Abduction with Probabilistic Logic Programming under the Distribution Semantics

Damiano Azzolini[a,*], Elena Bellodi[b], Stefano Ferilli[c], Fabrizio Riguzzi[a], Riccardo Zese[d]

[a] *Dipartimento di Matematica e Informatica – Università di Ferrara*
Via Saragat 1, 44122, Ferrara, Italy
[b] *Dipartimento di Ingegneria – Università di Ferrara*
Via Saragat 1, 44122, Ferrara, Italy
[c] *Dipartimento di Informatica – Università degli Studi di Bari*
Via Orabona 4, 70125, Bari, Italy
[d] *Dipartimento di Scienze Chimiche, Farmaceutiche ed Agrarie – Università di Ferrara*
Via Luigi Borsari 46, 44121, Ferrara, Italy

Abstract

In Probabilistic Abductive Logic Programming we are given a probabilistic logic program, a set of abducible facts, and a set of constraints. Inference in probabilistic abductive logic programs aims to find a subset of the abducible facts that is compatible with the constraints and that maximizes the joint probability of the query and the constraints. In this paper, we extend the PITA reasoner with an algorithm to perform abduction on probabilistic abductive logic programs exploiting Binary Decision Diagrams. Tests on several synthetic datasets show the effectiveness of our approach.

Keywords: Abduction, Distribution Semantics, Probabilistic Logic Programming, Statistical Relational Artificial Intelligence

1. Introduction

Probabilistic Logic Programming (PLP) [1, 2] has recently attracted a lot of interest thanks to its ability to represent several scenarios [3, 4] with a simple

*Corresponding author

Email addresses: damiano.azzolini@unife.it (Damiano Azzolini), elena.bellodi@unife.it (Elena Bellodi), stefano.ferilli@uniba.it (Stefano Ferilli), fabrizio.riguzzi@unife.it (Fabrizio Riguzzi), riccardo.zese@unife.it (Riccardo Zese)

yet powerful language. Furthermore, the possibility of integrating it with sub-symbolic systems makes it a relevant component of explainable probabilistic models [5].

An extension of Logic Programming that can manage incompleteness in the data is given by Abductive Logic Programming (ALP) [6, 7]. The goal of abduction is to find, given a set of hypotheses called *abducibles*, a subset of these that explains an observed fact. With ALP, users can perform logical abduction from an expressive logic model possibly subject to constraints. However, a limitation is that observations are often noisy since they come from real-world data. Furthermore, there may be different levels of belief among rules. It is thus fundamental to extend ALP and associate probabilities to observations, to both handle these scenarios and test the reliability of the computed solutions.

Starting from the probabilistic logic language of LPADs, in this paper we introduce *probabilistic abductive logic programs* (PALP), i.e., probabilistic logic programs including a set of *abducible* facts and a (possibly empty) set of (possibly probabilistic) *integrity constraints*. Probabilities associated with integrity constraints can represent how strong the belief is that the constraint is true and can help in defining a more articulated probability distribution of queries. These programs define a probability distribution over abductive logic programs inspired by the distribution semantics in PLP [8]. *Given a query, the goal is to maximize the joint probability distribution of the query and the constraints by selecting the minimal subsets of abducible facts to be included in the abductive logic program while ensuring that constraints are not violated.*

Consider the following motivating example: suppose you work in the city center and, starting from your home, you may choose several alternative routes to reach your office. However, streets are often congested, but you want to avoid traffic and reach the destination with the lowest probability of encountering a car accident. You can associate different probabilities (representing beliefs or noisy data that came from historical measurements) of encountering (or not encountering) a car accident in all the possible alternative streets, and impose an integrity constraint that states that only one path (combination of streets) can be selected (clearly, you cannot travel two routes simultaneously). Then, you look for the best combination of streets to maximize the probability of not encountering a car accident. A possible encoding for this situation is presented in Section 6 (experiments on graph datasets). Alternatively, suppose that you want to study more in depth a natural phenomenon that may happen in a region. In the model, there may be some variables that describe the land morphological characteristics and some variables that relate the possible events that can occur, such as eruption or earthquake. Moreover, you want to impose that some of these cannot be observed together (or it is unlikely that they will be). The goal may consist in finding the optimal combination of variables (representing possible events) that better describes a possible scenario and maximizes its probability. This will be the running example we use through the paper, starting from Example 1, where we model events possibly occurring in the island of Stromboli.

To perform inference on PALP, we extend the PITA system [9], which computes the probability of a query from an LPAD by means of Binary Decision Diagrams (BDD). One of the key points of this extension is that it has the version of PITA used to make inference on LPADs as a special case: when both the set of abducibles and the set of constraints are empty, the program is treated as a probabilistic logic program. This has an important implication: we do not need to write an *ad hoc* algorithm to treat the probabilistic part of the LPAD, we just need to extend the already-existing algorithm. Furthermore, (probabilistic) integrity constraints are implemented by means of operations on BDDs and so they can be directly incorporated in the representation. The extended system has been integrated into the web application "cplint on SWISH" [10, 11], available online at `https://cplint.eu/`.

To test our implementation, we performed several experiments on five synthetic datasets. The results show that PALP with probabilistic or deterministic integrity constraints often require comparable inference time. Moreover, through a series of examples, we compare inference on PALP with other related tasks, such as Maximum a Posteriori (MAP), Most Probable Explanation

(MPE), and Viterbi proof.

The paper is structured as follows: Section 2 and Section 3 present respectively an overview of Abductive and Probabilistic Logic Programming. Section 4 introduces probabilistic abductive logic programs and some illustrative examples. Section 5 describes the inference algorithm we developed, which was tested on several datasets in Section 6. Section 7 provides an analysis of related works, and Section 8 concludes the paper.

2. Abductive Logic Programming and Well-Founded Semantics

Abduction is the inference strategy that copes with incompleteness in the data by guessing information that was not observed. Abductive Logic Programming [6, 7] extends Logic Programming [12] by considering some atoms, called *abducibles*, to be only indirectly and partially defined using a set of *constraints*. The reasoner may derive *abductive hypotheses*, i.e., sets of abducible atoms, as long as such hypotheses do not violate the given constraints. Let us now introduce more formally some definitions.

Definition 1 (Integrity Constraint). *A (deterministic) integrity constraint* IC *is a formula of the form*

$$:-\ Body$$

where $Body = b_1, \ldots, b_n$ *and each* b_i *is a logical literal (i.e., a logical atom or the negation of a logical atom). Logically, an* IC *can be understood for the logical formula*

$$false \leftarrow \exists Body$$

where $\exists$ *is over all variables in Body.*

Definition 2 (Abductive Logic Program). *An abductive logic program is a triple* $(P, \mathcal{IC}, A)$ *where* P *is a normal program,* $\mathcal{IC}$ *is a set of integrity constraints and* A *is a set of ground atoms, the* abducibles*, that do not appear in the head of a rule of any grounding of* P.

Before introducing the definition of abductive explanation, we review the basic concepts regarding the Well-founded semantics (WFS) [13]. Following [14], a 3-valued interpretation I for a logic program P is a pair $I = \langle T, F \rangle$ where T and F contain respectively the true and false ground atoms in I, and both are subsets of the Herbrand base H_P of P. The truth value of the atoms in $H_P \setminus (T \cup F)$ is undefined. A 3-valued interpretation is *consistent* if $T \cap F = \emptyset$. If $H_P = T \cup F$ for a 3-valued interpretation I of P, I is called 2-valued. A consistent 3-valued interpretation M is a 3-valued *model* of P if, for every clause C in P, the clause is true in M. If M is 2-valued, it is called a 2-valued model. The WFS assigns a meaning to logic programs through a 3-valued interpretation. We consider here the definition provided in [14] which is based on an iterated fixpoint. Given a program P and an interpretation $I = \langle T, F \rangle$, we define with $\mathcal{T}_I(T)$ and $\mathcal{F}_I(F)$ the operators containing new true and false facts that can be derived from P knowing I. Both are monotonic [14], so they have a least and greatest fixpoint. Call T_I the least fixpoint of $\mathcal{T}_I$ and F_I the greatest fixpoint of $\mathcal{F}_I$. Consider this new operator $\mathcal{I}(I) = I \cup \langle T_I, F_I \rangle$ that assigns to every interpretation I of P a new interpretation $\mathcal{I}(I)$. $\mathcal{I}$ is also monotonic [14]. Its least fixpoint is considered the well-founded model (WFM) of P, denoted as $\mathit{WFM}(P)$. Undefined atoms are not added to $\mathcal{I}$ in none of its iterations. If the set of undefined atoms of $\mathit{WFM}(P)$ is empty, the WFM is called total or 2-valued, and the program is *dynamic stratified*.

