\mathbf{q})}\left(W_{Adv(L_i,\mathbf{p})}(\mathbf{q})\right)$.

We let

$$L_{i+1} = \bigcup_{\mathbf{p} \in L'_i} \bigcup_{\mathbf{q} \in Adv(L_i, \mathbf{p})} \bigcup_{\substack{A \subseteq At \\ A = U_i(A, \mathbf{p}, \mathbf{q})}} Z(A, \mathbf{q})$$

where $L'_i = \widehat{L_i}$ and $Z(A, \mathbf{q}) = M(A, \mathbf{q})$ if $i \geq 0$ and $L'_i = \{J(I)\}$ and $Z(A, \mathbf{q}) = M(A \cup \{\text{new}\}, \mathbf{q})$ if $i = -1$.

In other words, we take every generalized position in $\widehat{L_i}$ (or $J(I)$ if $i = -1$), and use the advancing rules to generate successor generalized positions. For each successor position $\mathbf{q}$ and each $A \subseteq At$, we determine whether $\langle A, \mathbf{q}\rangle$ corresponds to a valid hybrid state, and, if so, then $Z(A, \mathbf{q})$ is added to L_{i+1}.

We will define $M'_{\max} = \bigcup_{i=0}^{\infty} L_i$. We can then prove the following theorem.

Theorem. *$M'_{\max}$ is the unique maximal stable model of P' with the initial condition $J(I)$. Moreover, if N is a stable model of P' with the initial condition $J(I)$, then $N \subseteq M'_{\max}$ and, for all $\mathbf{p} \in \widehat{N}$, $W_N(\mathbf{p}) = W_{M'_{\max}}(\mathbf{p})$.*

The theorem shows that we can use the Local Algorithm to generate the unique maximal stable model $M'_{\max}$ of P' and that all the single trajectory stable models of P' are part of $M'_{\max}$.

Let $Y \subseteq X'$ and let $\mathbf{q} = (t, x_1, \ldots, x_n, x_{n+2})$. Define $Ext_Y(\mathbf{q}) = \{\mathbf{q}' \in Y |\ \mathbf{q}' = (t, x_1, \ldots, x_n, z, x_{n+2})$ for some $z\}$. A set $B \subseteq X'$ is called a **chain** if

(1) $\forall \mathbf{q}_1, \mathbf{q}_2 \in B\ (t(\mathbf{q}_1) = t(\mathbf{q}_2) \rightarrow \mathbf{q}_1 = \mathbf{q}_2)$ and

(2) $\forall \mathbf{q} \in B\ (\text{prev}(\mathbf{q}) = \emptyset$ or $|Ext_B(\text{prev}(\mathbf{q})| = 1)$.

Theorem *There exists a bijection between the set of chains of $M'_{\max}$ and the set of single trajectory stable models of P' with the initial condition $J(I)$.*

The proof constructs the bijection and is omitted due to the considerations of space. Since we have already shown that there exists a bijection between the set of single trajectory stable models of P and the set of single trajectory stable models of P', we have established the existence of a bijection between the set of chains of $M'_{\max}$ and the set of single trajectory stable models of P. Thus the Local Algorithm can be used to generate the MDP for the Merchant Problem.

It is important to note that the Local Algorithm takes a generalized position as its input. Thus the Local Algorithm can be used within a H-ASP program in the same way as any other algorithm. Thus to compute a finite-horizon optimal

policy we will transform P' to another H-ASP program P'' which in addition to the rules of P' will contain extra rules that will use the Local Algorithm to compute the optimal policy. P'' is defined as follows.
(1) Add to the parameters of X' a parameter LA that will be used to store an optimal policy. The new parameter space will be denoted X''.
(2) Replace the rule $\frac{:A_J,O_J}{\text{new}}$ of P' that was to be applied at time $t = \gamma$ by a new rule that will compute an optimal policy. That is, replace $\frac{:A_J,O_J}{\text{new}}$ by the new rule $\frac{:A''_J,O''_J}{new}$ where $\mathbf{p} \in O''_J$ implies $t(\mathbf{p}) = \gamma - \Delta$ and, for any $\mathbf{p} = (\gamma - \Delta, x_1, \ldots, x_{n+2}, \pi) \in O''_J$,

