Proceedings of the 23rd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Probabilistic Reasoning within Answer Set Programming with Quantifiers

Damiano Azzolini[1] , **Giuseppe Mazzotta**[2] , **Francesco Ricca**[2]
[1]University of Ferrara
[2]University of Calabria
damiano.azzolini@unife.it, {giuseppe.mazzotta, francesco.ricca}@unical.it

Abstract

Answer Set Programming with Quantifiers (ASP(Q)) extends Answer Set Programming (ASP) by allowing quantification over answer sets. Although probabilistic extensions to ASP exist, there is no such counterpart for ASP(Q). In this paper, we close this gap by introducing Inferential Quantified Answer Set Programming (ASP(Q)^{Inf}), an extension of ASP(Q) that supports probabilistic inference over programs with alternating quantifiers, allowing uncertainty at the innermost level. We demonstrate the modeling capabilities of ASP(Q)^{Inf}, analyze its computational complexity, and present an implementation based on Algebraic Model Counting. An experimental evaluation confirms its effectiveness and practical applicability.

1 Introduction

Answer Set Programming (ASP) (Brewka, Eiter, and Truszczynski 2011) is a well-established declarative knowledge representation and reasoning formalism for combinatorial problem solving (Falkner et al. 2018; Erdem, Gelfond, and Leone 2016), supported by the availability of efficient and mature solvers (Gebser et al. 2019; Alviano et al. 2017). During the years ASP has been extended in several directions with the aim of improving the framework to model different possible application scenarios (Baselice, Bonatti, and Gelfond 2005; Gebser et al. 2016; Eiter, Redl, and Schüller 2016; Amendola, Ricca, and Truszczynski 2019).

Concerning the expressive power capabilities of the language, ASP can solve problems up to the second level (Dantsin et al. 2001) of the Polynomial Hierarchy (PH) (Stockmeyer 1976). From the perspective of the knowledge engineer, ASP often excels at modeling problems in the NP complexity class; however, modeling problems in Σ_2^P typically becomes cumbersome, as it requires the use of advanced yet non-intuitive techniques such as saturation (Eiter and Gottlob 1995). To overcome these limitations, Answer Set Programming with Quantifiers, ASP(Q) (Amendola, Ricca, and Truszczynski 2019), was proposed. By allowing quantification over answer sets, ASP(Q) enhances the expressive power of ASP, enabling intuitive encodings of problems across the *entire* PH.

On the other hand, concerning the modeling of uncertainty and performing probabilistic inference, several probabilistic extensions of ASP have been proposed (Cozman and Mauá 2020; Baral, Gelfond, and Rushton 2009; Lee and Wang 2016; Lee and Yang 2017; Totis, De Raedt, and Kimmig 2023). These approaches aim to address, within the ASP framework, one of the central challenges of Statistical Relational AI, namely the integration of logical reasoning and probabilistic inference (Raedt et al. 2016). Among these approaches, Probabilistic ASP (PASP) (Cozman and Mauá 2020) extends ASP with the ability to handle uncertainty while preserving the expressive power of ASP.

While probabilistic extensions of ASP enable reasoning under uncertainty, and ASP(Q) provides the expressive power required to model problems across the entire PH, these two lines of research have so far remained largely disconnected. This naturally leads to the following research questions: *How can ASP(Q) be extended with probabilistic inference in order to combine the benefits of both formalisms, enabling the modeling and reasoning of computationally hard problems under uncertainty? What are the computational complexity consequences of such a combination? And is it possible to design and implement a practical system that effectively supports the resulting formalism?*

In this paper, we answer these questions by proposing ASP(Q)^{Inf}, an extension of ASP(Q) in which the innermost quantified program is a probabilistic logic program. This design naturally enriches ASP(Q) with the ability to reason about uncertainty, allowing one to verify whether given properties hold with at least a specified probability. We show that this extended formalism is not only theoretically well-founded, but also practically relevant, as it enables intuitive modeling of a variety of AI problems, including network resilience analysis (O'Neil et al. 2025), viral marketing (Domingos and Richardson 2001; Nath and Domingos 2009; Van den Broeck et al. 2010), and probabilistic conformant planning (Kushmerick, Hanks, and Weld 1995; Zhang, Grastien, and Gretton 2024). Moreover, we provide a detailed complexity analysis of its main reasoning task, precisely characterizing its position within the Counting Polynomial-time Hierarchy (Wagner 1986). Moreover, we present a first prototype implementation based on Algebraic Model Counting (AMC) (Eiter, Hecher, and Kiesel 2024) for programs with two quantifiers, demonstrating that the proposed formalism can be implemented using existing model counting techniques. An experimental analysis confirms the viability of the approach.

2 Background

We now provide concepts related to Answer Set Programming, Probabilistic Logic Programming, and Algebraic Model Counting.

2.1 Answer Set Programming

In ASP *variables* are alphanumeric strings starting with an uppercase letter, whereas *constants* are either numbers or alphanumeric strings starting with a lowercase letter. A *term* is either a variable or a constant. An *atom* is an expression of the form $p(t_1, \ldots, t_n)$ where p is a *predicate* of arity n and $t_1, \ldots, t_n$ are terms. A *literal* is either an atom a or its negation $not\ a$, where not stands for *negation as failure*. A literal is said to be *negative* if it is of the form $not\ a$; otherwise, it is *positive*. A *rule* is an expression of the form $h \leftarrow l_1, \ldots, l_k$, where h is an atom, referred to as *head*, which can also be omitted, and $l_1, \ldots, l_k$, with $k \geq 0$, is a conjunction of literals referred to as *body*. A rule is said to be a *constraint* if its head is omitted; whereas it is said to be a *fact* if $k = 0$. A rule r is said to be *safe* if each variable appearing in r, also appears in some positive literal of the body of r. A *program* is a finite set of safe rules. An ASP expression ϵ (i.e., atom, literals, etc.) is said to be *ground* if ϵ does not contain any variable. Given a program P, the *Herbrand Universe*, denoted by U_P, is the set of constants appearing in P; whereas the *Herbrand Base*, denoted by B_P, is the set of ground atoms obtained from predicates in P and constants in U_P. Given a rule $r \in P$, we denote by $ground(r)$ the set of all possible instantiations of r obtained by substituting variables with constants appearing in U_P. The concept of instantiation also extends to programs, and so $ground(P)$ denotes the union of $ground(r)$, for each $r \in P$. The *dependency graph* of P, denoted by G_P, is a labeled directed graph whose nodes are atoms in B_P, and there exists a positive (resp. negative) edge from u to v if there exists $r \in ground(P)$ such that u appears in some positive (resp. negative) literal in the body of r and v appear in the head of r. The program P is said to be *stratified* if G_P does not contain any loop involving negative edges. An *interpretation* $I \subseteq B_P$ is a set of ground atoms. A positive (resp. negative) literal $l = a$ (resp. $l = not\ a$) is true w.r.t. I, denoted by $I \models l$, if $a \in I$ (resp. $a \notin I$) otherwise it is false. A conjunction of literals $l_1, \ldots, l_k$, with $k \geq 0$, is true w.r.t. I, denoted by $I \models l_1, \ldots, l_k$, if $I \models l_i$ for each $i \in \{1, \ldots, k\}$. A rule $r \in ground(P)$ is said to be *satisfied* w.r.t. I, denoted by $I \models r$, if the head of r is true w.r.t. I whenever the body is true w.r.t. I. An interpretation I is a *model* of P if all the rules in $ground(P)$ are satisfied w.r.t. I. Let M be a model of P, then the GL-reduct (Gelfond and Lifschitz 1991a), denoted by P^M, is the program obtained from P by: (i) removing all rules having at least one negative literal false w.r.t. M in the body; and (ii) all negative literals from the body of the remaining rules. M is said to be an answer set of P if M is a $\subset$-minimal model of P^M (no model $M' \subset M$ of P^M exists). In the following, we will use the acronym ASP to denote both the formalism and an answer set program.

2.2 Probabilistic Logic Programming

Syntax. Probabilistic Logic Programming (PLP) (Riguzzi 2022) extends logic programs with primitives to represent uncertainty. Among the possible formalisms, there are ProbLog probabilistic facts (De Raedt, Kimmig, and Toivonen 2007) of the form $\pi :: a$ where $\pi \in [0, 1]$ is the probability associated with the atom a. A PLP (we use the same acronym to denote both the formalism and a program in such a formalism), is composed by a stratified program and a set of probabilistic facts. We also consider Annotated Disjunctions (ADs) (Vennekens, Verbaeten, and Bruynooghe 2004) which are of the form $\pi_1 :: a_1; \pi_2 :: a_2; \ldots; \pi_n :: a_n :- B$ where π_i is the probability value associated with the atom a_i and B is a conjunction of literals. Intuitively, this means that if B is true, then a_1 is true with probability π_1 or a_2 is true with probability π_2, and so on. Here, we assume that the π_is sum to 1. Note that the original proposal of annotated disjunctions had the probability after the atom, but we consider an alternative notation here for uniformity with ProbLog probabilistic facts. An AD rule can be converted into a set of ProbLog probabilistic facts (Riguzzi 2022) so in the following we limit the treatment to probabilistic facts.

