Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Reasoning with Restricted Statistical Statements in Probabilistic Answer Set Programming: Complexity and Algorithms

Damiano Azzolini[1] and **Markus Hecher**[2]
[1]University of Ferrara, Ferrara, Italy
[2]Univ. Artois, CNRS, UMR8188, Computer Science Research Center of Lens (CRIL), Lens, France
damiano.azzolini@unife.it, hecher@mit.edu

Abstract

Statistical statements are an expressive tool for representing statistical information of a domain of interest. Recently, these statements were given a meaning in the context of Probabilistic Answer Set Programming (PASP), allowing one to encode properties like "x% of elements of a domain have the feature y". Although the computational complexity of different tasks in PASP is well known, the complexity of restricted programs composed only of statistical statements and probabilistic facts has not been studied. As a first contribution, we address this problem, confirming that even in seemingly restricted cases the complexity is high. Indeed, even with this restriction we do not lose expressiveness, reaching higher levels of the polynomial hierarchy. To mitigate these high complexities, we focus on the structure of the programs. Thereby, we design novel structure-guided reductions, demonstrating how one can efficiently answer queries along treewidth decompositions. We obtain precise upper bounds and we show that under reasonable assumptions in complexity theory we cannot significantly improve, as we give matching lower bounds.

1 Introduction

Probabilistic Answer Set Programming under the credal semantics (PASP) (Cozman and Mauá 2020) is one of the possible languages to express uncertainty with a logic-based language. PASP extends Answer Set Programming (ASP) (Brewka, Eiter, and Truszczyński 2011) with probabilistic facts (De Raedt, Kimmig, and Toivonen 2007), i.e., facts associated with a probability.

Recently, statistical statements, initially proposed by Halpern in 1990 (Halpern 1990) to express statistical information of a domain, have been given an encoding in PASP (Azzolini, Bellodi, and Riguzzi 2022; Azzolini and Riguzzi 2023) with the PASTA language. This encoding converts each statement into a disjunctive rule and two constraints that contain counting aggregates. Here, we focus on programs composed of probabilistic facts and statistical statements only, and call them "PASTA programs". Although the computational complexity of general PASP is well studied (Mauá and Cozman 2020; Cozman and Mauá 2020), it is unknown whether the fragment of PASTA programs belongs to different complexity classes. That is, we want to check whether such programs are powerful enough to model any PASP. This is indeed not trivial, as PASTA seems rather restricted: We can only model statements of the form "x% of domain elements have feature y". Here, we want to answer the following questions: How do these restricted forms of probabilistic logic programming behave complexity-wise? Can we do better than with seemingly more expressive formalisms in probabilistic reasoning?

Contributions. In this paper, we close this gap and study the complexity of cautious inference and most probable explanation (MPE) in PASTA programs, providing a more fine-grained and refined view and identifying the core sources of complexity. We derive complexity bounds for the whole class of PASTA programs as well as for some of its subfragments (namely, without disjunction and without negation). Surprisingly, we find that solving such tasks *still remains hard*. After this, given the high complexity of these probabilistic tasks, in the second part of the paper, we analyze whether *structural properties* like treewidth can help in solving reasoning tasks efficiently. We devise an algorithm to answer queries along a tree decomposition and present the first parameterized complexity results for PASP. It turns out that our reductions and algorithms are in line with what we can hope for. Indeed, while the runtime is exponential in the treewidth, we also give precise *matching runtime lower bounds* under reasonable assumptions in computational complexity. So, we do not expect significant improvements. For an overview of our results, see Table 1.

Structure. The paper is structured as follows: Section 2 introduces background concepts and the PASTA language. Section 3 discusses the complexity of inference and MPE in PASTA programs. Section 4 shows how we can leverage structural properties of the programs for inference. Section 5 illustrates related work and Section 6 concludes the papers with some directions for future works.

2 Background

Here we briefly summarize the background concepts.

Tree Decompositions. Given a graph $G = (V, E)$, where V and E are the sets of vertices and edges, respectively, a tree decomposition (TD) of G is a pair $\mathcal{T} = (T, \chi)$ where T is a rooted tree and χ is a mapping assigning each node $t \in T$ a set $\chi(t) \subseteq V$. $\chi(t)$ is called *bag*. A tree decomposition has the following properties: i) $V = \bigcup_{t \in T} \chi(t)$; ii) for all pairs of nodes $(v, w) \in E$, there exists a $\chi(t)$ s.t. $(v, w) \subseteq \chi(t)$;

Fragment	Inference	MPE	TW Lower Bound
PASTA	$PP^{\Sigma_2^P}$ (Thm. 1)	Σ_3^P (Corr. 2)	$2^{2^{2^{o(k)}}}$ (Thm. 5)
PASTA$_{\not\gamma}$	PP^{NP} (Thm. 2)	Σ_2^P (Corr. 3)	$2^{2^{o(k)}}$ (Corr. 5)
PASTA$_{\not\gamma\vee}$	PP (Corr. 1)	NP (Corr. 4)	$2^{o(k)}$ (Corr. 5)

Table 1: Overview of obtained results in this paper. Results include completeness of PASTA and exponential time hypothesis-based runtime lower bounds for treewidth (excluding polynomial factors). Corresponding matching runtime upper bounds (for tight programs) are given through Reduction 1+Algorithm 1 for non-disjunctive programs without negation, as well as Reduction 2 for non-disjunctive programs. Disjunctions can be eliminated by exponentially increasing treewidth, e.g., see (Hecher and Kiesel 2024).

iii) for each triple of nodes r, s, $t \in T$ where s lies on the path from r to t, $\chi(r) \cap \chi(t) \subseteq \chi(s)$. The width of a tree decomposition is $width(\mathcal{T}) = max_{t \in T}|\chi(t)| - 1$. There may be multiple tree decompositions for a graph G. The treewidth of a graph G is the minimum $width(\mathcal{T}_i)$ over all its tree decompositions $\mathcal{T}_i$. Computing the exact treewidth is hard (Bodlaender 1988a; Bodlaender and Koster 2008), so it is often approximated using heuristics. Let $\mathrm{chld}(t)$ denote the set of child nodes of a node t and t^* be the parent of a node t. Without loss of generality (Bodlaender and Koster 2008), we assume that tree decompositions (T, χ) are *nice*, i.e., they are composed of four types of nodes only:

- node t with $\mathrm{type}(t)$=*leaf*, where $\mathrm{chld}(t)=\emptyset$ and $\chi(t)=\emptyset$,
- node t with $\mathrm{type}(t)$=*intr* introduces a single vertex, i.e., $\chi(t)=\chi(t_1)\cup\{a\}$ and $\{t_1\}=\mathrm{chld}(t)$ for some vertex a,
- node t with $\mathrm{type}(t)$=*rem* removes a single vertex, i.e., $\chi(t)=\chi(t_1)\setminus\{a\}$ and $\{t_1\}=\mathrm{chld}(t)$ for some vertex a, and
- node t with $\mathrm{type}(t)$=*join* joins two child nodes, i.e., $\chi(t) = \chi(t_1) = \chi(t_2)$ where $\{t_1, t_2\} = \mathrm{chld}(t)$.

Dynamic Programming. Usually, tree decompositions are paired with dynamic programming (DP) algorithms (Bodlaender 1988b). Each node t in the tree decomposition is associated with a table τ_t containing intermediate results obtained via a *table algorithm* applied on all children t' of t (and thus considering all tables $\tau_{t'}$). At a high level, a DP algorithm works in the following four steps: given an instance of a problem, 1) it builds a graph representation G; 2) it computes a (possibly sub-optimal) TD $\mathcal{T} = (T, \chi)$ of G; 3) T is traversed bottom up and at each node t a table algorithm is executed on the subproblem identified by t, $\chi(t)$, the current local instance of the problem, and child tables $\tau_{t'}$; lastly, 4) at the root r, the table τ_r contains the solution of the considered problem.

Computational Complexity. We briefly recall the polynomial time hierarchy (PH) (Stockmeyer 1976; Papadimitriou 1994): $\Delta_0^P = \Pi_0^P = \Sigma_0^P = P$ and $\Delta_{k+1}^P = P^{\Sigma_k^P}$, $\Sigma_{k+1}^P = NP^{\Sigma_k^P}$, $\Pi_{k+1}^P = coNP^{\Sigma_k^P}$ for $k > 0$ where D^O denotes the class of decision problems D equipped with an oracle for complete problems in O. The PP class (Gill 1977) is the class of decision problems solved by an NP machine such that if the machine's answer is positive, at least half of the total computation paths accept, while, otherwise, less than half of the total computation paths accept.