$$A''_J(\mathbf{p}) =$$

$$\{(\gamma, x_1, \ldots, x_n, \emptyset, z, \pi^*) | (\gamma, x_1, \ldots, x_n, \emptyset, z) \in \widehat{L_0}\}$$

where L_0 is produced by the Local Algorithm run on the input $\mathbf{p}$, π^* is the policy computed by the DP algorithm when it is used on the MDP generated from the output of the Local Algorithm run on the input $\mathbf{p}$. That is, A''_J will run the Local Algorithm and generate the flat representation of the MDP for the Merchant Problem with the initial states corresponding to the set of hybrid states at time γ. A''_J will then run the DP algorithm on the flat representation to compute an optimal policy π^*.
(3) We need to modify the algorithms and the constraint sets of P' to ignore the new parameters. That is every advancing algorithm A' in P' is replaced by an advancing algorithm A'' such that, for every $\mathbf{r} = (t, x_1, \ldots, x_{n+2}, \pi^*) \in X''$, $A''(\mathbf{r}) = \{(s, y_1, \ldots, y_{n+2}, \pi^*) \mid (s, y_1, \ldots, y_{n+2}) \in A'((t, x_1, \ldots, x_{n+2}))\}$, except for A_J in item 2 above. Every constraint set $O' \subseteq X'$ is replaced by the constraint set $O'' \subseteq X''$ where $(t, x_1, \ldots, x_{n+2}, \pi^*) \in O''$ iff $(t, x_1, \ldots, x_{n+2}) \in O'$. The replacement of a constraint set $O' \subseteq X'^2$ and Boolean algorithms with the domain X' and X'^2 is done analogously.

For $I = (\gamma, x_1, \ldots, x_n)$, the initial condition for P'' is $J''(I) = (\gamma - \Delta, x_1, \ldots, x_n, \emptyset, 0, 0)$. We will assume that P' is such that for every $\mathbf{p} \in \widehat{M'_{\max}}$, $\text{Prob}(\mathbf{p}) \in \mathbb{R}^+ \cup \{0\}$ and $\text{PROFIT}(\mathbf{p}) \in \mathbb{R}$.

We can prove that for every stable model M' of P' with the initial condition $J(I)$ there exists a unique stable model $U(M')$ of P'' with the initial condition $J''(I)$ such that $(a, \mathbf{p}) | \mathbf{p} = (t, x_1, \ldots, x_{n+2}) \in M'$ iff $(a, (t, x_1, \ldots, x_{n+2}, \pi^*)) \in U(M')$ for an optimal policy π^*. Similarly, for every stable model M'' of P'' with the initial condition $J''(I)$, $\{(a, \mathbf{p}) | \mathbf{p} = (t, x_1, \ldots, x_{n+2})$ and $(a, (t, x_1, \ldots, x_{n+2}, \pi^*) \in M''\}$ is a stable model of P' with the initial condition $J(I)$.

It follows that $U(M'_{\max})$ is the unique maximal stable model of P'' with the initial condition $J''(I)$. The concept of a chain for X'' can be defined similarly to the concept of a chain for X'. Then there exists a bijection between the chains of $U(M'_{\max})$ and the single trajectory stable models of P'' with the initial condition $J''(I)$. Hence, an optimal policy for the Merchant Problem is given by the values of LA parameter of the elements of $\widehat{U(M'_{\max})}$.