Semantics. A *world* is a stratified logic program composed of the rules of the PLP and an atom a_i for each $\pi_i :: a_i$ considered true. A PLP has 2^n worlds and the Distribution Semantics (Sato 1995) requires that every world has exactly one well-founded (Van Gelder, Ross, and Schlipf 1991) model (which also coincides with the unique stable model (Gelfond and Lifschitz 1991b)). Probabilistic facts are considered independent. The probability of a world w, $P(w)$, is computed as $P(w) = \prod_{a_i \in w} \pi_i \prod_{a_j \notin w} (1 - \pi_i)$ where with $a_i \in w$ (resp. $a_i \notin w$) we denote a probabilistic fact considered as true (resp. false) in the world w. The probability of a ground atom q called query is computed as $P(q) = \sum_{w \models q} P(w)$. We also use the notation $P(q \mid C)$ to denote the probability of the query q computed while considering the PLP C.

Example 1. *The following PLP has 3 probabilistic facts.*

```
0.2::a. 0.3::b. 0.4::c.
q:- a.
q:- b,c.
```

Among the 8 possible worlds, the one where a is true and both b and c are false has probability $0.2 \cdot (1 - 0.3) \cdot (1 - 0.4) = 0.084$. *If we are interested in computing the probability of q, we get* $P(q) = 0.296$.

2.3 Algebraic Model Counting

Algebraic Model Counting (AMC) (Kimmig, Van den Broeck, and De Raedt 2017) is a generic framework to encode tasks in Statistical Relational Artificial Intelligence.

Definition 1. *Given a propositional theory $\mathcal{T}$ with variables V, a commutative semiring $\mathcal{R} = (D, \oplus, \otimes, n_\oplus, n_\otimes)$, and a weight function w mapping the literals associated with variables in V to the elements of D, consider the tuple*

$A = (\mathcal{T}, \mathcal{R}, w)$. *AMC requires computing*

$$AMC(A) = \bigoplus_{I \in \varphi(\mathcal{T})} \bigotimes_{l \in I} w(l)$$

where $\varphi(\mathcal{T})$ *is the set of models of* $\mathcal{T}$.

For example, the probability semiring $P = ([0,1], +, \cdot, 0, 1)$ can be adopted to perform inference in PLPs, by considering as weight function $w(l)$ with $w(l) = \pi$ if $l = a$ with probabilistic fact $\pi :: a$; or $w(l) = (1 - \pi)$ if $l = not\ a$ with probabilistic fact $\pi :: a$; $w(l) = 1$ otherwise.

Recently, AMC was extended to Second Level AMC (Kiesel, Totis, and Kimmig 2022). Intuitively, this allows nesting two levels of AMC and connect them through a transformation function which converts the result of the inner AMC into a value for the outer AMC.

Definition 2. *Given a propositional theory* $\mathcal{T}$ *whose variables are partitioned into* (V_i, V_o)*, two commutative semirings* $\mathcal{R}_i = (D^i, \oplus^i, \otimes^i, n^i_\oplus, n^i_\otimes)$ *and* $\mathcal{R}_o = (D^o, \oplus^o, \otimes^o, n^o_\oplus, n^o_\otimes)$*, two weight functions* w_i *and* w_o*, and a transformation function* $f_{io} : R^i \to R^o$*, let* $A = (\mathcal{T}, V_i, V_o, \mathcal{R}_i, \mathcal{R}_o, w_i, w_o, f_{io})$*. 2AMC is defined as:*

$$2AMC(A) = \bigoplus^{o}_{I_o \in \mu(V_o)} \bigotimes^{o}_{a \in I_o} w_o(a) \otimes^o f_{io}\Big(\bigoplus^{i}_{I_i \in \varphi(\mathcal{T} | I_o)} \bigotimes^{i}_{b \in I_i} w_i(b)\Big) \quad (1)$$

where $\mu(V_o)$ *is the set of possible assignments to* V_o *and* $\varphi(\mathcal{T} \mid I_o)$ *is the set of assignments* I_i *of* V_i *such that* (I_i, I_o) *satisfies* $\mathcal{T}$.

3 ASP(Q) with Probabilistic Inference

A recent extension of ASP, namely ASP with Quantifiers (ASP(Q), and we use the same symbol to denote a program in this formalism) (Amendola, Ricca, and Truszczynski 2019; Mazzotta, Ricca, and Truszczynski 2024), has introduced the concept of quantification over answer sets of ASP which increases the expressive power of ASP to the entire Polynomial-time Hierarchy (Stockmeyer 1976). Despite its effectiveness, ASP(Q) cannot handle probabilistic reasoning. Thus, to close this gap, we propose an extension of ASP(Q) called ASP(Q)^{Inf}.

Definition 3 (ASP(Q)^{Inf} syntax)**.** *An ASP(Q) with probabilistic inference, ASP(Q)*^{Inf}*, is an expression of the form:*

$$\Box_1 P_1 \cdots \Box_n P_n : (C, q, k) \quad (2)$$

where q *is an atom called query,* k *is a threshold value* $\in [0,1]$*,* C *is a PLP, and, for each* $i \in \{1, \ldots, n\}$*,* $\Box_i \in \{\exists^{st}, \forall^{st}\}$ *and* P_i *is an ASP.*

W.l.o.g, we assume that atoms appearing in some rule head of a subprogram P_i, with $i \in \{1, \ldots, n+1\}$, $P_{n+1} = C$, cannot appear in any P_j, with $j < i$. Furthermore, for a PLP C and an answer set M, $C^{\downarrow M} = C \cup \{a \leftarrow \mid a \in M\}$.

Definition 4 (ASP(Q)^{Inf} semantics)**.** *The semantics of an ASP(Q)*^{Inf} *(i.e., the coherence) is inductively defined as:*

- $\exists^{st} P : (C, q, k)$ *is coherent if there exists* $M \in AS(P)$ *such that* $P(q \mid C^{\downarrow M}) \geq k$*;*
- $\forall^{st} P : (C, q, k)$ *is coherent if for each* $M \in AS(P)$*,* $P(q \mid C^{\downarrow M}) \geq k$*;*
- $\exists^{st} P \mathcal{A}(q, k)$ *is coherent if there exists* $M \in AS(P)$ *such that* $\mathcal{A}_{P,M}(q, k)$ *is coherent;*
- $\forall^{st} P \mathcal{A}(q, k)$ *is coherent if for each* $M \in AS(P)$*,* $\mathcal{A}_{P,M}(q, k)$ *is coherent;*

where $\mathcal{A}(q, k)$ *is an ASP(Q)*^{Inf} *of the form (2) and* $\mathcal{A}_{P,M}(q, k)$ *denotes the ASP(Q)*^{Inf} *obtained from* $\mathcal{A}(q, k)$ *by replacing* P_1 *with* $P_1 \cup fix_P(M)$*, where* $fix_P(M) = \{a \leftarrow \mid a \in M\} \cup \{\leftarrow a \mid a \in at(P) \setminus M\}$*. ASP(Q)*^{Inf} *is existential if* $\Box_1 = \exists^{st}$*; universal otherwise. Let* $\exists^{st} P \mathcal{A}(q, k)$ *be an existential ASP(Q)*^{Inf} *then* $M \in AS(P)$ *is a* quantified answer set *of* $\exists^{st} P \mathcal{A}(q, k)$ *if and only if* $\mathcal{A}_{P,M}(q, k)$ *is coherent.*

We are now ready to define the main reasoning tasks concerning ASP(Q)^{Inf}.

Definition 5 (Existential ASP(Q)^{Inf} coherence problem)**.** *Let* $\exists^{st} P\ \mathcal{A}(q, k)$ *be an existential ASP(Q)*^{Inf}*. The existential ASP(Q)*^{Inf} *coherence problem asks whether there exists at least one quantified answer set of* $\exists^{st} P\ \mathcal{A}(q, k)$.

Definition 6 (Existential ASP(Q)^{Inf} enumeration problem)**.** *Let* $\exists^{st} P\ \mathcal{A}(q, k)$ *be an existential ASP(Q)*^{Inf}*. The existential ASP(Q)*^{Inf} *enumeration problem requires computing the set of quantified answer sets of* $\exists^{st} P\ \mathcal{A}(q, k)$.

Observation 1. *The existential ASP(Q)*^{Inf} *enumeration problem (Definition 6) generalizes the existential ASP(Q)*^{Inf} *coherence problem (Definition 5), as the former requires computing* all *quantified answer sets while for the latter it is sufficient to compute* one.

Definition 7 (Universal ASP(Q)^{Inf} problem)**.** *Let* $\forall^{st} P\ \mathcal{A}(q, k)$ *be a universal ASP(Q)*^{Inf}*. The universal ASP(Q)*^{Inf} *problem requires asking whether* $\forall^{st} P\ \mathcal{A}(q, k)$ *is coherent.*

Observation 2. *In universal ASP(Q)*^{Inf} *there are no quantified answer sets, as* $\forall$ *either holds or does not hold.*

Observation 3. *The result of the existential ASP(Q)*^{Inf} *enumeration problem (definition 6) is a set of sets of atoms (i.e., a set of quantified answer sets) while the result of the universal ASP(Q)*^{Inf} *problem is a Boolean value (true or false).*

The following example clarifies ASP(Q)^{Inf}.