Answer Set Programming. Answer Set Programming (ASP) is a well-studied formalism, particularly efficient in modeling combinatorial problems (Gelfond and Lifschitz 1988; Brewka, Eiter, and Truszczyński 2011; Erdem, Gelfond, and Leone 2016). A *disjunctive* rule r is of the form

$$h_1; \ldots; h_m :- b_1, \ldots, b_m$$

where each h_1 is an atom and each b_1 is a literal. The $:-$ operator divides the *head* (disjunction of the h_is) denoted as $H(r)$ from the *body* (conjunction of b_is). With $B^+(r)$ and $B^-(r)$ we denote the set of positive and *(default) negated* literals appearing in the body of a rule r, respectively. If the head is empty and the body is not, the rule is also called *constraint*. If there is only one atom in the head and the body is empty, the rule is called *fact*. We also consider *aggregates* in the body of rules of the form $\#count\{V : T\} = C$ where V is the set of variables appearing in the conjunction of literals T and C is a variable that will contain the result of the aggregate. In general, aggregates may have more complex structures (Faber, Pfeifer, and Leone 2011), but here we limit the treatment to the structures used in this paper. An *answer set (ASP) program* is a finite set of disjunctive rules.

The *dependency graph* $D_\mathcal{P}$ of a ground program $\mathcal{P}$ is a graph whose vertices V are the predicates appearing in the grounding of $\mathcal{P}$. Two vertices p_0 and p_1 are connected by a positive (resp. negative) edge if there is a rule r such that $p_0 \in B^+(r)$ (resp. $not\ b \in B^-(r)$) and $p_1 \in H(r)$. A program $\mathcal{P}$ is *tight* if $D_\mathcal{P}$ is acyclic, is *stratified* if $D_\mathcal{P}$ has no loops involving negative edges, and is *head-cycle-free* (Linke, Tompits, and Woltran 2004) if its dependency graph does not contain a directed cycle going through two head-sharing atoms. The *primal graph* $P_\mathcal{P}$ of a program $\mathcal{P}$ has the same vertices as $D_\mathcal{P}$ and has an undirected edge between p_0 and p_1 if there is a rule with both appearing in it.

With B_P we denote the set of Herbrand base of an ASP Π, i.e., the set of all ground atoms in Π. For a program Π we can obtain its grounding by replacing variables with constants in all possible ways. The grounding of aggregates requires considering local and global variables. Here we focus on aggregates with local variables only, i.e., variables appearing only in the aggregate they are defined. The grounding of an aggregate with local variables only requires replacing local variables with ground terms in all possible ways.

A literal l is true in an interpretation $I \subseteq B_P$ if $l \in I$. I is a model if it satisfies all groundings of Π. Given an interpretation I we can obtain the *reduct* (Gelfond and Lifschitz 1991) of Π w.r.t. I by removing from Π every rule with body false in I. An interpretation I is called answer set of a program Π if it is a minimal model (under set inclusion) of the reduct of Π w.r.t. I. We denote with $AS(\Pi)$ the set of answer sets of a program Π.

Example 1. *Consider the following answer set program* Π.

```
a.
b.
p ; q :- a.
r :- b.
```

It has two answer sets: $AS(\Pi) = \{\{a, b, p, r\}, \{a, b, q, r\}\}$.

Probabilistic Answer Set Programming. A probabilistic answer set program (PASP) (Cozman and Mauá 2020) is an answer set program extended with ground probabilistic facts (De Raedt, Kimmig, and Toivonen 2007) that allow representing uncertain information. A probabilistic fact is of the form $p_i :: a_i$ where $p_i \in [0, 1]$ is the probability associated with the ground atom a_i. We denote with $pf(\mathcal{P})$ the set of probabilistic facts that appear in a PASP $\mathcal{P}$. We interpret such programs under the *credal semantics (CS)* (Cozman and Mauá 2020): each possible subset of probabilistic facts identifies a *world*, i.e., an ASP (deterministic) composed by the rules in $\mathcal{P}$ and the facts in the considered subset. Each PASP has 2^n worlds, where n is the number of probabilistic facts. The probabilistic facts are considered independent so the probability of a world w is computed as

$$P(w) = \prod_{a_i \in w} p_i \prod_{a_i \notin w} (1 - p_i).$$

The probabilities of all the worlds sum to 1. A conjunction of ground literals q, often called *query*, is associated to a lower ($\underline{\mathrm{P}}(q)$) and an upper ($\overline{\mathrm{P}}(q)$) probability. In formulas,

$$\underline{\mathrm{P}}(q) = \sum_{w_i | \forall m \in AS(w_i),\ m \models q} P(w_i)$$

and

$$\overline{\mathrm{P}}(q) = \sum_{w_i | \exists m \in AS(w_i),\ m \models q} P(w_i).$$

Without loss of generality, we can consider a single atom as query (by adding a new rule in the program with q in the body and a fresh atom q' in the head, which will be considered as the new query). The upper probability of a query q and the lower probability of the negation of q are related: $\underline{\mathrm{P}}(q) = 1 - \overline{\mathrm{P}}(not\ q)$. If some worlds have no answer sets, this relation does not hold, so we have a loss of probability mass. To avoid this, the CS requires that *every world* has at least one answer set. We call *inference* the task of computing the probability (bounds) for a query. Given a ground atom e (also in this case, we can consider a single atom w.l.o.g.), the most probable explanation (MPE) task requires finding the subset of probabilistic facts identifying the world with the highest probability and in which e is present in every answer set ($\underline{\mathrm{MPE}}(e)$) or in some answer sets ($\overline{\mathrm{MPE}}(e)$).

Example 2. *Consider the following PASP.*

```
0.7::a.
0.8::b.
p ; q :- a.
q :- b.
```

There are 2^2 *worlds:* w_0 *where both* a *and* b *are absent,* w_1 *where* a *is present and* b *is absent,* w_2 *where* a *is absent and* b *is present, and* w_3 *where both* a *and* b *are present.* $P(w_0) = (1 - 0.7) \cdot (1 - 0.8) = 0.06$, $P(w_1) = 0.7 \cdot (1 - 0.8) = 0.14$, $P(w_2) = (1 - 0.7) \cdot 0.8 = 0.24$, *and* $P(w_3) = 0.7 \cdot 0.8 = 0.56$. $AS(w_0) = \{\}$, $AS(w_1) = \{\{a, p\}, \{a, q\}\}$, $AS(w_2) = \{\{b, q\}\}$, *and* $AS(w_3) = \{\{a, b, q\}\}$. *If we are interested in computing the probability of* q*, we obtain:* $\underline{\mathrm{P}}(q) = P(w_2) + P(w_3) = 0.8$ *and* $\overline{\mathrm{P}}(q) = P(w_1) + P(w_2) + P(w_3) = 0.94$. *For the same query,* $\underline{\mathrm{MPE}}(q) = \{a, b\}$*, which also coincides with* $\overline{\mathrm{MPE}}(q)$ *(but this does not hold in general).*