Thus our solution to the Merchant Problem is as follows.
(1) Construct a H-ASP program P and the initial condition I such that single trajectory stable models of P with the initial condition I describe the trajectories of the Merchant Problem. P will satisfy the assumptions defined in Step 1.
(2) Transform P to P'. Modify P' to obtain P''.
(3) Run the Local Algorithm with the input $J(I)$ to compute $M'_{\max}$ and the flat representation $G(M'_{\max})$ of the MDP for the Merchant Problem.
(4) Use the DP algorithm on $G(M'_{\max})$ to construct an optimal policy π^*.
(5) Construct $U(M'_{\max})$ as $U(M'_{\max}) = \{(a, (t, x_1, \ldots, x_{n+2}, \pi^*) | (a, (t, x_1, \ldots, x_{n+2})) \in M'_{\max}\}$.

Implementation

Because of the large number of generalized positions generated by the advancing algorithms, to implement our solution, we introduce a slight variation of H-ASP called H-ASP#. The semantics of a H-ASP# program $W\#$ is given by the unique maximal stable model of W'' where W is the translation of $W\#$ into H-ASP and is a program of type discussed in step 1. W'' is the result of performing the program transforms of step 2 and step 3 on W.

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 (Brik and Remmel 2011), 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 $S_1, S_2, \ldots, S_n$ 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 $i_1, i_2, \ldots, i_k$ and let 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 $i = 1, \ldots, n$ $\frac{not\ S_i,\ not\ \text{FAIL}}{\text{FAIL}}$.

We will assume that this mechanism is used in any H-ASP programs that we will be considering. Then we can implement the mechanism implicitly. 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 $\text{param}(p)$ where p is a parameter to be used by the H-ASP# program. Parameters Act and Prob do not need to be declared. An algorithm declaration has the form $\text{program}\ p(a_1, \ldots, a_k)\ \{c_1; \ldots;\ c_m\}$ 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 $\mathrm{sig}(p, (s_1, \ldots, s_k))$ 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 representation of an atom, B_1, B_2 are of the form $c_1, \ldots, c_k, not\ d_1, \ldots, not\ d_m$ 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 the 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 is a number of other optimizations introduced in H-ASP#. However we have to omit their description due to the considerations of space.

We note that the transform of P to P' and of P' to P'' can be applied to any H-ASP program W satisfying the criteria of step 1. We will use this fact to define the stable model semantics of H-ASP#.

The stable model of the H-ASP# program $W\#$ is the unique maximal stable model of W'' with the initial condition $J''(I)$ where W is the H-ASP program obtained from $W\#$ by explicitly stating all the implicit assumptions of H-ASP#, and I is the initial condition specified in $W\#$.

Let $W\#$ be a H-ASP# program , let I be an initial condition, let κ be the horizon. Let $\{L_i\}_{i=-1}^{\infty}$ be the output of the Local Algorithm for W' and $J(I)$. For $\mathbf{p} \in \widehat{L_i}$, define $\mathrm{Succ}(\mathbf{p}) = \{\mathbf{q} | \mathbf{q} \in L_{i+1} \text{ and } \mathbf{p} \in Ext_{L_i}(\mathrm{prev}(\mathbf{q}))\}$. $W\#$ is said to *generate a MDP* if for $i = 0, 1, \ldots, 2\kappa$ for $\mathbf{p} \in L_i$ $|\mathrm{Succ}(\mathbf{p})| \geq 1$ and, if $\mathrm{LEVEL}(\mathbf{p}) = 1$, then there exists $\mathbf{q} \in \mathrm{Succ}(\mathbf{p})$ such that $\mathrm{Prob}(\mathbf{q}) > 0$. That is, intuitively $W\#$ generates a MDP if none of the branches of its trajectory tree terminate in less than κ steps.

The computation with $W\#$ uses the Modified Local Algorithm where the main difference with the version of step 3 is that the Modified Local Algorithm needs to assemble generalized positions from the outputs of H-ASP# advancing algorithms that specify only some of the parameter values. The Modified Local Algorithm processes fewer generalized positions at every step than the Local Algorithm since the generalized position where some parameters are not set are not generated.

The Parameter Restriction Mechanism allows us to avoid the explosion in the number of produced generalized positions. Because of this we can prove the following result.

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 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 κ 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 (Papadimitriou and Tsitsiklis 1987).