Example 2. *Let* $\mathcal{A}$ *be the ASP(Q)*^{Inf}*:*

```
% exists % Program P1
a :- not na.
na :- not a.
% forall % Program P2
b :- a, not nb.
nb :- a, not b.
c :- not a, not nc.
nc :- not a, not c.
% inference % Program C
0.3::d.
0.6::f.
q :- c, d.
```

```
q :- b, f.
q :- not a, not c, f.
q :- a, not b, not d.
% query: q
% threshold: 0.6
```

P_1 has two answer sets: $M_1 = \{na\}$ and $M_1' = \{a\}$. Let us first consider M_1 and fix it in the program. Now, $P_2' = P_2 \cup fix_{P_1}(M_1)$ has two answer sets: $M_2 = \{na, nc\}$ and $M_2' = \{na, c\}$. M_1 is a quantified answer set if $P(q \mid C^{\downarrow M_2}) \geq 0.6$ and $P(q \mid C^{\downarrow M_2'}) \geq 0.6$.

$M \in AS(P_2')$	$\sigma \subseteq \mathcal{F}(C)$	$AS(C_\sigma^{\downarrow M})$	$P(C_\sigma^{\downarrow M})$	$P(q \mid C^{\downarrow M})$
$\{na, nc\}$	$\{\}$	$\{na, nc\}$	*0.28*	*0.6*
	$\{d\}$	$\{na, nc, d\}$	*0.12*	
	$\{f\}$	$\{na, nc, f, q\}$	***0.42***	
	$\{d, f\}$	$\{na, nc, d, f, q\}$	***0.18***	
$\{na, c\}$	$\{\}$	$\{na, c\}$	*0.28*	*0.3*
	$\{d\}$	$\{na, c, d, q\}$	***0.12***	
	$\{f\}$	$\{na, c, f\}$	*0.42*	
	$\{d, f\}$	$\{na, c, d, f, q\}$	***0.18***	

Since $P(q \mid C^{\downarrow M_2'}) = 0.3 < 0.6$, then $M_1 = \{na\}$ is not a quantified answer set. Let us now consider $M_1' = \{a\}$ and $P_2'' = P_2 \cup fix_{P_1}(M_1')$. We obtain:

$M \in AS(P_2'')$	$\sigma \subseteq \mathcal{F}(C)$	$AS(C_\sigma^{\downarrow M})$	$P(C_\sigma^{\downarrow M})$	$P(q \mid C^{\downarrow M})$
$\{a, nb\}$	$\{\}$	$\{a, nb, q\}$	***0.28***	*0.7*
	$\{d\}$	$\{a, nb, d\}$	*0.12*	
	$\{f\}$	$\{a, nb, f, q\}$	***0.42***	
	$\{d, f\}$	$\{a, nb, d, f\}$	*0.18*	
$\{a, b\}$	$\{\}$	$\{a, b\}$	*0.28*	*0.6*
	$\{d\}$	$\{a, b, d\}$	*0.12*	
	$\{f\}$	$\{a, b, f, q\}$	***0.42***	
	$\{d, f\}$	$\{a, b, d, f, q\}$	***0.18***	

Here, for each $M \in AS(P_2'')$, $P(q \mid C^{\downarrow M}) \geq 0.6$. Thus, $M_1' = \{a\}$ is a quantified answer set.

4 Modeling Examples

In what follows, we showcase the modeling capabilities of ASP(Q)^{Inf} through a series of problems of practical interest.

Network Resilience. The network resilience problem is a well-studied task (O'Neil et al. 2025) which, at a high level, requires assessing the resilience of an infrastructure to possible disruption scenarios. The goal is to ensure that the probability of reaching a given destination, starting from a given source, is above a certain threshold. To represent this, we first create an ASP P whose answer set corresponds to possible network models.

```
{model1}. {model2}.
:- model1, model2.
```

We define a PLP C where probabilistic facts denote edges where the probability represents, for example, the reliability of the links. Then, the underlying logic program checks the reachability according to the specific network model.

```
0.2::edge(a,b). 0.3::edge(b,d).
0.4::edge(a,c). 0.4::edge(c,d).
0.2::edge_(b,f). 0.2::edge_(f,d).
0.7::edge_(c,e). 0.6::edge_(e,d).
edge(f,d):- model1, edge_(f,d).
edge(b,f):- model1, edge_(b,f).
edge(c,e):- model2, edge_(c,e).
edge(e,d):- model2, edge_(e,d).
path(A,B):- edge(A,B).
path(A,B):- edge(A,C), path(C,B).
```

We can now build an ASP(Q)^{Inf} of the form $\exists^{st} P : (C, path(a, d), k)$ that is coherent if and only if there exists a network model that provides a probability of reaching d from a greater than or equal to k.

Now, we add another layer: suppose that there are two possible network failures, which are unlikely to happen together, that influence the existence of two edges. Assume now F to be the following ASP

```
{failure1}. {failure2}.
:- failure1, failure2.
```

C is augmented with the following rules

```
0.3::_edge(b,d).
0.4::_edge(c,d).
edge(b,d):- not failure1, edge_(b,d).
edge(c,d):- not failure2, edge_(c,d).
```

We can now ask whether there exists one network model such that, for all possible failures, the probability of reaching d from a is $\geq k$. This problem can be modeled as an ASP(Q)^{Inf} of the form $\exists^{st} P \forall^{st} F : (C, path(a, d), k)$.

Viral Marketing The viral marketing problem (Domingos and Richardson 2001; Nath and Domingos 2009; Van den Broeck et al. 2010) involves a network of people which can be targeted by ads. The purchasing behavior of people is influenced by their friendship relationship and the effectiveness of the advertising campaign. The following ASP(Q)^{Inf} describes a (small) viral marketing scenario where there are two possible campaign strategies (*cmpgn1* and *cmpgn2*) with three people involved (a, b, and c). These people know each other with a certain probability (*friend*$/2$). Furthermore, an ad may or may not be effective (*effect*$/1$). Different campaign strategies target different people (*target*$/1$). Lastly, a person buys (*buys*$/1$) a product if she/he is targeted or some of his/her friends buy it. We may be interested in computing whether there is a campaign strategy where all the people buy the product (*all_buy*) or at least one buys (*at_least_one_buys*) with a probability greater than or equal to a certain value.

```
%@exists
{cmpgn1}. {cmpgn2}.
%@inference
0.1::effect(a). 0.2::effect(b).
0.3::effect(c).
0.2::friend(a,b). 0.3::friend(a,c).
0.4::friend(b,c).
target(a):- effect(a), cmpgn1.
target(b):- effect(b), cmpgn2.
target(c):- effect(c), cmpgn1, not cmpgn2.
buys(P):- target(P).
buys(P):- friend(P,X), buys(X).
```

```
all_buy:- buys(a), buys(b), buys(c).
at_least_one_buys:- buys(P).
```

Probabilistic Planning Automated planning is a fundamental component in the design of autonomous intelligent systems (Ghallab, Nau, and Traverso 2004). It concerns the problem of computing a course of actions, called a plan, that transforms the state of the world from a given initial configuration to a desired goal configuration. A typical planning problem consists of a domain representation, an initial state specification, and a goal description. The objective is to determine the existence of a valid plan, i.e., a plan that allows to reach the final goal.

In classical planning, the initial state is assumed to be fully specified, the action domain is deterministic, and a plan is defined as a finite sequence of actions to be executed. However, these assumptions can be relaxed to capture richer and more realistic classes of planning problems (Turner 2002). In particular, probabilistic conformant planning (Kushmerick, Hanks, and Weld 1995; Zhang, Grastien, and Gretton 2024) considers a planning problem under uncertainty by assuming a probability distribution over the possible initial states and seeking plans that achieve the goal with a guaranteed probability. This probabilistic setting naturally fits the ASP(Q)^{Inf} framework.