PASTA Statements. PASTA statements were recently introduced in (Azzolini, Bellodi, and Riguzzi 2022; Azzolini and Riguzzi 2023) and assigned a meaning by translating them into answer set rules. These give a practical encoding to what Halpern calls "Type" 1 statements, which describe statistical information about a domain of interest. Their syntax is

$$(C \mid A)[\pi_l, \pi_u]$$

where C is an (non-ground) atom, A is a conjunction of (non-ground) literals, and $\pi_l, \pi_u \in [0, 1], \pi_l \leq \pi_u$. Their interpretation is: "the fraction of As that have the property C is between π_l and π_u". We will refer to C and A respectively as *consequent* and *antecedent*. Note that the symbol "|" is used as a separator for the consequent and antecedent and does not mean disjunction (as sometimes happens in ASP), which we always indicate with ";". To simplify the notation, we omit inserting $[\pi_l, \pi_u]$ when both π_l and π_u are 1. For a statement $r = (C \mid A)[\pi_l, \pi_u]$, where C is a ground atom and A is a conjunction of ground literals, let us define $h(r) = C$ and $at(r) = \{C\} \cup \bigcup_{a \in A} at(a)$ where $at(a)$ is the atom associated with the literal a. For example, if $r = (c \mid a, not\ b)$, $h(r) = \{c\}$ and $at(r) = \{c, a, b\}$. We overload $at()$ by defining $at(\mathcal{P}) = \bigcup_{r \in \mathcal{P}} at(r)$. A PASTA program $\mathcal{P}$ is composed of a (possibly empty) set of probabilistic facts and a (possibly empty) set of PASTA statements. To perform inference, a PASTA statement is converted into a disjunctive rule $C; not\_C :- A$. This rule states that the property C may or may not hold for each element A. To impose the specified $[\pi_l, \pi_u]$ bounds, two constraints with two counting aggregates are added to the program, to mimic

$$\pi_l \leq \frac{\#count\{\mathbf{X} : A(\mathbf{X}), C(\mathbf{X})\}}{\#count\{\mathbf{X} : A(\mathbf{X})\}} \leq \pi_u \qquad (1)$$

where $\mathbf{X}$ is a set of variables and $A(\mathbf{X})$ and $C(\mathbf{X})$ are sets of literals. Namely, the constraints are i) $:- \#count\{\mathbf{X} : A(\mathbf{X})\} = CA$, $\#count\{\mathbf{X} : A(\mathbf{X}), C(\mathbf{X})\} = CAC$, $CAC < \pi_l \cdot CA$ and ii) $:- \#count\{\mathbf{X} : A(\mathbf{X})\} = CA$, $\#count\{\mathbf{X} : A(\mathbf{X}), C(\mathbf{X})\} = CAC$, $CAC > \pi_u \cdot CA$.

During the grounding of PASTA rules, every non-ground rule $(C \mid A)[\pi_l, \pi_u]$ is turned into a ground rule $(C' \mid A')[\pi_l, \pi_u]$, where C' is a ground atom and A' is a set of sets of ground literals. Indeed, each non-ground literal in A is a set (of ground literals) in A'. We will rely on this characterization when studying treewidth. Note that PASTA statements of the form $(c \mid a)[1, 1]$ represent a rule $c :- a$.

The original proposal Azzolini, Bellodi, and Riguzzi (2022) did not place emphasis on how the program should behave in the case that the denominator of Equation (1) is 0, since the translation into constraints hides this detail. For this reason, when the denominator is 0, we define it as *vacuously fulfilled*, i.e., always satisfied. Let us now introduce two simple examples.

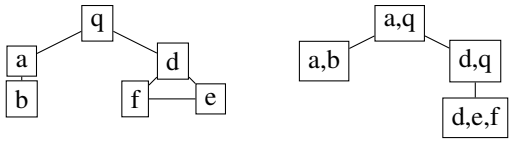

Figure 1: Primal graph (left) and a TD (right) for Example 4.

Example 3. *The following PASTA program models an urn with four marbles (indexed with increasing integers from 1 to 4) which can be made of wood with a certain probability, and where at least 40% of wooden made marbles are red.*

```
0.3::wooden(1).
0.1::wooden(2).
0.4::wooden(3).
0.8::wooden(4).
(red(X) | wooden(X))[0.4,1].
```

The statement is converted into

```
%a wooden marble may or may not be red
red(X) ; not_red(X) :-  wooden(X).
%at least 40% of wooden marbles are red
:- #count{X: wooden(X)} = W,
   #count{X:red(X), wooden(X)} = RW,
   100*RW < 40*W.
```

while the probabilistic facts are kept as they are. Note that, since ASP does not support floating point numbers, we multiply the bounds by 100. Also note that the constraint involving π_u (1 in this example) is omitted, since it imposes that at most 100% (i.e., all) of the wooded marbles are red. In the world where all $wooden/1$ facts are not present, call it w_0, both W and RW are 0. Since $100 \cdot 0 \not< 40 \cdot 0$, the constraint is always satisfied. If we are interested in, for example, computing the probability that the marble indexed with 1 is red ($red(1)$), we obtain the range $[0.0324, 0.3]$. The lower MPE state is $\{w(1)\}$, where $w/1$ stands for $wooden/1$, with a value of 0.0324 while the upper MPE state is $\{w(1), w(4)\}$, with a probability of 0.1296.

Example 4. *Consider the following simple PASTA program.*

```
0.1::b. 0.6::e.
(q | a)[1,1]. (a | b)[1,1]. (q | d)[1,1].
(d | e)[1,1]. (d | f)[1,1]. (f | e)[1,1].
```

Its primal graph and one possible tree decomposition are shown in Figure 1. Note that the treewidth of this program is 2 since there is a clique of size 3 in the primal graph. All four worlds have only one answer set. If we consider the query q, $\underline{\mathrm{P}}(q) = \overline{\mathrm{P}}(q) = 0.64$ and $\underline{\mathrm{MPE}}(q) = \overline{\mathrm{MPE}}(q) = \{e\}$ with probability 0.54.

3 The Complexity of PASTA Inference

Let us now introduce two decision problems that were studied in (Mauá and Cozman 2020) for PASP.

Definition 1 (Inference)**.** *Given a propositional PASP $\mathcal{P}$, a set of ground literals q (query), a set of ground atoms e (evidence), and a rational number γ, inference requires deciding whether $\underline{\mathrm{P}}(q \mid e) \geq \gamma$.*

Note that we can decide whether $\overline{\mathrm{P}}(q \mid e) \leq \gamma$ by deciding whether $\underline{\mathrm{P}}(not\ q \mid e) \geq 1 - \gamma$, so one problem can be reconducted to the other. Thus, considering only one of the two is sufficient.

Definition 2 (Most Probable Explanation)**.** *Given a propositional PASP $\mathcal{P}$ with probabilistic facts Q, a set of ground literals e (evidence), and a rational number γ, the Most Probable Explanation task decides whether* $\arg\max_q \underline{\mathrm{P}}(Q = q \mid e) \geq \gamma$.

The computational complexity of the two aforementioned tasks highly depends on the considered fragment (Mauá and Cozman 2020) and span different levels of the polynomial hierarchy. However, the special case of the PASTA fragment was not considered. Here, we ask the question of whether restricting to PASTA programs has an impact on the computational complexity of the two aforementioned tasks. In the following, we only consider ground PASTA programs, which is already hard.

3.1 How Hard is PASTA?

We now study the complexity of the two aforementioned tasks, starting from inference.

Inference

Theorem 1 (Complexity)**.** *Inference in ground PASTA programs is $PP^{\Sigma_2^P}$-complete.*

Proof. Hardness: reduce from $\#_{\geq q} X.\forall Y.\exists Z.\varphi(X, Y, Z) = \#_{\geq q} X.\forall Y.\neg(\forall Z.\neg\varphi(X, Y, Z))$. Observation: We can deterministically derive information if both probability bounds are 1. We construct a PASTA program $\mathcal{P}$, where for every variable x occurring in X, we add:

```
0.5::t(x).
```

For every 3-CNF term c and the i-th positive occurrence v_i (j-th negative occurrence v_j) add:

```
pos_i(c, v_i). neg_j(c, v_j).
```

Note that we can indeed map any atom a to a PASTA statement by using a as consequent and $\pi_l = \pi_u = 1$.

We guess truth value for every variable y in Y:

```
(t(y) | ⊤) [0,1].
```

Similarly, we guess truth values for every variable z in Z:

```
(t(z) | ⊤) [0,1].
```

We derive satisfiability via an auxiliary atom sat if c holds, for any 3-DNF term c in $\neg\varphi$. Concretely, for a 3-DNF term $c = u \wedge \neg v \wedge w$ over variables in $X \cup Y \cup Z$, we construct the following:

```
(sat | pos_1(c,u), t(u), neg_2(c,v), not_t(v),
    pos_3(c,w), t(w))[1,1].
```

Note that this construction reuses atoms over auxiliary predicate $not\_t/1$. Further, it easily extends in the same manner to any arbitrary 3-DNF term c', where the i-th positive occurrence of u in c' is addressed by $pos_i(c', u), t(u)$ and the j-th negative occurrence of v is referred to by $neg_j(c', v), not\_t(v)$.