Definition 3 (Abductive Explanation). *Given an abductive logic program* (P, IC, A) *and a conjunction of ground atoms* q*, the* query*, the problem of abduction is to find a set of atoms* $\Delta \subseteq A$*, called* abductive explanation*, such that* $P \cup \Delta \models q$ *and no constraints are violated, i.e.,* $P \cup \Delta \not\models \exists Body$ *for every integrity constraint, where* $\models$ *is to be interpreted as truth in the well-founded model (WFM) of the program [15]*[1].

Here, we require that $P \cup \Delta$ has a 2-valued WFM for every Δ. Consequently,

[1]We consider Δ a set of facts rather than a set of atoms when we add it to P.

negation is defined under the WFM. This also means that $\models$ is well-defined and it is either true or false for any $P \cup \Delta$ and any q. By default, abducible facts not present in the explanation are considered false.

Consider the following example:

```
fire :- spark, not wet.
spark :- lighter.
spark :- flint.
wet:- grass_is_wet.

:- not wet, lighter.
```

where `grass_is_wet`, `lighter`, and `flint` are abducibles and `not` means negation. The query `fire` has the abductive explanation $\Delta_1 = \{\texttt{flint}\}$. Note that also $\Delta_2 = \{\texttt{lighter}\}$ could be an abductive explanation, but it is forbidden by the IC.

3. Probabilistic Logic Programming

The distribution semantics [8] is becoming increasingly important in Probabilistic Logic Programming. According to this semantics, a probabilistic logic program defines a probability distribution over a set of normal logic programs (called *worlds*). The distribution is extended to a joint distribution over worlds and a ground query, and the probability that the query is true is obtained from this distribution by marginalization. The languages based on the distribution semantics differ in the way they define the distribution over Logic Programs. Here, we consider *Logic Programs with Annotated Disjunctions* (LPADs) [16], which are sets of disjunctive clauses in which each atom in the head is annotated with a probability (see Section 7 for a discussion of related proposals).

Formally, a Logic Program with Annotated Disjunctions (LPAD) consists of a finite set of annotated disjunctive clauses. An annotated disjunctive clause C_i is of the form

$$h_{i1} : \Pi_{i1}; \ldots ; h_{in_i} : \Pi_{in_i} :- b_{i1}, \ldots , b_{im_i}.$$

In such a clause, the semicolon stands for disjunction, $h_{i1}, \ldots h_{in_i}$ are logical atoms, $b_{i1}, \ldots, b_{im_i}$ are logical literals, and $\Pi_{i1}, \ldots, \Pi_{in_i}$ are real numbers in the interval $]0, 1]$ such that $\sum_{k=1}^{n_i} \Pi_{ik} \leq 1$. Note that, if $n_i = 1$ and $\Pi_{i1} = 1$, the clause corresponds to a non-disjunctive clause. If $\sum_{k=1}^{n_i} \Pi_{ik} < 1$, the head of the annotated disjunctive clause implicitly contains an extra atom *null* that does not appear in the body of any clause and whose annotation is $1 - \sum_{k=1}^{n_i} \Pi_{ik}$, with the meaning that none of the previous h_i is true. Probabilistic facts are considered as independent: this may seem restrictive but, in practice, does not limit the possibility to express dependence between facts [2, 17]. We denote by $\mathit{ground}(T)$ the grounding of an LPAD T, i.e., the result of replacing variables with constants in T.

Example 1. *The island of Stromboli is located at the intersection of two geological faults, one in the southwest-northeast direction, the other in the east-west direction, and contains one of the three volcanoes that are active in Italy. This program ([18, 19]) models the possibility that an eruption or an earthquake occurs at Stromboli.*

```
(C1) eruption:0.6;earthquake:0.3 :- sudden_er, fault_rupture(X).
(C2) sudden_er:0.7.
(C3) fault_rupture(southwest_northeast).
(C4) fault_rupture(east_west).
```

If there is a sudden energy release (`sudden_er`*) under the island and there is a fault rupture (*`fault_rupture(X)`*), then there can be an eruption of the volcano on the island with probability 0.6 or an earthquake in the area with probability 0.3. The energy release occurs with probability 0.7 and we are sure that ruptures occur along both faults.*

We now present the distribution semantics for programs without function symbols, so with a finite Herbrand base. For the distribution semantics with function symbols see [8, 20, 21, 22].

An *atomic choice* is a selection of the k-th head atom for a grounding $C_i\theta_j$ of a probabilistic clause C_i and is represented by the triple (C_i, θ_j, k), where θ_j

is a grounding substitution (a set of couples $Var/constant$ grounding C_i) and $k \in \{1, \ldots, n_i\}$. An atomic choice represents an equation of the form $X_{ij} = k$ where X_{ij} is a random variable associated with $C_i\theta_j$. A set of atomic choices κ is *consistent* if $(C_i, \theta_j, k) \in \kappa, (C_i, \theta_j, m) \in \kappa$ implies that $k = m$, i.e., only one head is selected for a ground clause.

A *composite choice* κ is a consistent set of atomic choices. The probability of a composite choice κ is $P(\kappa) = \prod_{(C_i,\theta_j,k)\in\kappa} \Pi_{ik}$. A *selection* σ is a total composite choice (it contains one atomic choice for every grounding of each probabilistic clause). Let us call S_T the set of all selections. A selection σ identifies a logic program w_σ called a *world*. The probability of w_σ is $P(w_\sigma) = P(\sigma) = \prod_{(C_i,\theta_j,k)\in\sigma} \Pi_{ik}$. Since the program does not contain function symbols, the set of worlds $W_T = \{w_1, \ldots, w_m\}$ is finite and $P(w)$ is a distribution over worlds: $\sum_{w\in W_T} P(w) = 1$. We consider only sound LPADs as defined below.

Definition 4. *An LPAD T is called sound iff for each selection σ in S_T, the program w_σ chosen by σ is 2-valued.*

The conditional probability of a query q (ground atom) given a world w can be defined as: $P(q \mid w) = 1$ if q is true in the WFM of w ($w \models q$) and 0 otherwise. We can obtain the probability of the query by marginalization:

$$P(q) = \sum_{w} P(q, w) = \sum_{w} P(q \mid w)P(w) = \sum_{w \models q} P(w). \tag{1}$$

Formula 1 can be also used to compute the probability of a query when q is composed of a conjunction of ground atoms, since the truth of a conjunction of ground atoms is still well defined in a world.