The following ASP(Q)^{Inf} encodes a probabilistic conformant planning problem in which a robot moves blindly on a 2x2 grid and has to reach a goal cell. At each step, the robot can move in the four possible directions (i.e., up, right, down, or left). The initial position is not known and it is described by a probability distribution over possible value grid's cells.

```
%@exists
len(3). step(1). step(2). step(3)
% activity
act(up). act(right).
act(down). act(left).
% guess a plan
exec(A,I) :- step(I), act(A), not skip(A,I).
skip(A,I) :- step(I), act(A), not exec(A,I).
% exactly one activity for each step
hasAct(I):-exec(A,I).
:-step(I),not hasAct(I).
:-exec(A1,I), exec(A2,I), A1!=A2.
%@inference
p0::init(x,0);p1::init(x,1);p2::init(x,2).
p3::init(y,0);p4::init(y,1);p5::init(y,2).
pos(0..2). coord(x). coord(y).
%conditional-effect
move(right,x,1). move(right,y,0).
move(left,x,-1). move(left,y,0).
move(up,x,0). move(up,y,-1).
move(down,x,0). move(down,y,1).
%goal-state
goal(x,1,true). goal(y,1,true).
%pre-condition
pre(up,y,0,false).
pre(right,x,2,false).
pre(down,y,2,false).
pre(left,x,0,false).
```

```
state(1,C,V,true) :- init(C,V).
state(1,C,V,false) :- coord(C),pos(V),
   not init(C,V).
state(T+1,C,V+D,true):- exec(A,T),
   move(A,C,D), state(T,C,V,true).
state(T+1,C,V1,false):- exec(A,T),
   move(A,C,D), state(T,C,V,true),
   pos(V1), V1!=V+D.
invalid :- exec(A,T), state(T,C,F,V1),
   pre(A,C,F,V2), V1!=V2.
invalid :- len(K),state(K+1,C,F,V1),
   goal(C,F,V2), V1!=V2.
q :- not invalid.
```

In this ASP(Q)^{Inf}, the answer sets of the first program are in one-to-one correspondence with plans of length 3. For each step $i \in \{1, 2, 3\}$, the program guesses exactly one action to be executed. For example, a possible plan consists of moving *up* in step 1, *right* in step 2, and *down* in step 3. Then, in the PLP, worlds model possible initial states and the corresponding state transformation according to the plan computed by the previous quantifier. More precisely, a world entails the query q if and only if the plan is executable (i.e., no actions' preconditions are violated at each step) and the final state, reached by executing the plan, corresponds to the goal state (i.e., the robot is in the goal cell at the last step). Thus, the probability of the query q is obtained by summing the probabilities of all worlds that entail q, which consists of the probability that the plan reaches the goal state. As a result, given a desired probability threshold k, the ASP(Q)^{Inf} program is coherent if and only if there exists a plan whose probability of reaching the goal state is $\geq k$.

5 Computational Complexity Study

Here, we recall the definition of Counting Polynomial-time Hierarchy and study the complexity of ASP(Q)^{Inf}.

Counting Hierarchy. The Counting Polynomial-time Hierarchy (Wagner 1986; Stockmeyer 1983) extends the Polynomial-time Hierarchy (Stockmeyer 1976) by introducing the notion of counting quantifiers. Given a formula $H(x, y)$, with free variables x and y, the counting quantifier $\mathbf{C}$ is defined as:

$$\overset{k}{\underset{y}{\mathbf{C}}}\ H(x,y) \leftrightarrow |\{y \mid H(x,y) \textit{ is true}\}| \geq k.$$

Together with existential and universal quantifiers, denoted as V and $\mathsf{\Lambda}$ respectively, it is possible to define the Counting Polynomial-time Hierarchy. For a class of languages $\mathcal{K}$:

$A \in \mathsf{V}\,\mathcal{K}$ if and only if there exists $B \in \mathcal{K}$ and a polynomial p such that

$$x \in A \leftrightarrow \bigvee_{|y| \leq p(|x|)} (x,y) \in B$$

$A \in \mathsf{\Lambda}\,\mathcal{K}$ if and only if there exists $B \in \mathcal{K}$ and a polynomial p such that

$$x \in A \leftrightarrow \bigwedge_{|y| \leq p(|x|)} (x,y) \in B$$

$A \in \mathbf{C}\,\mathcal{K}$ if and only if there exists $B \in \mathcal{K}$, a polynomial-time computable function f, and a polynomial p such that

$$x \in A \leftrightarrow \overset{f(x)}{\underset{|y|\leq p(|x|)}{\mathbf{C}}} (x, y) \in B$$

Interestingly, the counting quantifier generalizes existential and universal ones as:

$$\bigvee_{|y|\leq p(|x|)} (x, y) \in B \leftrightarrow \overset{1}{\underset{|y|\leq p(|x|)}{\mathbf{C}}} (x, y) \in B$$

and

$$\bigwedge_{|y|\leq p(|x|)} (x, y) \in B \leftrightarrow \overset{N(x)}{\underset{|y|\leq p(|x|)}{\mathbf{C}}} (x, y) \in B$$

where $N(x)$ is the number of y with $|y| \leq p(|x|)$. We can now recall the Counting Polynomial-time Hierarchy.

Definition 8 (Wagner 1986)**.** *Let $\mathscr{P}$ be the class of problems solvable in polynomial time, then the Counting Polynomial-time Hierarchy $\mathscr{CH}$ is the smallest family $\mathcal{F}$ of classes of languages satisfying:*

1. $\mathscr{P} \in \mathcal{F}$
2. *if $\mathcal{K} \in \mathcal{F}$ then $\bigvee \mathcal{K}$, $\bigwedge \mathcal{K}$, and $\mathbf{C}\,\mathcal{K} \in \mathcal{F}$*

The complexity classes in $\mathscr{CH}$ capture many problems that require counting, such as inference in probabilistic logic programs (Mauá and Cozman 2020). For example, deciding whether the probability of a given query q in a PLP is greater than or equal to k is complete for the PP class (Mauá and Cozman 2020). Recall that the class PP is defined as the family of problems that can be decided by a probabilistic Turing machine (Gill 1977), and it coincides with the class $\mathbf{C}\mathscr{P} \in \mathscr{CH}$ (Torán 1991). Thus, the inference problem in PLP falls into $\mathscr{CH}$ and is complete for $\mathbf{C}\mathscr{P}$. Other than probabilistic reasoning tasks, many problems fall in classes of $\mathscr{CH}$ (Wagner 1986).

Definition 9 (Wagner 1986)**.** *Let $k \geq 1$, $Q_1 \dots Q_{k-1} \in \{\bigvee, \bigwedge, \mathbf{C}\}$, and $Q_k \in \{\bigvee, \mathbf{C}\}$. Then $(F(x_1, \dots, x_k), m_1, \dots, m_k) \in Q_1 \dots Q_k\ B_{be}$ if and only if $F(x_1, \dots, x_k)$ is a boolean formula in conjunctive normal form, $m_1, \dots, m_k \in \mathbb{N}$, and $\overset{m_1}{\underset{\alpha_1}{Q_1}} \dots \overset{m_k}{\underset{\alpha_k}{Q_k}}\ F(\alpha_1, \dots, \alpha_k) = 1$.*

Essentially, the problem of Definition 9 is an extension of Quantified Boolean Formulae (QBF) satisfiability which also considers counting quantifiers. As for QBF, the complexity is guided by quantifier alternation and so, also in this case, completeness for the class $Q_1 \dots Q_k\ \mathscr{P}$ has been proved by Wagner (1986).

Theorem 1 (Wagner 1986)**.** *For every $k \geq 1$ and $Q_1, \dots, Q_k \in \{\bigvee, \bigwedge, \mathbf{C}\}$, $Q_1 \dots Q_k\ B_{be}$ is complete for $Q_1 \dots Q_k\ \mathscr{P}$.*

Complexity Results. To study the complexity of deciding the coherence of ASP(Q)^{Inf} we define:

- $\Sigma_0^{\mathbf{C}} = \Pi_0^{\mathbf{C}} = \mathbf{C}\mathscr{P}$
- $\Sigma_{n+1}^{\mathbf{C}} = \bigvee \Pi_n^{\mathbf{C}}$, with $n \geq 0$
- $\Pi_{n+1}^{\mathbf{C}} = \bigwedge \Sigma_n^{\mathbf{C}}$, with $n \geq 0$

Intuitively, $\Sigma_n^{\mathbf{C}}$ (resp. $\Pi_n^{\mathbf{C}}$) denotes the class $\bigvee \bigwedge \dots Q\ \mathbf{C}\ \mathscr{P}$ (resp. $\bigwedge \bigvee \dots Q\ \mathbf{C}\ \mathscr{P}$) in which the existential and universal quantifiers alternate, except for the last one which is a counting quantifier. This quantifier alternation naturally captures the complexity of deciding the coherence of ASP(Q)^{Inf}.

Theorem 2 (Membership)**.** *Let $\mathcal{A}$ be a ASP(Q)^{Inf} of the form $\Box_1 P_1 \dots \Box_n P_n : (C, q, k)$; then deciding coherence of $\mathcal{A}$ is*

- *in $\Sigma_n^{\mathbf{C}}$ if $\Box_1 = \exists^{st}$*
- *in $\Pi_n^{\mathbf{C}}$ if $\Box_1 = \forall^{st}$*

Proof. Let us first consider the base case with $n = 1$. Here, we have two possible scenarios either: i) $\mathcal{A} = \exists^{st} P : (C, q, k)$ or ii) $\mathcal{A} = \forall^{st} P : (C, q, k)$.

i) In this case, we prove that deciding the coherence of $\mathcal{A}$ is $\Sigma_1^{\mathbf{C}} = \bigvee \mathbf{C}\mathscr{P}$. Here $\mathcal{A}$ is coherent if and only if there exists $M \in AS(P)$ such that $P(q \mid C') \geq k$, with $C' = C \cup \{a :- \mid a \in M\}$. Let $M \subseteq B_P$ be an interpretation. Verifying that $M \in AS(P)$ is in $\mathscr{P}$ (Dantsin et al. 2001) and verifying that $P(q \mid C') \geq k$ is in PP (Mauá and Cozman 2020). This means that the language B which accepts (P, M) if and only if M is an answer set of P and $P(q \mid C') \geq k$ is in $\mathbf{C}\mathscr{P}$. Thus, $\mathcal{A}$ is coherent if and only if

$$\bigvee_{M\subseteq B_P} (P, M) \in B$$

and so deciding coherence of $\mathcal{A}$ is in $\bigvee \mathbf{C}\mathscr{P}$.

ii) In this case, we prove that deciding the coherence of $\mathcal{A}$ is $\Pi_1^{\mathbf{C}} = \bigwedge \mathbf{C}\mathscr{P}$. Here $\mathcal{A}$ is coherent iff for each $M \in AS(P)$, $P(q \mid C') \geq k$, with $C' = C \cup \{a :- \mid a \in M\}$.