To check whether this holds for all assignments, we apply the saturation technique (Eiter and Gottlob 1995) over Z:

```
(t(z) | sat)[1,1]. (not_t(z) | sat)[1,1].
```

Finally, to query for an atom instead of a negated atom, we derive $usat$ if sat does not hold (using default negation):

```
(usat | ¬sat)[1,1].
```

It is easy to observe that every world has an answer set, as we do not have constraints excluding a world. Then, $\underline{\mathrm{P}}(usat) \geq \frac{q}{2^{|X|}}$ iff $\#_{\geq q}X.\forall Y.\neg(\forall Z.\neg\varphi(X,Y,Z))$.

Membership: follows directly from previous work (Mauá and Cozman 2020, Table 1), which shows $PP^{\Sigma_2^P}$ membership for disjunction and default negation. □

Theorem 1 positions inference in PASTA programs in $PP^{\Sigma_2^P}$, the same complexity class obtained in (Mauá and Cozman 2020) for inference in propositional PASP with *any* construct. This shows that, despite being seemingly simpler, PASTA statements retain a huge expressive power.

We now provide other complexity results by further restricting this fragment, in particular not allowing negation and not allowing disjunction and negation.

Theorem 2 (Complexity)**.** *Inference in ground PASTA programs without negation or with only head-cycle-free disjunction (denoted with PASTA$_{\not\neg}$) is PP^{NP}-complete.*

Proof. Hardness: Reduce from $\#_{\geq q}X.\forall Y.\varphi(X,Y)$, similarly to above. However, we query for sat and skip the part about variables z in Z.

Membership: follows directly from previous work (Mauá and Cozman 2020, Table 1), which shows $PP^N P$ membership for aggregates and default negation. Indeed, disjunction can be directly translated to default negation (Eiter and Gottlob 1997), as by definition there cannot be cyclic dependencies between disjunctive head atoms. □

Corollary 1. *Inference in ground PASTA programs without disjunction and negation (denoted PASTA$_{\not\neg,\not\vee}$) is PP-complete.*

Proof (Idea). In this sub-fragment we only have probabilistic facts and derivable consequences, so we simply need to count the number of satisfying assignments. □

Most Probable Explanation (MPE)

Corollary 2. *Computing the most probable explanation for ground PASTA programs, is Σ_3^P-complete.*

Proof. The reduction works as in the proof of Theorem 1. It therefore follows that a given QBF $\exists X.\forall Y.(\neg\forall Z.\neg\varphi(X,Y,Z))$ is valid if and only if $\arg\max_q \underline{\mathrm{P}}(Q = q \mid sat_{\leq r}) \geq 0 + \epsilon$ for some $\epsilon > 0$. □

Corollary 3. *Computing the most probable explanation for ground PASTA programs without negation, is Σ_2^P-complete.*

Proof. Reduction works as in the proof of Theorem 2. □

This leads to the following result without disjunction.

Corollary 4. *Computing MPE for ground PASTA programs without disjunction and negation is NP-complete.*

4 Exploiting Structure to the Rescue

Given the high complexity of probabilistic reasoning, we exploit dynamic programming on tree decompositions (Bodlaender 1988a) for inference. This paradigm works for Logic Programming (Jakl, Pichler, and Woltran 2009; Fichte et al. 2017), but it has not been studied in detail for PASP. In the context of the PASTA fragment and for the ease of presentation, we mainly focus on tight programs (Fages 1994) (prohibiting cycles in the positive dependency graph of a program). Note that there are treewidth-aware translations readily available, see, e.g., (Hecher 2022; Eiter, Hecher, and Kiesel 2024), to translate a non-tight program into a tight program that only increase the treewidth from k to $k\log(k)$. While this might seem problematic, alternatively, one could extend our algorithms below. However, we probably cannot avoid this treewidth blowup to $k\log(k)$ (Hecher 2022).

4.1 Beginner: Make Programs Locally Provable

Before discussing a dynamic programming algorithm for exploiting treewidth, we first normalize PASTA programs to make them *locally provable*. This normalization reduction will later also simplify the reduction from non-tight PASTA programs to tight programs. That is, we simplify a given PASTA program $\mathcal{P}$, thereby transforming $\mathcal{P}$ along a tree decomposition. This results in a program $\mathcal{P}'$ such that provability of atoms in $\mathcal{P}'$ can be locally verified within a tree decomposition bag (hence the term "locally"). Consider a PASTA program $\mathcal{P}$. Let (T,χ) be a tree decomposition of the primal graph $P_\mathcal{P}$. For an atom C, let $\mathcal{P}_C = \{r \in \mathcal{P} \mid h(r) = C\}$ be the set of all *rules with C in the consequent*. For every node t in T, we define the *bag program* $\mathcal{P}_t = \bigcup_{C\in at(\mathcal{P}):at(\mathcal{P}_C)\subseteq\chi(t)} \mathcal{P}_C$ to contain *those* rules with C as a consequent that *only contain atoms in the bag*.

Let us introduce a simple running example that will be used several times in this section.

Example 5. *Consider this locally provable PASTA program.*

```
0.2::a. 0.3::b.
(q1 | a)[1,1]. (q2 | b)[1,1].
(q2 | q1)[1,1]. (q | q2)[1,1].
```

Its tree decomposition has only one path with the following bags (root to leaf): $[q]$, $[q,q2]$, $[q1,q2,b]$, and $[q1,a]$. The first two columns of Table 2 show a possible nice TD and the bag program at each node, respectively. Let us discuss some of them: at $[q1,a]$ the bag program is $\{(q1 \mid a)\}$ since there is only one statement with $q1$ in the consequent and a in the antecedent. At $[q1,q2]$, despite having a statement $(q2 \mid q1)$ in the program, the bag program is empty. To include it, we would need all statements with $q2$ in the consequent, so also $(q2 \mid b)$, but b is not present in this bag, so it is empty. This happens at the node $[q1,q2,b]$ above.

Definition 3 (Locally Provable)**.** *Let $\mathcal{P}$ be a PASTA program and $\mathcal{T}{=}(T,\chi)$ be a tree decomposition of $P_\mathcal{P}$. $\mathcal{P}$ is* locally provable *(for $\mathcal{T}$) if for every $a \in at(\mathcal{P})$ there is a unique node t in T with $\mathcal{P}_t$ containing rules of the form $(a \mid L)[\pi_l,\pi_u]$.*

Nice TD	Bag Programs	Tables
$[q]$	$\{\}$	$\{\langle\emptyset, 0.56\rangle, \langle\{q\}, 0.44\rangle\}$
$[q, q2]$	$\{(q \mid q2)\}$	$\{\langle\emptyset, 0.56\rangle, \langle\{q, q2\}, 0.44\rangle\}$
$[q2]$	$\{\}$	$\{\langle\emptyset, 0.56\rangle, \langle\{q2\}, 0.44\}$
$[q2, b]$	$\{\}$	$\{\langle\emptyset, 0.8\rangle, \langle\{q2\}, 0.2\rangle, \langle\{q2, b\}, 1.0\rangle\}$
$[q1, q2, b]$	$\{(q2 \mid q1), (q2 \mid b)\}$	$\{\langle\emptyset, 0.8\rangle, \langle\{q1, q2\}, 0.2\rangle, \langle\{q2, b\}, 0.8\rangle, \langle\{q1, q2, b\}, 0.2\rangle\}$
$[q1, q2]$	$\{\}$	$\{\langle\emptyset, 0.8\rangle, \langle\{q1\}, 0.2\rangle, \langle\{q2, 0.8\}\rangle, \langle\{q1, q2\}, 0.2\rangle\}$
$[q1]$	$\{\}$	$\{\langle\emptyset, 0.8\rangle, \langle\{q1\}, 0.2\rangle\}$
$[q1, a]$	$\{(q1 \mid a)\}$	$\{\langle\emptyset, 1\rangle, \langle\{q1, a\}, 1\rangle\}$
$[a]$	$\{\}$	$\{\langle\emptyset, 1\rangle, \langle\{a\}, 1\rangle\}$
$[]$	$\{\}$	$\{\langle\emptyset, 1\rangle\}$

Table 2: (Left): Nice tree decomposition (a path) of the program of Example 5 (that we also report here for clarity: $\{(q1 \mid a), (q2 \mid b), (q2 \mid q1), (q \mid q2).\}$). (Middle): bag programs. (Right): Tables obtained by applying Algorithm 1. Green denotes *intr* nodes while red *rem* nodes (bottom to top). The bottom node is a *leaf* so it is not colored.