Example 2. *For the LPAD T of Example 1, clause C_1 has two groundings, $C_1\theta_1$ with $\theta_1 = \{X/southwest\_northeast\}$ and $C_1\theta_2$ with $\theta_2 = \{X/east\_west\}$, while clause C_2 has a single grounding $C_2\emptyset$. Since C_1 has three head atoms and C_2 two, T has $3 \times 3 \times 2 = 18$ worlds, shown in Table 1. The query* `eruption` *is true in 5 of them (highlighted in the table) and its probability is $P(eruption) = 0.6 \cdot 0.6 \cdot 0.7 + 0.6 \cdot 0.3 \cdot 0.7 + 0.6 \cdot 0.1 \cdot 0.7 + 0.3 \cdot 0.6 \cdot 0.7 + 0.1 \cdot 0.6 \cdot 0.7 = 0.588$.*

w	eruption:0.6; earthquake:0.3 :- sudden_er, fault_rupture(sw_ne).	eruption:0.6; earthquake:0.3 :- sudden_er, fault_rupture(east_west).	sudden_er:0.7.	$P(w)$
1	eruption	eruption	sudden_er	0.252
2	eruption	earthquake	sudden_er	0.126
3	eruption	*null*	sudden_er	0.042
4	eruption	eruption	*null*	0.108
5	eruption	earthquake	*null*	0.054
6	eruption	*null*	*null*	0.018
7	earthquake	eruption	sudden_er	0.126
8	earthquake	earthquake	sudden_er	0.063
9	earthquake	*null*	sudden_er	0.021
10	earthquake	eruption	*null*	0.054
11	earthquake	earthquake	*null*	0.027
12	earthquake	*null*	*null*	0.009
13	*null*	eruption	sudden_er	0.042
14	*null*	earthquake	sudden_er	0.021
15	*null*	*null*	sudden_er	0.007
16	*null*	eruption	*null*	0.018
17	*null*	earthquake	*null*	0.009
18	*null*	*null*	*null*	0.003

Table 1: Possible worlds w for the LPAD of Example 1 with the corresponding probability $P(w)$, computed as the product of the probabilities associated with the head atoms taking value true, reported in each row. Highlighted rows represent the worlds in which the query `eruption` is true.

A composite choice κ *identifies* a set ω_κ that contains all the worlds associated with a selection that is a superset of κ: i.e., $\omega_\kappa = \{w_\sigma \mid \sigma \in S_T, \sigma \supseteq \kappa\}$. We define the set of worlds *identified* by a set of composite choices K as $\omega_K = \bigcup_{\kappa \in K} \omega_\kappa$. Given a ground atom q, a composite choice κ is an *explanation* (not to be confused with an *abductive* explanation, that will be defined later) for q if q is true in every world of ω_κ. For example, the composite choice $\kappa_1 = \{(C_1, \{X/southwest\_northeast\}, 1), (C_2, \emptyset, 1)\}$ is an explanation for *eruption* in Example 1. A set of composite choices K is *covering* with respect to q if every world w_σ in which q is true is such that $w_\sigma \in \omega_K$. In Example 1, a covering set of explanations for *eruption* is $K = \{\kappa_1, \kappa_2\}$ where:

$$\kappa_1 = \{(C_1, \{X/southwest\_northeast\}, 1), (C_2, \emptyset, 1)\} \quad (2)$$

$$\kappa_2 = \{(C_1, \{X/east\_west\}, 1), (C_2, \emptyset, 1)\} \quad (3)$$

Given a covering set of explanations for a query, we can obtain a Boolean formula in Disjunctive Normal Form (DNF) where: (1) we replace each atomic choice of the form (C_i, θ_j, k) with the equation $X_{ij} = k$, (2) we replace an explanation with the conjunction of the equations of its atomic choices, and (3) we represent the set of explanations with the disjunction of the formulas for all explanations. If we consider a world as the specification of a truth value for each equation $X_{ij} = k$, the formula evaluates to true exactly on the worlds where the query is true [20]. In Example 1, if we associate the variable X_{11} with $C_1\{X/southwest\_northeast\}$, X_{12} with $C_1\{X/east\_west\}$ and X_{21} with $C_2\emptyset$, the query is true if the following Boolean formula is true:

$$f(\mathbf{X}) = (X_{21} = 1 \wedge X_{11} = 1) \vee (X_{21} = 1 \wedge X_{12} = 1). \quad (4)$$

Since the disjuncts in the formula are not necessarily mutually exclusive, the probability of the query cannot be computed by a summation as in Formula (1). The problem of computing the probability of a Boolean formula in DNF, known as *disjoint sum*, is #P-complete [23]. One of the most effective ways of solving the problem makes use of Decision Diagrams.

3.1. Binary and Multi-valued Decision Diagrams

We can apply *knowledge compilation* [24] to the Boolean formula $f(\mathbf{X})$ to translate it into a "target language" that allows the computation of its probability in polynomial time. We can use Decision Diagrams as a target language. Since the random variables appearing in the Boolean formula that are associated with atomic choices can take on multiple values, we need to use Multi-valued Decision Diagrams (MDDs) [25]. An MDD represents a function $f(\mathbf{X})$ taking Boolean values on a set of multi-valued variables $\mathbf{X}$ by means of a rooted graph that has one level for each variable. Each node n has one child for each possible value of the multi-valued variable associated with n. The leaves store either 0 or 1. Since MDDs split paths on the basis of the values of a variable, the branches are mutually exclusive so a dynamic programming algorithm [26] can be applied for computing the probability. Figure 1a shows the MDD corresponding to Formula (4).

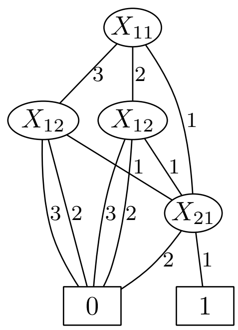
Most packages for the manipulation of decision diagrams are however restricted to work on Binary Decision Diagrams (BDD), i.e., decision diagrams where all the variables are Boolean. These packages offer Boolean operators among BDDs and apply simplification rules to the results of operations to reduce as much as possible the size of the binary decision diagram, obtaining a reduced BDD.

A node n in a BDD has two children: the 1-child and the 0-child. When drawing BDDs, rather than using edge labels, the 0-branch, the one going to the 0-child, is distinguished from the 1-branch by drawing it with a dashed line.

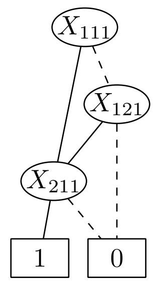
To work on Multi-valued Decision Diagrams with a BDD package we must represent multi-valued variables by means of binary variables. The following encoding used in [27] gives good performance. For a multi-valued variable X_{ij}, corresponding to a ground clause $C_i\theta_j$, having n_i values, we use $n_i - 1$ Boolean variables $X_{ij1}, \ldots, X_{ijn_i-1}$ and we represent the equation $X_{ij} = k$ for $k = 1, \ldots n_i - 1$ by means of the conjunction $\overline{X_{ij1}} \wedge \ldots \wedge \overline{X_{ijk-1}} \wedge X_{ijk}$, and the equation $X_{ij} = n_i$ by means of the conjunction $\overline{X_{ij1}} \wedge \ldots \wedge \overline{X_{ijn_i-1}}$. The BDD equivalent to the MDD of Figure 1a is shown in Figure 1b. Binary Decision Diagrams obtained in this way can be used as well for computing the probability of queries by associating a parameter π_{ik} with each Boolean variable X_{ijk}, representing $P(X_{ijk} = 1)$. The parameters are obtained from those of multi-valued variables in this way: $\pi_{i1} = \Pi_{i1}, \ldots, \pi_{ik} = \frac{\Pi_{ik}}{\prod_{j=1}^{k-1}(1-\pi_{ij})}$, up to $k = n_i - 1$.

To manage Binary Decision Diagrams, we exploit the CUDD[2] (Colorado University Decision Diagram) library, a library written in C that provides functions to manipulate different types of Decision Diagrams. CUDD allows the

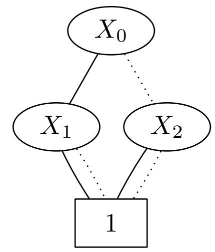
[2] `https://github.com/ivmai/cudd`



(a) Multi-valued Decision Diagram corresponding to Formula (4).



(b) Binary Decision Diagram (after simplification operations) equivalent to the MDD shown in Figure 1a.



(c) BDD with complemented edges.