As in the previous case, let B be the language which accepts (P, M), with $M \subseteq B_P$ being an interpretation, iff and only if M is not an answer set of P or $P(q \mid C') \geq k$. Then B is in $\mathbf{C}\mathscr{P}$. Thus, $\mathcal{A}$ is coherent if and only if

$$\bigwedge_{M\subseteq B_P} (P, M) \in B$$

and so deciding coherence of $\mathcal{A}$ is in $\bigwedge \mathbf{C}\mathscr{P}$.

By induction, let now assume that deciding the coherence of ASP(Q)^{Inf} of the form $\Box_1 P_1 \dots \Box_n P_n : (C, q, k)$ is in $\Sigma_n^{\mathbf{C}}$ if $\Box_1 = \exists^{st}$; otherwise is in $\Pi_n^{\mathbf{C}}$. Then, we prove that deciding the coherence of $\mathcal{A} = \Box P\, \Box_1 P_1 \dots \Box_n P_n : (C, q, k)$ is in $\Sigma_{n+1}^{\mathbf{C}}$ if $\Box = \exists^{st}$; otherwise is in $\Pi_{n+1}^{\mathbf{C}}$.

($\Box = \exists^{st}$) In this case, $\mathcal{A}$ is coherent if and only if there exists $M \in AS(P)$ such that $\mathcal{A}' = \forall^{st} P_1 \cup fix_P(M) \dots \Box_n P_n : (C, q, k)$ is coherent. Let $M \subseteq B_P$ be an interpretation. Then, verifying that M is an answer set of P is in $\mathscr{P}$ and, from the inductive hypothesis, verifying the coherence of $\mathcal{A}'$ is in $\Pi_n^{\mathbf{C}}$. Thus, the language B that accepts $(\mathcal{A}, M)$ if and only if M is an answer set of P and $\mathcal{A}'$ is coherent is in $\Pi_n^{\mathbf{C}}$. So, $\mathcal{A}$ is coherent if and only if

$$\bigvee_{M\subseteq B_P} (\mathcal{A}, M) \in B$$

and so deciding the coherence of $\mathcal{A}$ is in $\bigvee \Pi_n^{\mathbf{C}} = \Sigma_{n+1}^{\mathbf{C}}$.

($\square = \forall^{st}$) In this case, $\mathcal{A}$ is coherent if and only if for each $M \in AS(P)$, $\mathcal{A}' = \exists^{st} P_1 \cup fix_P(M) \dots \square_n P_n : (C, q, k)$ is coherent. As in the previous case, verifying that an interpretation $M \subseteq B_P$ is an answer set of P is in $\mathscr{P}$ and, from the inductive hypothesis, verifying the coherence of $\mathcal{A}'$ is in $\Sigma_n^{\mathbf{C}}$. Thus, the language B that accepts $(\mathcal{A}, M)$ if and only if M is not an answer set of P or $\mathcal{A}'$ is coherent is in $\Sigma_n^{\mathbf{C}}$. This means that $\mathcal{A}$ is coherent if and only if

$$\bigwedge_{M \subseteq B_P} (\mathcal{A}, M) \in B$$

and so deciding the coherence of $\mathcal{A}$ is in $\bigwedge \Sigma_n^{\mathbf{C}} = \Pi_{n+1}^{\mathbf{C}}$.

Thus, the thesis follows. □

Theorem 3 (Hardness)**.** *Let Π be an ASP(Q)^{Inf} of the form $\square_1 P_1 \dots \square_n P_n : (C, q, k)$ then deciding the coherence of Π is hard for $\Sigma_n^{\mathbf{C}}$ if $\square_1 = \exists^{st}$; otherwise it is hard for $\Pi_n^{\mathbf{C}}$.*

Proof. From Definition 9,

$$(F(x_1, \dots, x_n, x_{n+1}), m_1, \dots, m_n, m_{n+1}) \in Q_1 \dots Q_n Q_{n+1} B_{be} \tag{3}$$

if and only if $F(x_1, \dots, x_n, x_{n+1})$ is a boolean formula in conjunctive normal form, $m_1, \dots, m_n, m_{n+1} \in \mathbb{N}$, and $\overset{m_1}{\underset{\alpha_1}{Q_1}} \dots \overset{m_n}{\underset{\alpha_n}{Q_n}} \overset{m_{n+1}}{\underset{\alpha_{n+1}}{Q_{n+1}}} F(\alpha_1, \dots, \alpha_k) = 1$.

Let us consider the case where for each $i \in \{1, \dots, n\}$, $Q_i \in \{\mathsf{V}, \mathsf{\Lambda}\}$ and $Q_i \neq Q_{i+1}$; and $Q_{n+1} = \mathbf{C}$. Then, deciding whether $(F(x_1, \dots, x_n, x_{n+1}), m_1, \dots, m_n, m_{n+1}) \in Q_1 \dots Q_n Q_{n+1} B_{be}$ is $\Sigma_n^{\mathbf{C}}$-complete if $Q_1 = \mathsf{V}$; otherwise it is $\Pi_n^{\mathbf{C}}$-complete.

Such a problem can be encoded as a ASP(Q)^{Inf} of the form $\square_1 P_1 \dots \square_n P_n : (C, q, 0.5 \cdot m_{n+1})$, where for each $i \in \{1, \dots, n\}$:

- $\square_i = \exists^{st}$ if $Q_i = \mathsf{V}$ otherwise $\square_i = \forall^{st}$
- P_i is of the form:

```
alpha(x_i,true) :- not alpha(x_i,false)
alpha(x_i,false) :- not alpha(x_i,true)
```

and C is of the form:

```
0.5::true(x_{n+1})
alpha(x_{n+1},true) :- true(x_{n+1})
alpha(x_{n+1},false) :- not true(x_{n+1})
%for each positive literal x_i in a clause c
sat(c) :- alpha(x_i,true).
%for each negative literal ¬x_i in a clause c
sat(c) :- alpha(x_i,false).
%for each clause c
unsat :- not sat(c).
q :- not unsat.
```

Intuitively, for each $i \in \{1, \dots, n\}$, the answer sets of the program P_i are in one-to-one correspondence with the possible truth assignments α_i of the variable x_i, as the possible answer sets are $\{alpha(x_i, true)\}$ and $\{alpha(x_i, false)\}$. Moreover, each existential (resp. universal) quantifier V_i (resp. $\mathsf{\Lambda}_i$) corresponds to $\square_i = \exists^{st}$ (resp. $\square_i = \forall^{st}$). Thus, the quantifiers $\square_1 \dots \square_n$ iterate exactly over the all possible sequences of truth assignment $\alpha_1, \dots, \alpha_n$ for variables $x_1, \dots, x_n$. Let us now focus on the last quantifier. Given a sequence of truth assignment $\alpha_1, \dots, \alpha_n$ and the corresponding sequence of answer sets $M_1 \subseteq \dots \subseteq M_n$, then

$$\overset{m_{n+1}}{\underset{\alpha_{n+1}}{\mathbf{C}}} F(\alpha_1, \dots, \alpha_n, \alpha_{n+1}) = 1$$

if and only if there are at least m_{n+1} truth assignments α_{n+1} for x_{n+1} such that $F(\alpha_1, \dots, \alpha_n, \alpha_{n+1})$ evaluates true. Here we observe that the worlds of the PLP $C^{\downarrow M_n}$ are in one to one correspondence with the possible truth assignment α_{n+1} of the variable x_n as the probabilistic fact $true(x_{n+1})$ can be either included or not in a selection σ. Thus, each world $C_\sigma^{\downarrow M_n}$ corresponds to a complete sequence of truth assignments $\alpha_1, \dots, \alpha_n, \alpha_{n+1}$ which is encoded with atoms of the form $alpha(x_i, true)$ or $alpha(x_i, false)$ for each $1 \leq i \leq n+1$. Then, the remaining rules in C'_σ encode the evaluation of $F(\alpha_1, \dots, \alpha_n, \alpha_{n+1})$. More precisely, an atom of the form $sat(c)$ is derived if and only c is a clause in $F(x_1, \dots, x_n, x_{n+1})$ and a variable x_i appearing in a positive (resp. negative) literal is assigned to true (resp. to false) w.r.t. to α_i (i.e., the clause c is satisfied). Finally, the query q is derived if and only if all the clauses are satisfied. Thus, the world $C_\sigma^{\downarrow} \models q$ if and only $F(\alpha_1, \dots, \alpha_n, \alpha_{n+1}) = 1$. Since each world has probability 0.5 (as there is only one probabilistic fact) then there are at least m_{n+1} truth assignments α_{n+1} for x_{n+1} such that $F(\alpha_1, \dots, \alpha_n, \alpha_{n+1})$ evaluates true if only if the the probability of the query q is greater than or equal to $0.5 \cdot m_{n+1}$. Thus the thesis follows. □

6 Implementation

We now show how to encode the introduced problems within the AMC framework. We focus on existential ASP(Q)^{Inf} with one level of the form $\exists P : (C, q, k)$ and show how the existential ASP(Q)^{Inf} enumeration problem (Definition 6) can be intuitively encoded as 2AMC. To do so, we need to consider two different semirings, one for the existential level and one for the probabilistic inference level. For probabilistic inference, we use the well-known probability semiring (Kimmig, Van den Broeck, and De Raedt 2017) $\mathcal{R}_{\mathcal{P}} = ([0, 1], +, \cdot, 0, 1)$ while for the existential level, we define a new semiring that we call *existential semiring*.