Example 6. *Example 5 is locally provable as there are no two bag programs having statements with the same consequent.*

Next, we design a reduction to make any program $\mathcal{P}$ locally provable with only a linear increase of treewidth.

Reduction 1. *Let (T, χ) be a tree decomposition of a non-locally provable program $\mathcal{P}$. Then, we create $\mathcal{P}'$ by guiding the evaluation of $\mathcal{P}$ along (T, χ). Fist, we add in $\mathcal{P}'$*

Probabilistic Facts

$p{::}a$ *for every* $p{::}a \in \mathcal{P}$ (2)

Define Rule Provability

$(a \mid \top)[0, 1]$ *for every* $a \in at(\mathcal{P}) \setminus pf(\mathcal{P})$ (3)

Then, for every node t in T, we add the following rules:

Define Rule Provability

$(a_t \mid A)[\pi_l, \pi_u]$ *for every* $(a \mid A)[\pi_l, \pi_u] \in \mathcal{P}_t$ (4)

$(\bot_t \mid \neg\bot_t, \neg a, a_t)$ *for every* $a \in \chi(t) \setminus pf(\mathcal{P})$ (5)

Guide Provability Upwards

$(p^a_{\leq t} \mid a_t)$ *for every* $a \in \chi(t) \setminus pf(\mathcal{P})$ (6)

$(p^a_{\leq t} \mid p^a_{\leq t'})$ *for every* $t' \in \mathrm{chld}(t), a \in \chi(t) \cap \chi(t'), a \notin pf(\mathcal{P})$ (7)

Prohibit Unproven Atoms

$(\bot_t \mid \neg\bot_t, a, \neg p^a_{\leq t})$ *for every* $a \in \chi(t) \setminus pf(\mathcal{P})$, $a \notin \bigcup_{t^* : t \in \mathrm{chld}(t^*)} \chi(t^*)$ (8)

Equation (2) copies probabilistic facts and Equation (3) guesses truth values for the remaining atoms in $\mathcal{P}$. Then, in Equation (4) we derive a_t, whenever we are capable of proving a in a tree decomposition node t. Consequently, we must not derive a_t and do not have a, which is ensured by Equation (5). It remains to guide this information of provability upwards along the tree decomposition, which is ensured for a by Equation (6) due to local proofs and by Equation (7) if a has been proven already below t in the tree. It is left to model the equivalence to answer sets, which we ensure by Equation (8), verifying that if a is in an answer set candidate, it has to be proven up to the last bag where it occurs. Note that this reduction bijectively preserves answer sets since it neither removes nor creates additional worlds (see also Theorem 3). Note that this reduction linearly preserves the treewidth.

Theorem 3. *A PASTA program $\mathcal{P}$ can be converted to locally-provable program $\mathcal{P}'$ with a bijection between answer sets of $\mathcal{P}$ and those of $\mathcal{P}'$, which linearly preserves treewidth.*

Proof. Let $\mathcal{P}$ be a PASTA program and $\mathcal{T} = (T, \chi)$ be any tree decomposition of $P_\mathcal{P}$. Then, we refer to the program obtained from Equations (2)–(8) by $\mathcal{P}'$. For establishing linear treewidth increase, we construct a tree decomposition $\mathcal{T}' = (T, \chi')$ of $P_{\mathcal{P}'}$ as follows. For every node t in T, let $\chi'(t) = \chi(t) \cup \{\bot_t, p^a_{\leq t}, a_t \mid a \in \chi(t)\} \cup \{p^a_{\leq t^*} \mid t \in \mathrm{chld}(t^*), a \in \chi(t^*)\}$. Therefore, we have $|\chi'(t)| \leq 4|\chi(t)| + 1$ ($\bot_t$ causes the +1, remaining subscripted variables contribute to the factor 4).

Observe that $\mathcal{P}'$ is locally-provable on $\mathcal{T}'$. Indeed, by construction, every atom of $\mathcal{P}'$ can be proven in precisely one tree decomposition node of T. While Equation (3) might seem to allow free choices of every atom, it is ensured by Equation (5) that there is no choice to set a to false. Further, Equation (8) ensures that setting a to true implies that, eventually, we found a proof for a. Consequently, we have a bijective relationship between the answer sets of $\mathcal{P}$ and those of $\mathcal{P}'$. Therefore, we do not lose probability mass nor create additional mass. □

Example 7. *Suppose we have a PASTA program with $(q \mid a)$ and $(q \mid b)$ where $0.2 :: a$ and $0.3 :: b$. A possible TD has a root node t_2 with q having two child nodes containing $[q, a]$ (t_0) and $[q, b]$ (t_1), respectively. This has treewidth 1. This is not locally provable. If we apply Reduction 1, we get 2 rules from Equation (2), 1 rule from Equation (3), 3 rules from t_0 (from Equation (4), (5), and (6)), 3 rules from t_1 (from Equation (4), (5), and (6)), 5 rules from t_2 (from Equation (5), (6), 2 from Equation (7), and from Equation (8)) with treewidth 3. Each rule obtained from Reduction 1 is detailed in Table 3.*

Equation	t_0	t_1	t_2
4	$\{(q_{t_0} \mid a)\}$	$\{(q_{t_1} \mid b)\}$	$\emptyset$
5	$\{(\bot_{t_0} \mid not\ \bot_{t_0}, not\ q, q_{t_0})\}$	$\{(\bot_{t_1} \mid not\ \bot_{t_1}, not\ q, q_{t_1})\}$	$\{(\bot_{t_2} \mid not\ \bot_{t_2}, not\ q, q_{t_2})\}$
6	$\{(p^q_{\leq q_{t_0}} \mid q_{t_0})\}$	$\{(p^q_{\leq q_{t_1}} \mid q_{t_1})\}$	$\{(p^q_{\leq q_{t_2}} \mid q_{t_2})\}$
7	$\emptyset$	$\emptyset$	$\{(p^q_{\leq t_2} \mid p^q_{\leq t_1}), (p^q_{\leq t_2} \mid p^q_{\leq t_0})\}$
8	$\emptyset$	$\emptyset$	$\{(\bot_{t_2} \mid not\ \bot_{t_2}, q, not\ p^q_{\leq t_2})\}$

Table 3: Steps of Reduction 1 applied to Example 7.

Algorithm 1: Dynamic Programming Algorithm Strat-PASTA, called for every TD node t in a post-order traversal.

In: Node t, bag $\chi(t)$, bag program $\mathcal{P}_t$, child tables $\{\tau_1, \ldots \tau_\ell\}$.
Out: Table τ for node t.

1 **if** $\mathrm{type}(t) =$ *leaf* **then** $\tau := \{\langle \emptyset, 1\rangle\}$
2 **else if** $\mathrm{type}(t) =$ *intr and* $a \in \chi(t)$ *is introduced* **then**
3 $\quad \tau := \{\langle I', c\rangle \quad \mid \langle I, c\rangle \in \tau_1, I' \in \{I, I \cup \{a\}\},$
4 $\quad\quad I' \models \mathcal{P}_t, I' \cap h(\mathcal{P}_t) \subseteq pr(I', \mathcal{P}_t)\}$
5 **else if** $\mathrm{type}(t) =$ *rem and* $a \notin \chi(t)$ *is removed* **then**
6 $\quad \tau := \{\langle I \setminus \{a\}, \sum_{\langle J, c\rangle \in \tau_1 : J \setminus \{a\} = I \setminus \{a\}} \phi(a, J) \cdot c\rangle \mid \langle I, \cdot\rangle \in \tau_1\}$
7 **else if** $\mathrm{type}(t) =$ *join* **then**
8 $\quad \tau := \{\langle I, c_1 \cdot c_2\rangle \quad \mid \langle I, c_1\rangle \in \tau_1, \langle I, c_2\rangle \in \tau_2\}$

4.2 Intermediate: Stratified Tight Programs

From now on, given the efficient translation to locally provable programs, we assume $\mathcal{P}$ is *locally provable* (see Definition 3). We start with a simple dynamic programming algorithm for the case of stratified tight programs. This simplifies the presentation, as stratified programs guarantee that there is only a single answer set. As noted in the beginning of this section, focusing on tight programs is not a huge restriction as there are optimal translations (Hecher 2022) from non-tight programs. For simplicity, we present our dynamic programming algorithm for nice decompositions, but the algorithm extends to arbitrary decompositions, where cases may overlap. For gathering atoms that are justified by an interpretation, let $pr(I, \mathcal{P})$ be the set of atoms in $\mathcal{P}$ that are justified by I, i.e., where there is a rule deriving these. Formally, let $pr(I, \mathcal{P}) = pf(\mathcal{P}) \cup \{C \mid (C \mid A)[\pi_l, \pi_u] \in \mathcal{P}, I \models A\}$.