Figure 1: Decision Diagrams.

definition of three types of edges: edge to a 1-child, edge to a 0-child, and *complemented* edge to a 0-child. The meaning of a complemented edge is that the function represented by the child must be complemented: if the leaf value is 1 and we visited an odd number of complemented edges along the path, then the value 0 must be considered. With this representation, only the 1-leaf is needed. An example of a BDD with complemented edges can be found in Figure 1c: it encodes the function $(X_0 \wedge X_1) \vee (\overline{X_0} \wedge \overline{X_2})$.

4. Probabilistic Abductive Logic Programs

To introduce the concept of probabilistic abductive logic programs, consider again Example 1. Suppose we want to maximize the probability of the query `eruption`. However, we do not know whether there was a fault rupture in the southwest-northeast or east-west direction. Furthermore, suppose that the fault rupture may happen along only one of the two directions simultaneously. In the following, we formally introduce this problem.

Definition 5 (Probabilistic Integrity Constraint). *A* probabilistic *integrity constraint is an integrity constraint with an associated probability, i.e., is a for-*

mula of the form

$$\pi :- \mathit{Body}$$

where $\mathit{Body} = b_1, \ldots, b_n$ *and each* b_i *is a logical literal (i.e., a logical atom or the negation of a logical atom), and* $\pi \in]0, 1]$.

Definition 6 (Probabilistic Abductive Logic Program). *A probabilistic abductive logic program is a triple* $(T, \mathcal{IC}, A)$ *where* T *is an LPAD,* $\mathcal{IC}$ *is a (possibly empty) set of (possibly probabilistic) integrity constraints, and* A *is a set of ground atoms, the* abducibles*, that do not appear in the head of a rule of any grounding of* T.

According to Definition 6, in general, a probabilistic abductive logic program is composed of an LPAD, a set of integrity constraints (probabilistic, deterministic, or both), and a set of abducibles, which we indicate by prepending the functor `abducible` to the atoms. The set of integrity constraints may be empty.

The triple $(T, \mathcal{IC}, A)$ defines a distribution over abductive logic programs P in this way: we obtain a world w by selecting one head atom for each grounding of each probabilistic clause from the LPAD T and then by adding or not each grounding of each probabilistic integrity constraint from $\mathcal{IC}$. The probability of the world is given by the product among the probabilities of the atomic choices made for the LPAD clauses, a factor π for each grounding of each probabilistic integrity constraint $\pi : -\mathit{Body}$ inserted in the world, and a factor $1 - \pi$ for each constraint not included in the world. For example, in the program shown in Figure 2a, the two probabilistic facts (`b` and `d`) and the IC offer two alternatives each. There are $2 \times 2 \times 2 = 8$ worlds, whose probabilities are computed as shown in Figure 2b.

Given a probabilistic abductive logic program $(T, \mathcal{IC}, A)$ and a set of ground atoms $\Delta \subseteq A$, the joint probability $P(q, \mathcal{IC} \mid \Delta)$ of a query q and the integrity constraints in $\mathcal{IC}$ to be true in $(T, \mathcal{IC}, A)$ given Δ is the sum of the probabilities of the worlds where Δ is an abductive explanation of q and all constraints are satisfied. $P(q, \mathcal{IC} \mid \Delta)$ can be computed by marginalizing the joint probability

```
a:- b,c.
a:- d,e.

b:0.3.
abducible c.
d:0.6.
abducible e.
0.1 :- c,e.
```

(a) Program.

w	b	d	:- c,e.	$P(w)$
1	T	T	I	$0.3 \cdot 0.6 \cdot 0.1 = 0.018$
2	T	F	I	$0.3 \cdot 0.4 \cdot 0.1 = 0.012$
3	F	T	I	$0.7 \cdot 0.6 \cdot 0.1 = 0.042$
4	F	F	I	$0.7 \cdot 0.4 \cdot 0.1 = 0.028$
5	T	T	E	$0.3 \cdot 0.6 \cdot 0.9 = 0.162$
6	T	F	E	$0.3 \cdot 0.4 \cdot 0.9 = 0.108$
7	F	T	E	$0.7 \cdot 0.6 \cdot 0.9 = 0.378$
8	F	F	E	$0.7 \cdot 0.4 \cdot 0.9 = 0.252$

(b) Worlds.

Figure 2: Example program and its worlds. I and E indicate respectively whether the IC is included (I) or not (E) in each world.

of the worlds, the query, and the ICs, in this way:

$$P(q, \mathcal{IC} \mid \Delta) = \sum_w P(q, \mathcal{IC}, w \mid \Delta) = \sum_w P(q, \mathcal{IC} \mid w, \Delta) \cdot P(w \mid \Delta).$$

If we indicate respectively with P_w the abductive logic program and with IC_w the subset of integrity constraints considered in every world w, then

$$P(q, \mathcal{IC} \mid w, \Delta) = \begin{cases} 1 & \text{if } P_w \cup \Delta \models q \text{ and } P_w \cup \Delta \not\models IC_w \\ 0 & \text{otherwise} \end{cases}$$

so

$$P(q, \mathcal{IC} \mid \Delta) = \sum_{w: P_w \cup \Delta \models q \wedge P_w \cup \Delta \not\models IC_w} P(w \mid \Delta).$$

Definition 7 (Probabilistic Abductive Problem). *Given a probabilistic abductive logic program $(T, \mathcal{IC}, A)$ and a conjunction of ground atoms q, the* query, *the probabilistic abductive problem consists in finding a set $\Delta \subseteq A$, the* probabilistic abductive explanation, *such that $P(q, \mathcal{IC} \mid \Delta)$ is maximized and the explanations in Δ are minimal, i.e., solve*

$$\text{least}(\arg\max_\Delta P(q, \mathcal{IC} \mid \Delta))$$

where $\arg\max$ *returns the set of all sets of abducibles that maximizes the joint probability of the query and the ICs (there can be more than one set of abducibles if they all induce the same probability), and*

$$\text{least}(I) = \{\Delta \mid \Delta \in I, \nexists \Delta' \in I : \Delta' \subset \Delta\}.$$

That is, the goal is to find the minimal sets Δ *of abducibles that maximize the joint probability of the query and the integrity constraints. Here, minimality is intended in terms of set inclusion. We also say that the function* least *computes the set of undominated* Δ*, where* Δ *dominates* Δ' *if* $\Delta \subset \Delta'$*. If* $\mathcal{IC} = \emptyset$*, the task reduces to* $\text{least}(\arg\max_\Delta P(q \mid \Delta))$*.*

Let us now clarify all the previously introduced concepts through a series of examples.

Example 3. *Consider the program shown in Figure 2a. The query* $q =$ `a` *has the probabilistic abductive explanation* $\Delta =$ `{c,e}`*. In fact,* $P(q, \mathcal{IC} \mid \Delta) = 0.162 + 0.108 + 0.378 = 0.648$*, corresponding to worlds #5,6,7 of Figure 2b, where* q *is true given* Δ *and the IC is excluded (E) from the worlds. This happens because the IC does not completely exclude* `{c,e}`*, it just excludes it for the worlds where the constraint is present. The probability of such explanation is higher than the one associated to* `{e}` *and* `{c}`*, as:*

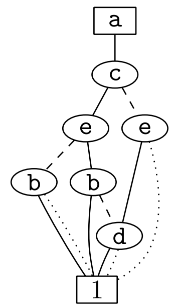
- *given the probabilistic abductive explanation* `{c}`*,* `a` *is true in 4 worlds (#1,2,5,6) with probability* $0.018 + 0.012 + 0.162 + 0.108 = 0.3$*;*
- *given the probabilistic abductive explanation* `{e}`*,* `a` *is true in 4 worlds (#1,3,5,7) with probability* $0.018 + 0.042 + 0.162 + 0.378 = 0.6$*.*

Variant 1. If we remove the integrity constraint from the program shown in Figure 2a, as reported in Figure 3a, the query $q =$ `a` *with the probabilistic abductive explanation* $\Delta =$ `{c,e}` *is true in the first three worlds, highlighted in Figure 3c, so it has probability* $P(q \mid \Delta) = 0.18 + 0.12 + 0.42 = 0.72$*.*

```
a:- b,c.
a:- d,e.

b:0.3.
abducible c.
d:0.6.
abducible e.
```

(a) Program.