Conclusion

The main advantages of using H-ASP or 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 logic and probability. However to our knowledge only the work of Saad (Saad 2008) addresses the problem of computing optimal policies of MDPs. In (Saad 2008) Saad introduces a Markov Action Language $\mathcal{A}_{MD}$ which allows to describe MDPs. There are three main differences between the present work and (Saad 2008). 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 everything else being equal the storm is twice as likely as its absence. This condition can be easily modeled using unnormalized probabilities by multiplying by 2 the unnormalized probability of a state in case of storm, and not multiplying the unnormalized probability in case of storm's absence. 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 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. (Saad 2008) 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 (Saad 2008). For instance the MDP for the Merchant Problem has over 4×10^{15} trajectories. It would be impractical to compute them all. In contrast the present paper resolves the issue of computing the optimal policy within the syntax and semantics of H-ASP#.

Baral et al. in (Baral, Gelfond, and Rushton 2004) 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 (Pearl 2000). Thus effect a is independent of all factors except B and the effects of B. The semantics of a probabilistic program Π is based on the mapping of Π to a logic programming Π', and is given by the sets of beliefs of a rational agent associated with Π together with their probabilities. There are significant differences between P-log and H-ASP#. First, H-ASP# allows the use of arbitrary algorithms, which allows 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 a dynamic domain. We have demonstrated our approach using the Merchant Problem - a typical textbook example for an application of MDPs and Dynamic Programming. Our solution is based on considering a certain subset of H-ASP programs, so that each program describes valid trajectories of the problem domain. Such a program W is transformed to another program W'' with the property that the unique maximal stable model of W'' captures all the valid trajectories of the problem domain and specifies an optimal policy. To implement our approach, we have described H-ASP# - a modification of H-ASP. 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 Merchant Problem. $P\#$ contains 384 lines of code including comments. Computing the stable model of $P\#$ takes 4 minutes and 2 seconds on a 2.5 GHz Intel processor. In the process of the computation, 1029276 states and 2499727 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 Merchant Problem can be found at

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

References

Altman, E. 2000. Applications of markov decision processes in communication networks: a survey. *Rapport de Recherche - Institut National de Recherche en Informatique et en Automatique*.

Baral, C.; Gelfond, M.; and Rushton, J. N. 2004. Probabilistic reasoning with answer sets. In Lifschitz, V., and Niemelä, I., eds., *LPNMR*, volume 2923 of *Lecture Notes in Computer Science*, 21–33. Springer.

Bellman, R. 1957. *Dynamic programming*. Princeton University Press.

Brik, A., and Remmel, J. B. 2011. Hybrid ASP. In Gallagher, J. P., and Gelfond, M., eds., *ICLP (Technical Communications)*, volume 11 of *LIPIcs*, 40–50. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Brik, A. 2012. *Extensions of Answer Set Programming*. Ph.D. Dissertation, UC San Diego.

Mundhenk, M.; Goldsmith, J.; Lusena, C.; and Allender, E. 2000. Complexity of finite-horizon markov decision process problems. *J. ACM* 47(4):681–720.

Papadimitriou, C. H., and Tsitsiklis, J. N. 1987. The complexity of markov decision processes. *Mathematics of Operations Research* 12(3):441–450.

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

Puterman, M. 1994. *Markov decision processes: discrete stochastic dynamic programming*. Wiley series in probability and statistics. Wiley-Interscience.

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

Saad, E. 2008. A logical framework to reinforcement learning using hybrid probabilistic logic programs. In Greco, S., and Lukasiewicz, T., eds., *SUM*, volume 5291 of *Lecture Notes in Computer Science*, 341–355. Springer.

Strehl, A. L.; Li, L.; and Littman, M. L. 2009. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research* 10:2413–2444.

Trench, M. S.; Pederson, S. P.; Lau, E. T.; Ma, L.; Wang, H.; and Nair, S. K. 2003. Managing credit lines and prices for bank one credit cards. *Interfaces* 33:4–21.