Dynamic Programming Algorithm. Algorithm 1 is called for every tree decomposition node t in a bottom-up (post-order) traversal, where we traverse nodes from the leaves of the decomposition towards its root and store information in tables. For every node t, we compute its *table* τ, thereby considering bag $\chi(t)$, bag program $\mathcal{P}_t$, and previously computed tables for child nodes of t.

To evaluate a PASTA program, we need to store in a table τ *(1) the assignments* I restricted to $\chi(t)$ and *(2) the weighted count* c over probabilistic facts needed to answer the query. For simplicity, we assume that the query comprises a *single atom* q that is solely present in the root of the tree decomposition, i.e., the root bag contains only q. Intuitively, (1) ensures that we only compute answer sets that fulfill every rule of the program and (2) allows computing the probability of the query q in the tree decomposition root by considering c. Since the program is stratified, there can be only a single answer set for each world. Hence, I can compute interpretations alongside answer set candidates.

The four cases of Algorithm 1 work as follows. Initially, in Line 1, since leaf bags are empty, the assignment I is empty, and its weight is 1, i.e., $c = 1$. Then, whenever we introduce an atom a, we take every preceding tuple and guess whether in the successor I' we set a to false or to true (see Line 3). Additionally, Line 4 concerns checking (i) satisfiability, i.e., we do not allow for disobeyed rules in $\mathcal{P}_t$ and (ii) provability, i.e., the atoms a in I that occur in the head of a rule in $\mathcal{P}_t$ are justified. The former (i) satisfiability is ensured by requiring $I' \models \mathcal{P}_t$; the latter (ii) provability is due to $I' \cap h(\mathcal{P}_t) \subseteq pr(I', \mathcal{P}_t)$. In Line 6, a is removed so we can forget about a as all rules containing a are already covered by the properties of a tree decomposition. There, we merge counts c whose corresponding interpretations are identical after removing a. Thereby, we need to weigh counts by the probability of a. Given a set of atoms J, the function $\phi(a_i, J)$ returns p_i if $a_i \in J$ and $1 - p_i$ if $a_i \notin J$ for a probabilistic fact $p_i :: a_i$. If a_i is not a probabilistic fact, $\phi(a_i, J)$ is assumed to return 1. Line 8 joins rows of compatible interpretations (both child nodes agree).

Example 8. *Consider again Example 5. Table 2 (third column) also shows the tables involved in the steps of Algorithm 1 while traversing the TD bottom up. Let us discuss some of them: at* $[q1, a]$ *we are at an intr node where* $q1$ *is introduced. Note that here we do not have only* $\{a\}$ *in the table since if we have* a *also* $q1$ *is present (there is* $(q1 \mid a)$*). At* $[q1]$*,* a*, which is a probabilistic fact with probability 0.2, is removed, so the counter of the first tuple is multiplied by (1-0.2) and the one of the second tuple by 0.2. At* $[q2, b]$*,* $q1$ *is removed. The last two tuples of the preceding node (*$[q1, q2, b]$*) are equal after the removal of* $q1$ *from the latter, so we have to sum their counters (0.8+0.2=1.0). Note that, not being* $q1$ *a probabilistic fact, we multiply by 1 (by definition of function* ϕ*, see Section 2). At* $[q2]$*, we remove* b*: for the right tuple, 0.44 is computed as* $0.2 \cdot 0.7$ *(since* b *is not present in* $\{q2\}$ *of the preceding node)* $+1.0 \cdot 0.3$.

Answering Queries. To answer a query $\underline{\mathrm{P}}(q)$ we check table τ_r for the root r. For computing $p{=}\underline{\mathrm{P}}(q)$, we query $p{=}\Sigma_{\langle\{q\}, c\rangle \in \tau_r} c$. Note that for stratified programs $\underline{\mathrm{P}}(q){=}\overline{\mathrm{P}}(q)$. Observe further that for stratified programs this algorithm extends to conditional probability queries.

Example 9 (Example 5 cont'd.)**.** *The table associated with the root node* $[q]$ *has two tuples: one with* $\{q\}$ *with associ-*

ated the probability of the query q being true and one with $\{\}$, associated with the probability of q being false. Note that the two probabilities sum to 1.

4.3 Advanced: Eliminate Non-Stratified Negation

We perform the following reduction that eliminates non-stratified negation by translation at the cost of an increase of treewidth. While this translation is not exponential in the instance size, it is exponential in the treewidth of the program. However, we will show later that this increase cannot be avoided (under reasonable complexity assumptions).

Reduction 2. *Let $\mathcal{P}$ be a PASTA program and (T, χ) be a tree decomposition of $P_\mathcal{P}$ of width k. For upper probability queries, we first add the probabilistic facts.*

Probabilistic Facts

$$p{::}a \qquad \text{for every } p{::}a \in \mathcal{P} \quad (9)$$

Then, we perform the following for every node t in T.

Rule Satisfiability (Depending on Probabilistic Facts)

$$(sat_t^J \mid J \cap pf(\mathcal{P}), \{\neg x \mid x \in pf(\mathcal{P}_t) \setminus J\}) \quad \text{for every } J \in 2^{\chi(t)}, J \models \mathcal{P}_t, J \cap h(\mathcal{P}_t) \subseteq pr(J, \mathcal{P}_t) \quad (10)$$

Propagation of Satisfiability

$$(sat_{\leq t}^J \mid sat_{<t}^J, sat_t^J) \qquad \text{for every } J \in 2^{\chi(t)} \quad (11)$$

$$(sat_{<t,t'}^J \mid sat_{\leq t'}^K) \qquad \text{for every } t' \in \mathrm{chld}(t), J \in 2^{\chi(t)}, K \in 2^{\chi(t')}, J \cap \chi(t') = K \cap \chi(t) \quad (12)$$

$$(sat_{<t}^J \mid \{sat_{<t,t'}^J : t' \in \mathrm{chld}(t)\}) \text{ for every } J \in 2^{\chi(t)} \quad (13)$$

By Equation (9), probabilistic facts are copied as is (this is done only once). Then, we use auxiliary atoms of the form sat_t^J for every tree decomposition node t in T and assignment $J \in 2^{\chi(t)}$. The idea of Equation (10) is to encode all possible parts of answer sets for every tree decomposition node into facts that are compatible with the probabilistic facts in the bag. Then, it remains to propagate this information upwards in the tree. Equation (11) propagates assignments J upwards, if satisfiability holds in t, i.e., sat_t^J, and if compatible assignments for J are satisfiable below (indicated by $sat_{<t}^J$). To determine that compatible assignments exist below, we determine by Equation (12) whether such a compatible assignment exists in a child node t' of t. Then, for J to be compatible with *all nodes below* t, such a compatible assignment has to exist for *every* child node below t, ensured by Equation (13).

Example 10. *Using Reduction 2 we can* answer upper probability queries *on q assuming a decomposition with bag $\{q\}$ of the root r for query atom q, by computing $\Sigma_{\langle\{sat_{\leq r}^{\{q\}}\},c\rangle \in \tau_r} c$.*

Theorem 4. *There is a translation from a head-cycle-free PASTA program $\mathcal{P}$ of treewidth k to a non-disjunctive, stratified tight PASTA program $\mathcal{P}'$ of treewidth $k' = 5 \cdot 2^{k+1}$.*

Proof. We take any tree decomposition $\mathcal{T} = (T, \chi)$ of $P_\mathcal{P}$ of width k and apply Equations (9)–(13) on $\mathcal{P}$, resulting in $\mathcal{P}'$. Observe that by construction the resulting program is non-disjunctive, tight, and stratified.