(b) BDD.

w	b	d	$P(w)$
1	T	T	$0.3 \cdot 0.6 = 0.18$
2	T	F	$0.3 \cdot 0.4 = 0.12$
3	F	T	$0.7 \cdot 0.6 = 0.42$
4	F	F	$0.7 \cdot 0.4 = 0.28$

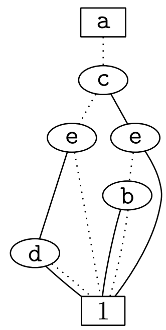
(c) Worlds.

Figure 3: Program, BDD, and worlds for Example 3 variant 1. Highlighted rows in the table represent the worlds in which the query `a` is true with probabilistic abductive explanation `{c,e}`, together with their probability.

Variant 2. Consider again the program shown in Figure 2a, but with the integrity constraint deterministic, i.e., `:- c,e`*. There are four possible worlds (see Figure 4b). The probabilistic abductive explanation that maximizes the probability of the query $q =$* `a` *and, at the same time, satisfies the constraint, is $\Delta =$* `{e}`*. $P(q, \mathcal{IC} \mid \Delta) = 0.18 + 0.42 = 0.6$, corresponding to the sum of the probabilities of the worlds where q is true given Δ, highlighted in Figure 4b. Note that the probabilistic abductive explanation* `{c,e}` *has higher probability than* `{e}` *(see above) but is forbidden by the IC.*

Variant 3. If the probability of the IC is set to 0.5, i.e., `0.5 :- c,e`*, the query $q =$* `a` *has the probabilistic abductive explanation $\Delta =$* `{e}`*, with probability $P(q, \mathcal{IC} \mid \Delta) = 0.09 \cdot 2 + 0.21 \cdot 2 = 0.6$, corresponding to worlds #1,3,5,7 (highlighted in Table 2). Such explanation gives higher probability than* `{c,e}` *and* `{c}` *as:*

- *given the probabilistic abductive explanation* `{c,e}`*,* `a` *is true in 3 worlds (#5,6,7) with probability $0.09 + 0.06 + 0.21 = 0.36$;*
- *given the probabilistic abductive explanation* `{c}`*,* `a` *is true in 4 worlds (#1,2,5,6) with probability $0.09 + 0.06 + 0.09 + 0.06 = 0.3$.*



(a) BDD.

w	b	d	$P(w)$
1	T	T	$0.3 \cdot 0.6 = 0.18$
2	T	F	$0.3 \cdot 0.4 = 0.12$
3	F	T	$0.7 \cdot 0.6 = 0.42$
4	F	F	$0.7 \cdot 0.4 = 0.28$

(b) Worlds.

Figure 4: BDD and worlds for the query of Example 3 variant 2. Highlighted rows in the table represent the worlds in which the query `a` is true with probabilistic abductive explanation `{e}`, together with their probability.

If we want to compute the minimum probability π of the IC π`:-c,e.` such that explanation `{e}` is chosen, we have to solve a system of two inequalities, imposing that the sum of the probabilities of worlds #5,6,7 (see Figure 2b) is greater than the sum of the probabilities associated both with worlds #1,2,5,6 and #1,3,5,7, with π as a variable. The result is $\pi < 0.167$. So, when the IC has probability greater than 0.167, explanation `{e}` is preferred to `{c,e}` (and `{c}`), as if the constraint were deterministic.

Example 4. *Abducibles facts can also be negated in the body of clauses. Consider a simple variation of the program shown in Figure 3a, where the abducible* `c` *appears negated in the first clause for* `a/0`*:*

```
a:- b,\+c.
```

Here, the query q = `a` *has the probabilistic abductive explanation Δ =* `{e}` *and probability 0.72, because, when* `c` *is not selected, the second clause still has the body satisfied.*

Example 5. *A program may have multiple minimal explanations yielding maximum probability of the query and the constraints. Consider the following example:*

w	`b`	`d`	`:- c,e.`	$P(w)$
1	T	T	I	$0.3 \cdot 0.6 \cdot 0.5 = 0.09$
2	T	F	I	$0.3 \cdot 0.4 \cdot 0.5 = 0.06$
3	F	T	I	$0.7 \cdot 0.6 \cdot 0.5 = 0.21$
4	F	F	I	$0.7 \cdot 0.4 \cdot 0.5 = 0.14$
5	T	T	E	$0.3 \cdot 0.6 \cdot 0.5 = 0.09$
6	T	F	E	$0.3 \cdot 0.4 \cdot 0.5 = 0.06$
7	F	T	E	$0.7 \cdot 0.6 \cdot 0.5 = 0.21$
8	F	F	E	$0.7 \cdot 0.4 \cdot 0.5 = 0.14$

Table 2: Worlds for Example 3 variant 3. Highlighted rows represent the worlds in which the query `a` is true with probabilistic abductive explanation {`e`}, together with their probability. I and E stand respectively for included and excluded.

```
a:0.4.
b:0.4.
abducible aa.
abducible bb.
q:- a,aa.
q:- b,bb.
:- aa,bb.
```

Both Δ_1 = `{aa}` *and* Δ_2 = `{bb}` *are minimal, each one giving a probability of 0.4.*

Example 6. *Consider the case of an abductive logic program (no probabilistic facts). For example, if we query* `a` *in the following program, where both* `b` *and* `c` *are abducibles:*

```
a:- b,c.
a:- c.
abducible b.
abducible c.
```

we would obtain, without the least *function, two explanations:* $\Delta_1 =$ `{b,c}` *and* $\Delta_2 =$ `{c}`*. However, this is in contrast with our definition, where the goal is to find sets that are also minimal. In this example* $\Delta_2 \subset \Delta_1$*, so the latter must not be considered as it is not minimal.*

Variant 1. If we add another clause `a:- d,e.` *with* `d` *and* `e` *abducibles, the set of explanations for* `a` *will be* $\Delta =$ `{{c},{d,e}}`*, since both are minimal.*

We now apply the semantics of probabilistic abductive logic programs to the "Stromboli" example.

Example 7. *Given the LPAD of Example 1 (where the variable* X *has been replaced by* _*),* $\mathcal{IC} = \emptyset$*, and* $A = \{C_3, C_4\}$*:*

```
eruption:0.6; earthquake:0.3 :- sudden_er, fault_rupture(_).
sudden_er: 0.7.
abducible fault_rupture(southwest_northeast).
abducible fault_rupture(east_west).
```

the query $q =$ `eruption` *has the probabilistic abductive explanation*[3]
$\Delta =$ `{fault_rupture(southwest_northeast),fault_rupture(east_west)}` *with probability* $P(q \mid \Delta) = 0.252 + 0.126 + 0.042 + 0.126 + 0.042 = 0.588$*, corresponding to worlds #1,2,3,7,13 in Table 1 where q is true given* Δ*.* Δ *yields the highest probability since*

- *given the probabilistic abductive explanations*
  $\Delta_1 =$ `{fault_rupture(southwest_northeast)}` *or*
  $\Delta_2 =$ `{fault_rupture(east_west)}`*,* $P(q \mid \Delta_1) = P(q \mid \Delta_2) = 0.42$*;*

- *given the probabilistic abductive explanation*
  $\Delta_3 = \emptyset$*,* $P(q \mid \Delta_3) = 0$*.*

[3] This example can be tested at `https://cplint.eu/e/eruption_abduction.pl`.