Definition 10. *Given an existential ASP(Q)^{Inf} of the form $\exists P : (C, q, k)$, the* existential semiring *is defined as*

$$\mathcal{R}_\exists = (\mathcal{D}, \oplus, \otimes, (\emptyset, 0), (\{\emptyset\}, 1))$$

where

- $\mathcal{D} = ((S, N) \mid S \in \mathbb{P}(\mathbb{P}(A)), N \in \{0, 1\})$*, where A is the set of atoms appearing in P and $\mathbb{P}$ denotes the powerset;*
- $(S_0, V_0) \otimes (S_1, V_1) = (\{\Delta_0 \cup \Delta_1 \mid \Delta_0 \in S_0, \Delta_1 \in S_1\}, 1)$ *if $V_0 \neq 0$ and $V_1 \neq 0$; $(\emptyset, 0)$ otherwise*
- $(S_0, V_0) \oplus (S_1, V_1) = (S_0, V_0)$ *if $V_0 \neq 0$ and $V_1 = 0$; (S_1, V_1) if $V_0 = 0$ and $V_1 \neq 0$; $(S_0 \cup S_1, V_0)$ if $V_0 = V_1 \neq 0$; or $(\emptyset, 0)$ if $V_0 = V_1 = 0$*

Definition 11. *Given an existential ASP(Q)^{Inf} of the form $\exists P : (C, q, k)$, the existential ASP(Q)^{Inf} enumeration problem (Definition 6) can be encoded into the 2AMC framework (Definition 2) by considering:*

- $V_o = atoms(P)$ *and* $V_i = H_B \setminus V_o$*, where* $atoms(P)$ *denotes the atoms appearing in* P
- $R_i = \mathcal{R}_\mathcal{P}$ *and* $R_o = \mathcal{R}_\exists$
- $w_i(a_i)$ *returning* p_i *for a probabilistic fact* $p_i :: a_i$ *true,* $1 - p_i$ *for a probabilistic fact* $p_i :: a_i$ *false,* 0 *for not* q*,* 1 *otherwise*
- w_o *returning* $(\{\{a\}\}, 1)$ *for an atom* a *selected;* $(\{\emptyset\}, 1)$ *otherwise*
- $f_{io}(p) = (\{\emptyset\}, 1)$ *if* $p \geq k$*;* $(\emptyset, 0)$ *otherwise*

Observation 4. *We can also encode the existential ASP(Q)^{Inf} coherence problem (Definition 5) using a specialization of the semiring of Definition 11.*

Definition 12. *Given an existential ASP(Q)^{Inf} of the form $\exists P : (C, q, k)$ checking its coherence (Definition 4) can be encoded into the 2AMC framework (Definition 2) by considering:*

- $V_o = atoms(P)$ *and* $V_i = H_B \setminus V_o$
- $R_i = \mathcal{R}_\mathcal{P}$ *and* $R_o = (\mathcal{D}, \oplus, \otimes, (\emptyset, 0), (\emptyset, 1))$ *where*
  - $\mathcal{D} = ((S, N) \mid S \in \mathbb{P}(A), N \in \{0, 1\})$*,* A *is the set of atoms appearing in* P*;*
  - $(S_0, V_0) \oplus (S_1, V_1) = (\min(S_0, S_1), 1)$ *if* $V_0 \neq 0$ *and* $V_1 \neq 0$*,* $(\emptyset, 0)$ *otherwise, where* min *imposes an order on quantified answer sets*
  - $(S_0, V_0) \otimes (S_1, V_1) = (S_0, V_0)$ *if* $V_0 \neq 0$ *and* $V_1 = 0$*;* (S_1, V_1) *if* $V_0 = 0$ *and* $V_1 \neq 0$*;* $(S_0 \cup S_1, V_0)$ *if* $V_0 = V_1 = 1$*; or* $(\emptyset, 0)$ *if* $V_0 = V_1 = 0$
- $w_i(a_i)$ *returning* p_i *for a probabilistic fact* $p_i :: a_i$ *true,* $1 - p_i$ *for a probabilistic fact* $p_i :: a_i$ *false,* 0 *for not* q*,* 1 *otherwise*
- w_o *returning* $(\{a\}, 1)$ *for an atom* a *selected,* $(\emptyset, 1)$ *otherwise*
- $f_{io}(p) = (\emptyset, 1)$ *if* $p \geq k$*;* $(\emptyset, 0)$ *otherwise*

That is, the difference between Definition 11 and Definition 12 is that in the former we compute *all* quantified answer sets while in the latter only one.

Example 3. *Consider the network resilience problem and its encoding (with only one level of quantification) using ASP(Q)^{Inf} of Section 4. For Definition 11 and Definition 12, we have $V_o = \{model1, model2\}$, $w_i(edge(a, b)) = 0.2$, $w_i(not\ edge(a, b)) = 1 - 0.2 = 0.8$ (we use not to explicitly denote the probabilistic fact false), and similarly for the remaining $edge/2$ and $edge\_/2$ probabilistic facts. By following Equation 1, assume that we consider the assignment $I_1 = \{model1\}$. Focus on the inference task (Definition 11). $w_o(model1) = (\{\{model1\}\}, 1)$. Moving to the inner AMC, after we evaluate the remaining PLP (which has 256 worlds) over the query $q = path(a, d)$, we get $P(q) = 0.215104$. If $P(q) \geq k$ (where k is the chosen threshold), the transformation function f_{io} outputs $(\{\emptyset\}, 1)$, otherwise $(\emptyset, 0)$. In the first case, $(\{\{model1\}\}, 1) \otimes (\{\emptyset\}, 1) = (\{\{model1\}\}, 1)$ as for both tuples, the second element is 1, while for the other $(\{\{model1\}\}, 1) \otimes (\emptyset, 0) = (\{\emptyset\}, 0)$. If we repeat this process with $I_2 = \{model2\}$, we have $P(q) = 0.305152$. Assume $k = 0.2$, also here f_{io} outputs $(\{\emptyset\}, 1)$. Then, $(\{\{model2\}\}, 1) \otimes (\{\emptyset\}, 1) = (\{\{model2\}\}, 1)$, $(\{\{model1\}\}, 1) \oplus (\{\{model2\}\}, 1) = (\{\{model1\}, \{model2\}\}, 1)$. The process is analogous for the other cases and for Definition 12.*

7 Experimental Evaluation

We implemented our proposed semirings on top of a modified version of the 2AMC solver aspmc (Eiter, Hecher, and Kiesel 2024), which is a *generic* tool based on knowledge compilation. We denote it as *aspmc*. To the best of our knowledge, no other tool handles the proposed tasks. Nonetheless, to provide a baseline to compare with, we develop an enumeration-based solver that, for each answer set of the program P_1, calls a probabilistic ASP solver (namely, PASTA[1]), to compute the probability of the query. We call this approach *enum*. All experiments were executed on a machine running at 3.7 GHz with a time limit of 1 hour for each instance and a memory limit of 64 GB. Source code and datasets are available here.

Dataset Generation. First, we consider the network resilience problem of Section 4 and generated datasets where the underlying networks follow the Barabási-Albert (BA) (Barabasi and Albert 1999) and Erdős-Renyi (ER) (Erdős and Rényi 1959) models, which are well-known models for real-world networks. BA networks were generated using the NetworkX library (Hagberg, Schult, and Swart 2008) and the function $barabasi\_albert\_graph$ with parameter m (number of edges to attach from a new node to existing nodes) set to 2, while ER networks with $erdos\_renyi\_graph$ with parameter p (probability for edge creation) set to 0.4. We consider *undirected networks* and instances with an increasing number of existential variables in the first level (i.e., $model$) ranging from 4 to 30 with step 2 with a number of nodes ranging from 4 to 20. Then, we declared some facts as probabilistic and a fraction of these is active if a certain set of models is considered. An example of an instance is

```
%exists
{model0}. {model1}.
% program
0.701::p01.
edge(0,1) :- p01, not model1.
0.965::p02.
edge(0,2) :- p02, not model0.
0.984::edge(0,3). 0.147::edge(1,3).
```

For example, in the listing above, $edge(0, 1)$ is considered only if the probabilistic fact $p01$ is true and $model1$ is not selected. We also generated instances encoding the viral marketing task of Section 4 with the same structure as for the network resilience dataset (i.e., the $friendship/2$ facts are generated following the same procedure). Overall, we generated and tested 1008 $(252 \cdot 4)$ instances and, for each

[1] https://github.com/damianoazzolini/pasta

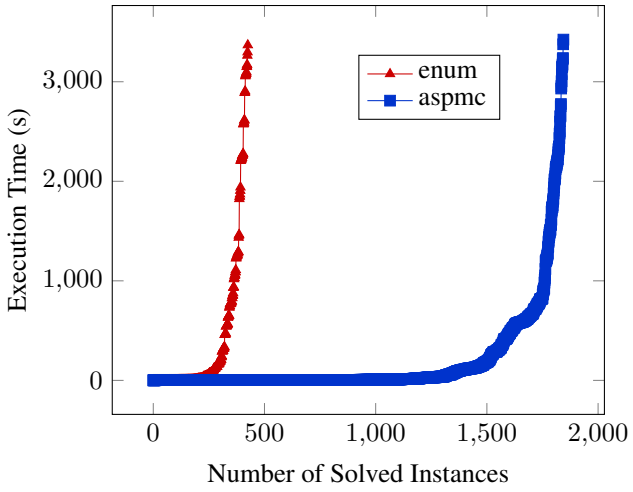

Figure 1: Cactus plot comparing the two solvers on all instances.

of these, we considered thresholds of 0.25, 0.5, and 0.75. Thus, in total, we run 3024 experiments for each of the two solvers (*aspmc* and *enum*).