To visualize the claim on treewidth, we construct a tree decomposition $\mathcal{T}' = (T, \chi')$ of $P_{\mathcal{P}'}$ and relate bag sizes to those of $\mathcal{T}$. For every t in T, let $\chi'(t) = (pf(\mathcal{P}) \cap \chi(t)) \cup \{sat_t^J, sat_{\leq t}^J, sat_{<t}^J \mid J \in 2^{|\chi(t)|}\} \cup \{sat_{<t^*,t}^J, sat_{\leq t}^K \mid J \in 2^{|\chi(t^*)|}, K \in 2^{|\chi(t)|}, J \cap \chi(t) = K \cap \chi(t^*), t \in \mathrm{chld}(t^*)\}$. It is easy to see that $|\chi'(t)| \leq |\chi(t)| + 3 \cdot 2^{|\chi(t)|} + 2^{|\chi(t^*)|}$, where t^* is the unique parent node of t, i.e., $t \in \mathrm{chld}(t^*)$. Hence, the claimed treewidth bound holds as desired. □

For Lower Probability Queries we can still apply an analogous of Reduction 2. However, instead of variables of the form $sat_{\leq t}^J$ we use variables $Asat_{\leq t}^J$ to propagate that up to node t *every answer set* that is compatible with J exists.

Reduction 3. *For a PASTA program $\mathcal{P}$, a tree decomposition (T, χ) of $P_\mathcal{P}$, we first add Equation* (9) *and* (10)*. Then, for any node t of T, we construct the following.*

Propagation of All-Satisfiability

$$(Asat_{\leq t}^J \mid Asat_{<t}^J, sat_t^J) \qquad \text{for every } J \in 2^{\chi(t)} \quad (14)$$

$$(Asat_{<t,t'}^J \mid \{Asat_{\leq t'}^K \mid t' \in \mathrm{chld}(t), J \in 2^{\chi(t)}, K \in 2^{\chi(t')}, J \cap \chi(t') = K \cap \chi(t)\}) \quad (15)$$

$$(Asat_{<t}^J \mid \{Asat_{<t,t'}^J : t' \in \mathrm{chld}(t)\}) \text{ for every } J \in 2^{\chi(t)} \quad (16)$$

Consequently, for lower probability queries, instead of Equations (11)–(13) we use Equations (14)–(16). Observe that Equations (14) and (16) are almost identical to Equations (11) and (13), respectively. The only difference is in the variable names. Hence, the only substantial difference lies in Equation (15)). Indeed, Equation (15) ensures that we can only derive $Asat_{<t,t'}^J$ if all compatible answer set candidates for t' sustain, whereas in Equation (12) a single answer set candidate in t' is enough for $sat_{<t,t'}^J$.

Example 11. *With the same assumptions of Example 10 we can ask lower probability queries by computing $\Sigma_{\langle\{Asat_{\leq r}^{\{q\}}\},c\rangle \in \tau_r} c$ after applying Reduction 3.*

Note that we can apply both Reduction 2 and 3 and obtain both sat_t^q and $Asat_t^q$ in the root. Then, to ask for one of the two probability bounds we can ask for the corresponding atom. Furthermore, this approach can be straightforwardly extended to compute conditional probabilities.

4.4 Lower Bounds: We Cannot Do Much Better

We need to slightly adapt the proof of Theorem 1 in a way to show that structural dependency measures like treewidth are linearly preserved, which yields the lower bounds below. To this end, we assume the *exponential time hypothesis (ETH)* (Impagliazzo, Paturi, and Zane 2001), implying that SAT with n variables cannot be solved in time $2^{o(n)}$.

Theorem 5. *Let $\mathcal{P}$ be any PASTA program whose primal graph has treewidth k. Then, under ETH, probabilistic inference cannot be achieved in time $2^{2^{2^{o(k)}}} \cdot \mathrm{poly}(\mathcal{P})$.*

Proof. Note that QBFs of the form $\exists X.\forall Y.\exists Z.\varphi(X, Y, Z)$ precisely admit this runtime bound under ETH, if k is the treewidth of the primal graph P_φ, see, e.g., (Fichte, Hecher,

and Pfandler 2020). Consequently, since the QBF can be easily reduced to $\#_{\geq 1}X.\forall Y.\exists Z.\varphi(X,Y,Z)$, for the lower bound to sustain, it remains to show that the reduction of Theorem 1 linearly preserves the treewidth. To this end, let $\mathcal{T} = (T, \chi)$ be a tree decomposition of P_φ of treewidth k. From this, we construct a tree decomposition $\mathcal{T}' = (T, \chi')$ of $P_\mathcal{P}$ with a linear increase of width compared to $\mathcal{T}$. Without loss of generality, we assume that φ is in 3-CNF and that each node t in T gets assigned at most one clause c_t whose variables are in $\chi(t)$. This can be achieved via copying decomposition nodes. For every node t in T, let $\chi'(t) = \{sat, usat, \top\} \cup \{t(x), not\_t(x) \mid x \in \chi(t)\} \cup facts(c_t)$, where $facts$ gives the (up to three facts) over predicates pos_{c_t} and neg_{c_t}. Then, $|\chi'(t)|$ is at most $2|\chi(t)| + 6$. □

Analogously, it turns out that, under ETH, Algorithm 1 and the reductions of Sections 4.2 and 4.3 are asymptotically optimal. As above, we obtain the following lower bounds.

Corollary 5. *Let $\mathcal{P}$ be any non-disjunctive PASTA program whose primal graph has treewidth k. Under ETH, probabilistic inference cannot be achieved in time $2^{2^{o(k)}} \cdot \mathrm{poly}(\mathcal{P})$ (or $2^{o(k)} \cdot \mathrm{poly}(\mathcal{P})$ if $\mathcal{P}$ is also negation-free).*

5 Related Work

The complexity of PASP under the credal semantics has been extensively studied (Cozman and Mauá 2017; Mauá and Cozman 2020; Cozman and Mauá 2020). There are other semantics that were proposed for handling programs under the stable model semantics extended to manage uncertainty, such as $\mathrm{LP}^{\mathrm{MLN}}$ (Lee and Wang 2016), P-log (Baral, Gelfond, and Rushton 2009), smProbLog (Totis, De Raedt, and Kimmig 2023), and the L-credal semantics (Rocha and Cozman 2022; Mauá, Cozman, and Garces 2024). These differ on how they express and manage uncertainty. For example, $\mathrm{LP}^{\mathrm{MLN}}$ associates weights to atoms and rules, rather than probabilities, and does not consider worlds but only answer sets. smProbLog is a specialization of the credal semantics, since the probability of a world is uniformly distributed on its answer sets, while the credal semantics has no such assumption. That is, the contribution of a world w to the probability of a query q, is $P(w)$ divided by the number of answer sets in which q is present. Furthermore, worlds without answer sets are admitted, since a third truth value is considered. The L-credal semantics is an extension of the credal semantics, also capable of also handling inconsistent worlds by considering a third "undefined" truth value.

Several solvers exist to perform inference in PASP: PASTA (Azzolini, Bellodi, and Riguzzi 2022), aspmc (Eiter, Hecher, and Kiesel 2024), DPASP (Geh et al. 2024), and PASOCS (Tuckey, Russo, and Broda 2021). However, PASTA is the only tool that supports PASTA statements, and it is based on projected answer set enumeration (Gebser, Kaufmann, and Schaub 2009). Both DPASP and PASOCS are based on exhaustive enumeration. aspmc (Eiter, Hecher, and Kiesel 2024) is a tool that also adopts tree decompositions to guide knowledge compilation (KC, (Darwiche and Marquis 2002), largely adopted in probabilistic inference) to solve tasks that can be expressed within the second level algebraic model counting framework (Kiesel, Totis, and Kimmig 2022) (which is an extension of algebraic model counting (Kimmig, Van den Broeck, and De Raedt 2017) which, in turn, is a generalization of the well-known task of weighted model counting), such as inference in PASP (Azzolini and Riguzzi 2024). However, they restrict themselves to programs composed by normal rules only. Here, we go a step further since we consider PASTA programs, also composed of disjunctive rules and aggregates, and develop algorithms that work on the tree decomposition.