Variant 1. Note that, given the program:

```
eruption:0.6; earthquake:0.3 :- sudden_er, fault_rupture(_).
sudden_er: 0.7.
abducible fault_rupture(southwest_northeast).
fault_rupture(east_west).
```

the query $q =$ `eruption` *would have the probabilistic abductive explanation* $\Delta =$ `{fault_rupture(southwest_northeast)}` *with the same probability as above, corresponding to the same worlds. The same result would be achieved by making abducible* `fault_rupture(east_west)` *instead of* `fault_rupture(southwest_northeast)`*.*

Variant 2. If we remove $C3$ *or* $C4$ *from the program, for instance* $C4$*:*

```
eruption:0.6; earthquake:0.3 :- sudden_er, fault_rupture(_).
sudden_er: 0.7.
abducible fault_rupture(southwest_northeast).
```

we would lose the second grounding `X/east_west`*. Now, the query* $q =$ `eruption` *would have the probabilistic abductive explanation* $\Delta =$`{fault_rupture(southwest_northeast)}` *but with probability* $P(q \mid \Delta) = 0.42 + 0.18 = 0.6$*, corresponding to worlds #1,2 of Table 3, where* q *is true given* Δ*.*

Variant 3. If we add an IC to the program stating that a fault rupture cannot happen along both directions at the same time:

```
eruption:0.6; earthquake:0.3 :- sudden_er, fault_rupture(_).
sudden_er: 0.7.
abducible fault_rupture(southwest_northeast).
abducible fault_rupture(east_west).

:- fault_rupture(southwest_northeast),fault_rupture(east_west).
```

w	eruption:0.6; earthquake:0.3 :- sudden_er, fault_rupture(sw_ne).	sudden_er:0.7.	$P(w)$
1	eruption	sudden_er	0.42
2	eruption	*null*	0.18
3	earthquake	sudden_er	0.21
4	earthquake	*null*	0.09
5	*null*	sudden_er	0.07
6	*null*	*null*	0.03

Table 3: Possible worlds for the LPAD of Example 7 (Variant 2) with the corresponding probability, computed as the product of the probabilities associated with the head atoms taking value true, reported in each row. Highlighted rows represent the worlds in which the query `eruption` is true.

the probabilistic abductive explanations that maximize the probability of the query $q =$ `eruption` *and satisfy the constraint are both*

$\Delta_1 =$ `{fault_rupture(southwest_northeast)}`

and

$\Delta_2 =$ `{fault_rupture(east_west)}`*, as* $P(q,\mathcal{IC} \mid \Delta_1) = P(q,\mathcal{IC} \mid \Delta_2) = 0.252 + 0.126 + 0.042 = 0.42$ *in both cases.*

Note that the probabilistic abductive explanation found at the beginning of this example, with a higher probability $P(q \mid \Delta) = 0.588$*, is now forbidden by the IC.*

Several related tasks, such as Maximum a Posteriori (MAP), Most Probable Explanation (MPE), and Viterbi proof, require the selection of an optimal subset of facts to optimize a function value. However, there are some important differences with abduction that will be investigated in the next subsection.

4.1. Relation to MAP/MPE and Viterbi proof problems

In PLP, the probabilistic abductive problem differs both from the Maximum A Posteriori (MAP)/Most Probable Explanation (MPE) task [28] and from the Viterbi proof [29, 30, 31].

In general terms, given a joint probability distribution over a set of random variables, a set of values for a subset of the variables (evidence), and another disjoint subset of the variables (query variables[4]), the MAP problem consists of finding the most probable values for the query variables given the evidence. The MPE problem is the MAP problem where the set of query variables is the complement of the set of evidence variables. More formally, given an LPAD T, a conjunction of ground atoms e, the *evidence*, and a set of random variables $\mathbf{X}$ (query random variables), associated with some ground rules of T, the MAP problem is to find an assignment $\mathbf{x}$ of values to $\mathbf{X}$ such that $P(\mathbf{x} \mid e)$ is maximized, i.e., solve

$$\arg\max_{\mathbf{x}} P(\mathbf{x} \mid e).$$

The MPE problem is a MAP problem where $\mathbf{X}$ includes all the random variables associated with all ground clauses of T. These problems differ from ours because we want to find the set Δ that maximizes the probability of the query variables $P(\mathbf{x} \mid \Delta)$, rather than the value of the query variables with maximum probability.

Example 8. *Given the program T of Example 1 where the two certain facts are made probabilistic:*

```
(C1) eruption:0.6;earthquake:0.3 :- sudden_er, fault_rupture(X).
(C2) sudden_er:0.7.
(C3) fault_rupture(southwest_northeast):0.5.
(C4) fault_rupture(east_west):0.4.
```

and evidence is `ev:-eruption`*, if all the random variables associated with all ground clauses are query variables, the* MPE *task finds the most probable explanation for* `ev`*, i.e., the explanation with the highest probability, corresponding to the assignment* $\mathbf{x}$*:*

```
[rule(1,eruption,(sudden_er,fault_rupture(southwest_northeast))),
rule(1,eruption,(sudden_er,fault_rupture(east_west))),
```

[4]In this subsection, we use the word *query* associated with variables, with a slightly different meaning with respect to the rest of the paper.

```
rule(2,sudden_er,true),
rule(3,fault_rupture(southwest_northeast),true),
rule(4,null,true)]
```

Predicate `rule/3` *specifies respectively the clause number, the selected head, and the clause body with the selected grounding.* $P(\mathbf{x} \mid ev) = 0.6 \cdot 0.6 \cdot 0.7 \cdot 0.5 \cdot (1 - 0.4) = 0.0756$[5].

Example 9. *Given the program of Example 8 and the evidence* `ev:-eruption`*, if only the random variables associated with* C3 *and* C4 *are query, the MAP assignment* $\mathbf{x}$ *is:*

```
[rule(3,fault_rupture(southwest_northeast),true),
rule(4,null,true)]
```

with probability $P(\mathbf{x} \mid ev) = 0.126$*. This probability is computed as* $\frac{P(\mathbf{x},ev)}{P(ev)}$ *where* $\mathbf{x}$ *is the composite choice* $\kappa = \{(C_3, X/southwest\_northeast, 1), (C_4, \{\}, 2)\}$[6].

Differently, the Viterbi proof is the most probable proof for a query, i.e., it is a partial assignment (a partial possible world) such that for all assignments *extending* the proof, the query is still true. In practice, the Viterbi proof corresponds to the most likely explanation (proof) in the set of covering explanations for a query.

Example 10. *Given the program of Example 8, the covering set of explanations for the query* `eruption` *is* $K = \{\kappa_1, \kappa_2\}$ *(see Eq. 2 and 3).*

κ_1 *(Eq. 2) corresponds to the following partial assignment:*

```
[rule(1,eruption,(sudden_er,fault_rupture(southwest_northeast))),
rule(2,sudden_er,true),
rule(3,fault_rupture(southwest_northeast),true)]
```

[5]This example can be tested at `https://cplint.eu/e/eruption_mpe.pl`.
[6]This example can be tested at `https://cplint.eu/e/eruption_map.pl`.

having probability $0.6 \cdot 0.7 \cdot 0.5 = 0.21$.

κ_2 *(Eq. 3) corresponds to the following partial assignment:*

```
[rule(1,eruption,(sudden_er,fault_rupture(east_west))),
rule(2,sudden_er,true),
rule(4,fault_rupture(east_west),true)]
```

having probability $0.6 \cdot 0.7 \cdot 0.4 = 0.168$. *Being the Viterbi proof the most likely explanation in the set* K*, it corresponds to* κ_1[7].

In conclusion, the MAP/MPE task distinguishes between evidence and query variables, with the goal of finding the assignment of values to the query variables such that the probability of that assignment given the evidence atoms is maximized.

The probabilistic abductive problem, instead, aims at identifying the best set of ground atoms, explicitly defined in the program as abducibles, which maximizes the probability of a query, while possibly satisfying some (probabilistic) integrity constraints, which are admitted neither in the MAP/MPE task nor in the Viterbi proof task.