The research questions we aim to answer are: Q1) what is the gap of performance provided by KC-based approaches w.r.t. enumeration-based approaches? Q2) does the selected threshold have an impact on the number of solved instances?

Results. Figure 1 shows a cactus plot comparing the performance of *aspmc* and *enum* on all instances for every threshold considered. As expected, *aspmc* can solve more instances w.r.t. *enum*, approximately 4 times more, which allows us to answer Q1 and assess the benefits provided by KC-approaches w.r.t. enumeration.

Table 1 provides a more fine-grained view, separating the BA and ER instances over the three thresholds considered. Here, we note that, for *enum*, the results seem to be independent of the threshold k. Similarly, for *aspmc*, thresholds do not play a crucial role, with a difference of at most 7 solved instances. This allows us to answer Q2 and state that the threshold does not seem to play a role in scalability. This can be expected since *enum* enumerates all answer sets of the outer program (independently of k) and then, for each one, calls a solver. On the other hand, *aspmc* compiles the whole program into a compact formula. Thus, k is taken into account only during the evaluation of this formula and therefore does not reduce the size of the overall program. However, a more strict value of k allows the solver to prune some answer sets during the evaluation process. Lastly, we note that for the instances that cannot be solved, *enum* stops due to the time limit while *aspmc* due to a knowledge compilation error, possibly due to the excessive memory requirements (i.e., we get "knowledge compilation failed" error).

8 Related Work

ASP(Q)^{Inf} extends the ASP(Q) framework (Amendola, Ricca, and Truszczynski 2019) by incorporating probabilistic reasoning. In recent years, ASP(Q) has attracted growing interest due to its ability to model computational problems across the PH and its successful application in several domains (Azzolini et al. 2025b; Faber, Morak, and Chrpa 2022; Faber and Morak 2022; Bellusci, Mazzotta, and Ricca 2022). Extending ASP(Q) remains an active line of research: for instance, preferences and optimization have been introduced via weak constraints (Mazzotta, Ricca, and Truszczynski 2024; Buccafurri, Leone, and Rullo 2000). However, these extensions do not address probabilistic reasoning, which requires aggregation over answer sets and is the focus of our work.

	BA			ER		
threshold	0.25	0.5	0.75	0.25	0.5	0.75
aspmc	342	345	349	269	268	272
enum	74	74	74	68	68	68

Table 1: Number of solved instances on the Barabási-Albert (BA) and Erdős-Renyi (ER) networks structures with different probability thresholds (0.25, 0.5, and 0.75).

Stable–unstable semantics (Bogaerts, Janhunen, and Tasharrofi 2016) and quantified answer set semantics (Fandinno et al. 2021) are closely related to ASP(Q), but do not support probabilistic reasoning. Our work may thus provide a basis for extending these higher-order ASP formalisms. A broader comparison of ASP extensions for PH problems can be found in (Amendola, Ricca, and Truszczynski 2019; Fandinno et al. 2021).

Within the probabilistic logic programming literature, several reasoning tasks related to ASP(Q)^{Inf} have been studied, including decision-theoretic inference (framed by DTProbLog) (Van den Broeck et al. 2010), MAP inference (Bellodi et al. 2020; Shterionov et al. 2015), and extensions of PLP with abducible facts (Azzolini et al. 2022), many of which admit formulations within the 2AMC framework (Kiesel, Totis, and Kimmig 2022). Despite their relevance, these approaches focus on optimization rather than threshold-based reasoning, are limited to two reasoning levels, and do not support existential or universal variables, all of which are central features of ASP(Q)^{Inf}.

Several of the aforementioned tasks have also been investigated within the probabilistic answer set programming (PASP) framework (Cozman and Mauá 2020; Azzolini et al. 2025a). Although PASP effectively supports probabilistic reasoning over answer sets, existing approaches are limited to a single reasoning level and do not support quantifier alternation. Other probabilistic ASP formalisms, such as LPMLN (Lee and Yang 2017), smProbLog (Totis, De Raedt, and Kimmig 2023), and L-credal semantics (Rocha and Cozman 2022), as well as practical implementations (Hahn et al. 2025), rely on alternative probabilistic semantics to model uncertainty in logic programs. Although effective in their respective settings, these formalisms are orthogonal to the contributions of ASP(Q)^{Inf}. Extending ASP(Q)^{Inf} to accommodate PASP as the innermost component, or considering alternative semantics therefore constitutes promising directions for future work.

9 Conclusion

In this paper, we presented ASP(Q)^{Inf}, extending ASP(Q) with probabilistic inference. We analyzed its complexity, demonstrated its modeling capabilities on real-world scenarios, and showed that it can be practically implemented using Algebraic Model Counting. Our results highlight the feasibility and relevance of combining probabilistic reasoning with highly expressive ASP extensions.

Future work includes the development of a full-fledged solver for ASP(Q)^{Inf}, and the investigation of additional reasoning tasks, such as MAP inference.

Acknowledgments

This work has been partially funded by the Italian Ministry of Industrial Development (MISE) under project EI-TWIN n. F/310168/05/X56 CUP B29J24000680005; by the Italian Ministry of Research (MUR) under project PNRR FAIR - Spoke 9 - WP 9.1 CUP H23C22000860006; and also partially by projects SIGENERA (CUP J29I24001770005) and MOZART (J49I24001740005) selected within the framework of the PR FESR – FSE Calabria 2021/2027 and implemented with the support of the Italian State and the Calabria Region. DA is member of the Gruppo Nazionale Calcolo Scientifico – Istituto Nazionale di Alta Matematica (GNCS-INdAM).

AI Declaration

The authors have not employed any Generative AI tools.

References

Alviano, M.; Calimeri, F.; Dodaro, C.; Fuscà, D.; Leone, N.; Perri, S.; Ricca, F.; Veltri, P.; and Zangari, J. 2017. The ASP system DLV2. In *LPNMR*, volume 10377 of *Lecture Notes in Computer Science*, 215–221. Springer.

Amendola, G.; Ricca, F.; and Truszczynski, M. 2019. Beyond NP: quantifying over answer sets. *Theory and Practice of Logic Programming* 19(5-6):705–721.

Azzolini, D.; Bellodi, E.; Ferilli, S.; Riguzzi, F.; and Zese, R. 2022. Abduction with probabilistic logic programming under the distribution semantics. *International Journal of Approximate Reasoning* 142:41–63.

Azzolini, D.; Mazzotta, G.; Ricca, F.; and Riguzzi, F. 2025a. An algebraic view of MAP inference in probabilistic answer set programs. In *ECAI 2025 28th - European Conference on Artificial Intelligence, 25-30 October 2025, Bologna, Italy - Including 14th Conference on Prestigious Applications of Intelligent Systems (PAIS 2025)*, volume 413 of *Frontiers in Artificial Intelligence and Applications*. IOS Press.

Azzolini, D.; Mazzotta, G.; Ricca, F.; and Riguzzi, F. 2025b. Most probable explanation in probabilistic answer set programming. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2025, Montreal, Canada, August 16-22, 2025*, 9049–9057. ijcai.org.

Barabasi, A.-L., and Albert, R. 1999. Emergence of scaling in random networks. *Science* 286(5439):509–512.

Baral, C.; Gelfond, M.; and Rushton, N. 2009. Probabilistic reasoning with answer sets. *Theory and Practice of Logic Programming* 9(1):57–144.

Baselice, S.; Bonatti, P. A.; and Gelfond, M. 2005. Towards an integration of answer set and constraint solving. In Gabbrielli, M., and Gupta, G., eds., *Logic Programming, 21st International Conference, ICLP 2005, Sitges, Spain, October 2-5, 2005, Proceedings*, volume 3668 of *Lecture Notes in Computer Science*, 52–66. Springer.

Bellodi, E.; Alberti, M.; Riguzzi, F.; and Zese, R. 2020. MAP inference for probabilistic logic programming. *TPLP* 20(5):641–655.