There are known complexity results for logic programming and treewidth (Jakl, Pichler, and Woltran 2009; Fichte et al. 2017; Hecher 2022). If we consider Probabilistic Logic Programming (PLP) (Riguzzi 2022), where each world has exactly one answer set, there are also multiple available languages. Most of them are based on the Distribution Semantics (Sato 1995), such as ProbLog (De Raedt, Kimmig, and Toivonen 2007) and Logic Programs with Annotated Disjunctions (LPADs) (Vennekens, Verbaeten, and Bruynooghe 2004). Note that these programs can be considered as a special case of PASP, so existing PLP solvers such as ProbLog (De Raedt, Kimmig, and Toivonen 2007), PITA (Riguzzi and Swift 2011), and schlandals (Dubray, Schaus, and Nijssen 2023) cannot be adopted here.

6 Conclusion and Future Work

In this paper, we studied the complexity of cautious reasoning and computing the most probable explanation in PASTA programs, i.e., programs composed by probabilistic facts and statistical statements only and interpreted under the credal semantics. According to the original proposal by Azzolini, Bellodi, and Riguzzi (2022), statistical statements are converted into disjunctive rules and constraints with aggregates. Being these seemingly restricted, in this paper, we precisely place them in the polynomial hierarchy, showing that inference belongs to the $PP^{\Sigma_2^P}$ class, which is the same class obtained in (Mauá and Cozman 2020) by considering propositional PASP with all constructs: this shows that, even with such simple programs, we retain a huge expressivity. Such a complexity requires also efficient algorithms. Thus, we assessed whether applying tree decomposition techniques can improve the efficiency of reasoning and provided a dynamic programming algorithm for inference obtaining precise upper bounds and matching lower bounds (under ETH). To the best of our knowledge, this is the first parameterized complexity study for probabilistic logic programming. As future work, we plan to extend the investigation to different semantics and different metrics and develop practical implementations. Furthermore, while in principle our (treewidth-based) algorithms carry over to PLP, specializing them for PLP inference is an exciting direction.

Acknowledgments

DA is a member of the Gruppo Nazionale Calcolo Scientifico – Istituto Nazionale di Alta Matematica (GNCS-INdAM). This research was funded in parts by the Austrian Science Fund (FWF), grant 10.55776/J4656.

References

Azzolini, D., and Riguzzi, F. 2023. Lifted inference for statistical statements in probabilistic answer set programming. *International Journal of Approximate Reasoning* 163:109040.

Azzolini, D., and Riguzzi, F. 2024. Inference in probabilistic answer set programs with imprecise probabilities via optimization. In *The 40th Conference on Uncertainty in Artificial Intelligence*.

Azzolini, D.; Bellodi, E.; and Riguzzi, F. 2022. Statistical statements in probabilistic logic programming. In Gottlob, G.; Inclezan, D.; and Maratea, M., eds., *Logic Programming and Nonmonotonic Reasoning*, 43–55. Cham: Springer International Publishing.

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

Bodlaender, H. L., and Koster, A. M. 2008. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal* 51(3):255–269.

Bodlaender, H. L. 1988a. Dynamic programming on graphs with bounded treewidth. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming (ICALP'88)*, volume 317 of *LNCS*, 105–118. Springer.

Bodlaender, H. L. 1988b. Dynamic programming on graphs with bounded treewidth. In Lepistö, T., and Salomaa, A., eds., *Automata, Languages and Programming*, 105–118. Berlin, Heidelberg: Springer Berlin Heidelberg.

Brewka, G.; Eiter, T.; and Truszczyński, M. 2011. Answer set programming at a glance. *Communications of the ACM* 54(12):92–103.

Cozman, F. G., and Mauá, D. D. 2017. On the semantics and complexity of probabilistic logic programs. *Journal of Artificial Intelligence Research* 60:221–262.

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.

Darwiche, A., and Marquis, P. 2002. A knowledge compilation map. *Journal of Artificial Intelligence Research* 17:229–264.

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.

Dubray, A.; Schaus, P.; and Nijssen, S. 2023. Probabilistic Inference by Projected Weighted Model Counting on Horn Clauses. In *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.* 15(3):289–323.

Eiter, T., and Gottlob, G. 1997. Expressiveness of stable model semantics for disjunctive logic programs with functions. *The Journal of Logic Programming* 33(2):167–178.

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

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

Faber, W.; Pfeifer, G.; and Leone, N. 2011. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence* 175(1):278–298.

Fages, F. 1994. Consistency of clark's completion and existence of stable models. *Methods of Logic in CS* 1:51–60.

Fichte, J. K.; Hecher, M.; Morak, M.; and Woltran, S. 2017. Answer set solving with bounded treewidth revisited. In Balduccini, M., and Janhunen, T., eds., *Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'17)*, volume 10377 of *LNCS*, 132–145. Espoo, Finland: Springer.

Fichte, J. K.; Hecher, M.; and Pfandler, A. 2020. Lower bounds for QBFs of bounded treewidth. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, 410–424.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2009. Solution enumeration for projected boolean search problems. In van Hoeve, W.-J., and Hooker, J. N., eds., *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 71–86. Berlin, Heidelberg: Springer Berlin Heidelberg.

Geh, R. L.; Gonçalves, J.; Silveira, I. C.; Mauá, D. D.; and Cozman, F. G. 2024. dPASP: a probabilistic logic programming environment for neurosymbolic learning and reasoning. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 21, 731–742.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *5th International Conference and Symposium on Logic Programming (ICLP/SLP 1988)*, volume 88, 1070–1080. USA: MIT Press.

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

Gill, J. 1977. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing* 6(4):675–695.

Halpern, J. Y. 1990. An analysis of first-order logics of probability. *Artificial Intelligence* 46(3):311–350.

Hecher, M., and Kiesel, R. 2024. On the structural hardness of answer set programming: Can structure efficiently confine the power of disjunctions? In Wooldridge, M. J.; Dy, J. G.; and Natarajan, S., eds., *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, 10535–10543. AAAI Press.

Hecher, M. 2022. Treewidth-aware reductions of normal ASP to SAT – is normal ASP harder than SAT after all? *Artificial Intelligence* 304:103651.

Impagliazzo, R.; Paturi, R.; and Zane, F. 2001. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.* 63(4):512–530.

Jakl, M.; Pichler, R.; and Woltran, S. 2009. Answer-set programming with bounded treewidth. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, volume 2, 816–822.

Kiesel, R.; Totis, P.; and Kimmig, A. 2022. Efficient knowledge compilation beyond weighted model counting. *Theory and Practice of Logic Programming* 22(4):505–522.

Kimmig, A.; Van den Broeck, G.; and De Raedt, L. 2017. Algebraic model counting. *Journal of Applied Logic* 22(C):46–62.

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.

Linke, T.; Tompits, H.; and Woltran, S. 2004. On acyclic and head-cycle free nested logic programs. In *International Conference on Logic Programming*, 225–239. Springer.

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

Mauá, D. D.; Cozman, F. G.; and Garces, A. 2024. Probabilistic logic programming under the L-stable semantics. In Gierasimczuk, N., and Heyninck, J., eds., *Proceedings of the 22nd International Workshop on Nonmonotonic Reasoning (NMR 2024)*, 24–33.

Papadimitriou, C. H. 1994. *Computational Complexity*. Addison-Wesley.

Riguzzi, F., and Swift, T. 2011. The PITA system: Tabling and answer subsumption for reasoning under uncertainty. *Theory and Practice of Logic Programming* 11(4-5):433–449.

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 *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning*, 309–319.

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, Tokyo, Japan, June 13-16, 1995*, 715–729. MIT Press.

Stockmeyer, L. J. 1976. The polynomial-time hierarchy. *Theoretical Computer Science* 3(1):1–22.

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.

Tuckey, D.; Russo, A.; and Broda, K. 2021. PASOCS: A parallel approximate solver for probabilistic logic programs under the credal semantics. *arXiv* abs/2105.10908.

Vennekens, J.; Verbaeten, S.; and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In Demoen, B., and Lifschitz, V., eds., *20th International Conference on Logic Programming (ICLP 2004)*, volume 3131 of *Lecture Notes in Computer Science*, 431–445. Berlin, Heidelberg: Springer.