5. Algorithm

In PLP, the probability of the query is computed by building a BDD and by applying a dynamic programming algorithm that traverses it, such as the one presented in [26] and reported in Algorithm 1 for the sake of clarity. *var*(*node*) represents the variable associated with the BDD node *node* and *comp* is a flag that indicates whether a node pointer is complemented or not. Intermediate results are stored in a table to avoid the execution of the same computation in case the algorithm encounters an already visited node. Essentially, the BDD is traversed until a terminal node is found. From there, probabilities are computed and returned to the root.

[7] This example can be tested at `https://cplint.eu/e/eruption_vit.pl`.

Algorithm 1 Function PROB: computation of the probability of a BDD.

```
function PROB(node, TableProb)
    if node is a terminal then
        return 1
    else
        if TableProb(node.pointer) ≠ null then
            return TableProb(node)
        else
            p_0 ←PROB(child_0(node), TableProb)
            p_1 ←PROB(child_1(node), TableProb)
            if child_0(node).comp then
                p_0 ← (1 − p_0)
            end if
            Let π be the probability of being true of var(node)
            Res ← p_1 · π + p_0 · (1 − π)
            Add node.pointer → Res to TableProb
            return Res
        end if
    end if
end function
```

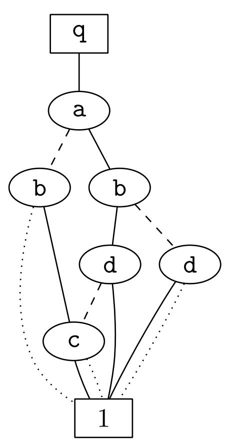
Algorithms 2 and 3 present the extension to the PITA system for returning the minimal set of the (probabilistic) abductive explanation for a query, by taking as input the root of a BDD representing its explanations.

Before analysing the algorithms, let us explain how ICs are managed. As described in the previous sections, they are represented as denials: a clause without head and a conjunction of literals in the body. Integrity constraints are implemented by conjoining BDDs. A BDD for the IC $:- \; b_1, \ldots, b_m$ is obtained by asking the query $b_1, \ldots, b_m$ with PITA (after applying the program transformation described in Appendix A). Abducible facts are represented with nodes (and thus Boolean random variables) of abducible type. Furthermore, constraints can contain variables and they can also be associated with probabilities. In this case, an extra variable associated with the probability is added to the BDD representing the constraint. Two BDDs, one for the query (BDDQ) and one for the constraints (BDDC), are built. The Boolean expression representing the query is given by the conjunction of BDDQ with the negation of BDDC (BDDQ and not BDDC). This definition can be straightforwardly extended in the case of multiple ICs. Consider the program shown in Example 11 and the query `q`.

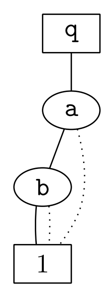
Example 11.

```
q:- a,d.
q:- b,c.
abducible a.
abducible b.
c:0.4.
d:0.5.
:- a,b.
```

BDDQ and BDDC represent respectively the Boolean expressions `(a and d) or (b and c)` (for the query `q`, Figure 5a) and `a and b` (for the constraint `:- a,b`, Figure 5b).



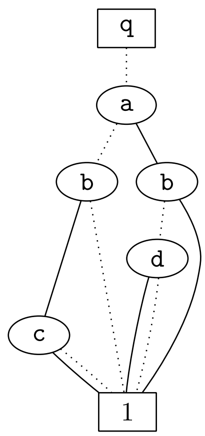
(a) BDD for `(a and d) or (b and c)` (BDDQ).



(b) BDD for `a and b` (BDDC).

Figure 5: BDDs for Example 11.

The final expression is the conjunction `((a and d) or (b and c)) and (not(a and b))`. Figure 6a shows the conjunction of BDDQ and BDDC while Figure 6b shows its truth table.



(a) BDD resulting from the conjunction of BDDQ and BDDC.

a	b	c	d	Expr
F	F	F	F	F
F	F	F	T	F
F	F	T	F	F
F	F	T	T	F
F	T	F	F	F
F	T	F	T	F
F	T	T	F	T
F	T	T	T	T
T	F	F	F	F
T	F	F	T	T
T	F	T	F	F
T	F	T	T	T
T	T	F	F	F
T	T	F	T	F
T	T	T	F	F
T	T	T	T	F

(b) Truth table.

Figure 6: BDD and truth table for Example 11. Highlighted rows represent the combinations of arguments such that the expression `((a and d) or (b and c)) and (not(a and b))` (compactly referred as Expr in the table) is true.

In the following, we describe step by step Algorithms 2 and 3. The function AbductiveExpl (Algorithm 2) gets as input the root of the BDD representing the explanations for a query, which is reordered (line 2) so that variables associated with abducibles come first in the order. This operation is crucial, since it allows us to directly integrate in PITA the algorithm to compute the probabilistic abductive explanation. Reordering the variables of a BDD may increase or decrease its size. However, having the abducible variables first allows the direct use of function Prob (Algorithm 1). *TableAbd* stores the pairs probability-set of explanations computed at each node corresponding to an abducible fact. Similarly, *TableProb* stores the values computed at probabilistic nodes and it is used by the function Prob. Both *TableAbd* and *TableProb* are initially empty. After that, function AbdInt (Algorithm 3) is called. This function starts from the root: if the current node does not represent an abducible, there are no abducibles in the remaining part of the diagram and

Algorithm 2 Function AbductiveExpl: computation of the minimal sets that maximize the joint probability of the query and the ICs, and of the corresponding probability.

1: **function** AbductiveExpl($root$)
2: $root' \leftarrow$ Reorder($root$) ▷ BDD reordering
3: $TableAbd \leftarrow \emptyset$
4: $TableProb \leftarrow \emptyset$
5: $(Prob, Abd) \leftarrow$ AbdInt($root'$, $TableAbd$, $TableProb$, $false$)
6: $Abd' \leftarrow$ RemoveDominated(Abd)
7: **return** $(Prob, Abd')$
8: **end function**

so the probability is computed using the function Prob (line 4) and a set Δ containing only an empty explanation is returned. This is possible only because the BDD has been reordered as previously described. The function Prob also handles the terminal case (i.e., BDD constant node 1). If a value for the current node has already been computed, it is retrieved from *TableAbd* and returned (lines 11 and 12).

Otherwise, function AbdInt is recursively called on both the true and false child. After the recursion, a *max* operation between the probability with or without the node is performed (line 16) to choose whether the abducible represented by the current node should be included in the explanations or not: if the probability of the true child is greater than the probability of the false child, the abducible represented by the current node is selected and added to the explanations. Otherwise, it is not. If it is selected (line 18), the probability at the current node is given by the probability of the true child (p_1). In this case, the set of explanations is built by adding the current abducible (represented by $var(node)$) to all the true child choices (represented by Abd_1) using the function AddNodeToExplanations. If the probabilities computed in the two children are the same, the explanations of the true child that are dominated (strict superset) by an explanation of the false child are removed (line 20). This is needed to preserve the minimality of the result: if we do not remove the explanations in the true child that are a superset of one explanation in the false child, we would

Algorithm 3 Function AbdInt: traversal of the BDD to compute the sets that maximize the joint probability of the query and the ICs and the corresponding value.

```
function AbdInt(node, TableAbd, TableProb, comp)
    comp ← node.comp ⊕ comp
    if var(node) is not associated with an abducible then
        p ←Prob(node)                                          ▷ Call to prob
        if comp then
            return (1 − p, [[]])
        else
            return (p, [[]])
        end if
    else
        if TableAbd(node.pointer) ≠ null then
            return TableAbd(node.pointer)
        else
            (p0, Abd0) ← AbdInt(child0(node), TableAbd, TableProb, comp)
            (p1, Abd1) ← AbdInt(child1(node).TableAbd, TableProb, comp)
            if p1 > p0 then                                    ▷ Max
                Abd ← AddNodeToExplanations(var(node), Abd1)
                Res ← (p1, Abd)
            else if p1 == p0 then                              ▷ Same probability
                Abd ←RemoveDominatedAndMerge(Abd0,Abd1)
                if Abd is empty then
                    Res ← (p0, Abd0)
                else
                    Res ← (p1, Abd)
                end if
            else
                Res ← (p0, Abd0)
            end if
            Add node.pointer → Res to TableAbd
            return Res
        end if
    end if
end function
```

obtain sets which are not minimal. We cannot remove the explanations of the false child that are dominated by an explanation of the true child: after the introduction of the current node in the explanations of the true child, the explanations that dominate the ones removed in the false child are no more subsets. This is because they will have the current node included, that it is not present in the explanations of the false child, thus breaking the subset relation. If the set of explanations obtained after the removal of the dominated ones is empty, the explanations of the false child (Abd_0), together with their probability (p_0), are returned (because the addition of the true child in the true explanations would still lead to a dominated explanation, so there is no need to consider it). Otherwise, the current node is added to all the true explanations and the result is merged with the explanations of the false child and returned. These operations are performed by the function RemoveDominatedAndMerge.