Bellusci, P.; Mazzotta, G.; and Ricca, F. 2022. Modelling the outlier detection problem in ASP(Q). In *PADL*, volume 13165 of *Lecture Notes in Computer Science*, 15–23. Springer.

Bogaerts, B.; Janhunen, T.; and Tasharrofi, S. 2016. Stable-unstable semantics: Beyond NP with normal logic programs. *TPLP* 16(5-6):570–586.

Brewka, G.; Eiter, T.; and Truszczynski, M. 2011. Answer set programming at a glance. *Commun. ACM* 54(12).

Buccafurri, F.; Leone, N.; and Rullo, P. 2000. Enhancing disjunctive datalog by constraints. *IEEE Transactions on Knowledge and Data Engineering* 12(5):845–860.

Cozman, F. G., and Mauá, D. D. 2020. The joy of probabilistic answer set programming: Semantics, complexity, expressivity, inference. *International Journal of Approximate Reasoning* 125:218–239.

Dantsin, E.; Eiter, T.; Gottlob, G.; and Voronkov, A. 2001. Complexity and expressive power of logic programming. *ACM Computing Surveys* 33(3):374–425.

De Raedt, L.; Kimmig, A.; and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In Veloso, M. M., ed., *20th International Joint Conference on Artificial Intelligence (IJCAI 2007)*, volume 7, 2462–2467. AAAI Press.

Domingos, P., and Richardson, M. 2001. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, 57–66.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence* 15(3-4):289–323.

Eiter, T.; Hecher, M.; and Kiesel, R. 2024. aspmc: New frontiers of algebraic answer set counting. *Artificial Intelligence* 330:104109.

Eiter, T.; Redl, C.; and Schüller, P. 2016. Problem solving using the HEX family. In *Computational Models of Rationality*, 150–174. College Publications.

Erdem, E.; Gelfond, M.; and Leone, N. 2016. Applications of answer set programming. *AI Magazine* 37(3):53–68.

Erdős, P., and Rényi, A. 1959. On random graphs i. *Publicationes Mathematicae Debrecen* 6(290-297):18.

Faber, W., and Morak, M. 2022. Evaluating epistemic logic programs via answer set programming with quantifiers. In

HYDRA/RCRA@LPNMR, volume 3281 of *CEUR Workshop Proceedings*, 78–89. CEUR-WS.org.

Faber, W.; Morak, M.; and Chrpa, L. 2022. Determining action reversibility in STRIPS using answer set programming with quantifiers. In *PADL*, volume 13165 of *Lecture Notes in Computer Science*, 42–56. Springer.

Falkner, A. A.; Friedrich, G.; Schekotihin, K.; Taupe, R.; and Teppan, E. C. 2018. Industrial applications of answer set programming. *Künstliche Intelligenz* 32(2-3):165–176.

Fandinno, J.; Laferrière, F.; Romero, J.; Schaub, T.; and Son, T. C. 2021. Planning with incomplete information in quantified answer set programming. *Theory Pract. Log. Program.* 21(5):663–679.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Wanko, P. 2016. Theory solving made easy with clingo 5. In *ICLP (Technical Communications)*, volume 52 of *OASICS*, 2:1–2:15. Schloss Dagstuhl.

Gebser, M.; Kaminski, R.; Kaufmann, B.; and Schaub, T. 2019. Multi-shot ASP solving with clingo. *Theory and Practice of Logic Programming* 19(1):27–82.

Gelfond, M., and Lifschitz, V. 1991a. Classical negation in logic programs and disjunctive databases. *New Gener. Comput.* 9(3/4):365–386.

Gelfond, M., and Lifschitz, V. 1991b. Classical negation in logic programs and disjunctive databases. *New generation computing* 9(3):365–385.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2004. *Automated planning - theory and practice*. Elsevier.

Gill, J. 1977. Computational complexity of probabilistic turing machines. *SIAM J. Comput.* 6(4):675–695.

Hagberg, A. A.; Schult, D. A.; and Swart, P. J. 2008. Exploring network structure, dynamics, and function using NetworkX. In Varoquaux, G.; Vaught, T.; and Millman, J., eds., *Proceedings of the 7th Python in Science Conference*, 11–15.

Hahn, S.; Janhunen, T.; Kaminski, R.; Romero, J.; Rühling, N.; and Schaub, T. 2025. Plingo: A system for probabilistic reasoning in answer set programming. *Theory and Practice of Logic Programming* 25(2):134–167.

Kiesel, R.; Totis, P.; and Kimmig, A. 2022. Efficient knowledge compilation beyond weighted model counting. *Theory and Practice of Logic Programming* (4):505–522.

Kimmig, A.; Van den Broeck, G.; and De Raedt, L. 2017. Algebraic model counting. *J. of Applied Logic*.

Kushmerick, N.; Hanks, S.; and Weld, D. S. 1995. An algorithm for probabilistic planning. *Artif. Intell.* 76(1-2):239–286.

Lee, J., and Wang, Y. 2016. Weighted rules under the stable model semantics. In Baral, C.; Delgrande, J. P.; and Wolter, F., eds., *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*, 145–154. AAAI Press.

Lee, J., and Yang, Z. 2017. LPMLN, weak constraints, and P-log. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, 1170–1177. AAAI Press.

Mauá, D. D., and Cozman, F. G. 2020. Complexity results for probabilistic answer set programming. *International Journal of Approximate Reasoning* 118:133–154.

Mazzotta, G.; Ricca, F.; and Truszczynski, M. 2024. Quantifying over optimum answer sets. *Theory and Practice of Logic Programming* 24(4):716–736.

Nath, A., and Domingos, P. 2009. A language for relational decision theory. In *Proceedings of the International Workshop on Statistical Relational Learning*.

O'Neil, R.; Diallo, C.; Khatab, A.; and Rezg, N. 2025. Enhancing critical network infrastructure resilience through optimal post-disruption maintenance and routing decisions. *Reliability Engineering & System Safety* 257:110717.

Raedt, L. D.; Kersting, K.; Natarajan, S.; and Poole, D. 2016. *Statistical Relational Artificial Intelligence: Logic, Probability, and Computation*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Riguzzi, F. 2022. *Foundations of Probabilistic Logic Programming Languages, Semantics, Inference and Learning, Second Edition*. Gistrup, Denmark: River Publishers.

Rocha, V. H. N., and Cozman, F. G. 2022. A credal least undefined stable semantics for probabilistic logic programs and probabilistic argumentation. In *KR*.

Sato, T. 1995. A statistical learning method for logic programs with distribution semantics. In Sterling, L., ed., *Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming*, 715–729. MIT Press.

Shterionov, D. S.; Renkens, J.; Vlasselaer, J.; Kimmig, A.; Meert, W.; and Janssens, G. 2015. The most probable explanation for probabilistic logic programs with annotated disjunctions. In *24th International Conference on Inductive Logic Programming (ILP 2014)*, volume 9046 of *Lecture Notes in Computer Science*, 139–153. Berlin, Heidelberg: Springer.

Stockmeyer, L. J. 1976. The polynomial-time hierarchy. *Theor. Comput. Sci.* 3(1):1–22.

Stockmeyer, L. J. 1983. The complexity of approximate counting (preliminary version). In Johnson, D. S.; Fagin, R.; Fredman, M. L.; Harel, D.; Karp, R. M.; Lynch, N. A.; Papadimitriou, C. H.; Rivest, R. L.; Ruzzo, W. L.; and Seiferas, J. I., eds., *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, 118–126. ACM.

Torán, J. 1991. Complexity classes defined by counting quantifiers. *J. ACM* 38(3):753–774.

Totis, P.; De Raedt, L.; and Kimmig, A. 2023. smProbLog: Stable model semantics in problog for probabilistic argumentation. *Theory and Practice of Logic Programming* 1–50.

Turner, H. 2002. Polynomial-length planning spans the polynomial hierarchy. In Flesca, S.; Greco, S.; Leone, N.;

and Ianni, G., eds., *Logics in Artificial Intelligence, European Conference, JELIA 2002, Cosenza, Italy, September, 23-26, Proceedings*, volume 2424 of *Lecture Notes in Computer Science*, 111–124. Springer.

Van den Broeck, G.; Thon, I.; van Otterlo, M.; and De Raedt, L. 2010. DTProbLog: A decision-theoretic probabilistic Prolog. In Fox, M., and Poole, D., eds., *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 1217–1222. AAAI Press.

Van Gelder, A.; Ross, K. A.; and Schlipf, J. S. 1991. The well-founded semantics for general logic programs. *Journal of the ACM* 38(3):620–650.

Vennekens, J.; Verbaeten, S.; and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In *International Conference on Logic Programming*, 431–445. Springer.

Wagner, K. W. 1986. The complexity of combinatorial problems with succinct input representation. *Acta Informatica* 23(3):325–356.

Zhang, X.; Grastien, A.; and Gretton, C. 2024. A counterexample based approach to probabilistic conformant planning. In Bernardini, S., and Muise, C., eds., *Proceedings of the Thirty-Fourth International Conference on Automated Planning and Scheduling, ICAPS 2024, Banff, Alberta, Canada, June 1-6, 2024*, 689–697. AAAI Press.