The sets of explanations are kept ordered, to speed up the comparisons. If the node is not selected (line 27), the probability and the set of explanations computed in the false child are returned. This function will return in variable Res the pair $P(q, \mathcal{IC} \mid \Delta)$ and the set Δ maximizing that probability. Note that, as in Algorithm 1, intermediate results (indicated with Res) are stored in a table to avoid the execution of the same computation in case the algorithm encounters an already visited node.

After the execution of function AbdInt, we remove once again the possible dominated sets from the set of explanations (Algorithm 2 line 6). Finally, Algorithm 2 returns the pair $(Prob, Abd')$ where $Prob = P(q, \mathcal{IC} \mid \Delta)$ and $Abd' = \text{least}(\arg\max_{\Delta} P(q, \mathcal{IC} \mid \Delta))$, i.e., the set of minimal sets Δ maximizing that probability.

Here, we focused on programs without function symbols (see Section 3). However, our algorithm can be extended to also manage programs with function symbols, and this can be an interesting direction for future work.

In the extreme case where there are no probabilistic facts, Algorithm 2 returns the abductive explanations: no probabilistic fact is involved, so the function Prob is called only to manage the terminal node. By definition, a BDD encodes a Boolean function that can be a solution of the abduction problem. In the case of multiple solutions, both the functions RemoveDominatedAndMerge and RemoveDominated eliminate those that are dominated, and the returned solutions are minimal.

Let us now focus on the complexity of the whole task. Exact inference in probabilistic logic programs is #P-complete (originating from the cost of the underlying graphical model) [32]. Here, we compute the probabilistic abductive explanation following the same pattern of exact inference in PLP (knowledge compilation and traversing the resulting structure with a dynamic programming algorithm) but we have an additional step, which is the reordering of the BDD. Changing the order of a BDD is done by swapping adjacent variables, an operation that can be performed polynomially [33]. We adopted this solution, and empirically noted (see Section 6) that the time required for this task is always negligible with respect to the traversing of the BDD. Checking whether one set is a subset of another set can be performed in a time linear with the size of the smallest of the two subsets since we kept them ordered. If the sets of explanations are of sizes respectively m and n, $m \cdot n$ comparisons are needed.

5.1. Execution Example

To better understand the algorithm, consider the illustrative program of Example 3 variant 1, shown, together with its BDD, in Figure 3. Suppose that the probability of `a` together with its probabilistic abductive explanation needs to be computed. The algorithm starts at node `a` and is recursively called until a non abducible node is found. Nodes `b` left and right are reached, and the probabilities are computed using the function PROB: for `b` left 0.3 is computed while for `b` right $0.3 + (1 - 0.3) \cdot 0.6 = 0.72$. At the left node `e`, a max operation between the true and false children is performed: $\max(0.3, 0.72) = 0.72$ and `e` is added to the current explanation, which now contains only `e`. Similarly, at right node `e`, $\max(0.6, 0) = 0.6$ and `e` is again added to the current empty explanation. At node `c`, $\max(0.72, 0.6) = 0.72$ so `c` is added to the true child's explanation {`e`} and the overall probability with its abducible explanation are respectively 0.72 and {`c,e`}.

The following theorem proves that Algorithm 2 solves the probabilistic abductive problem

Theorem 1. *Algorithm 2 solves the probabilistic abductive problem.*

Proof 1. *(Sketch) The BDDs that are generated for the query and the ICs represent the Boolean formulas according to which the query is true and the ICs are satisfied for the correctness of the PITA algorithm. By reordering the resulting BDD, we have abducible nodes first in the diagram: this means that when we reach a probabilistic node there are no more abducible nodes below and we can compute the probability of that node as in PITA. The upper diagram is then used to select the sets of abducibles that provide the largest probability by simply comparing the probabilities of the partial sets coming from the children. Special care must be taken for the case of equal probability of the two children because in this case domination must be checked.*

6. Experiments

We conducted some experiments to analyze the execution time of the proposed algorithm. We executed them on a cluster[8] with Intel® Xeon® E5-2630v3 running at 2.40 GHz on five synthetic datasets[9] taken from [28]: growing head (*gh*), growing negated body (*gnb*), *blood*, probabilistic graph (*graph*) and probabilistic complete graph (*complete graph*). As stated in Section 3 (see Definition 4), we consider only sound programs. For each one, we conducted three kinds of experiments: one with deterministic integrity constraints, one with probabilistic integrity constraints, and one without constraints. Since results with probabilistic and deterministic constraints are almost identical, only one curve is shown. We arbitrarily set the probability of all the integrity constraints to 0.5: this value typically indicates weak constraints. However, here we are interested in the execution time of our algorithm, not in the computed probability: if we set a value different from 0.5, we would likely obtain the same results in terms of execution time, since the BDDs must be traversed in the same manner.

[8] `http://www.fe.infn.it/coka/doku.php?id=start`

[9] All datasets can be found at: `https://bitbucket.org/machinelearningunife/palp_experiments`.

We selected the previously listed set of programs with the goal of covering a broad spectrum of possible cases: with *gh* and *gnb*, we investigate, respectively, how a growing number of atoms in the head and negated literals in the body influences the execution time. Furthermore, the dataset *gh* with integrity constraints has multiple explanations with the same probability. *blood* represents a possible application in the biological domain, while the experiments on graphs are representative of the motivating example introduced in Section 1 and can be as well associated to real-world scenarios. For all the experiments, we computed the total execution time which is given by the time required for constructing, reordering, and traversing the BDDs. As we discuss in Section 7, current comparable systems do not exist to the best of our knowledge, so a direct comparison with other implementations is not possible.

6.1. Data

The first dataset (*gh*) is composed of a set of programs characterized by clauses with a growing number of atoms (from 1 to 14) in the head. The most complex program has 28 clauses and 14 abducibles. The following is a program with two abducibles:

```
abducible aba1.
abducible aba2.
a0 :- a1.
a1:0.5:- aba1.
a0:0.5; a1:0.5:- a2.
a2:0.5:- aba2.
```

The query is `a0`. For the experiments with ICs, we considered an XOR constraint: only one abducible should be selected. For the previous example, this can be implemented with:

```
r:- aba1,aba2.
r:- \+aba1,\+aba2.
:- r.
```

In general, if there are `n` abducibles, an XOR constraint can be implemented with $\binom{n}{2} + 2$ clauses. In the previous example, $\binom{2}{2} + 2 = 3$. The second clause represents the case where none of the abducibles is considered. The third one (denial) forbids a disjunction of the first two clauses: `not((aba1 and aba2) or (not aba1 and not aba2))` which is true only if one between `aba1` and `aba2` is selected.

The second dataset (*gnb*) is composed of a set of programs with an increasing number of negated atoms (from 1 to 14) in the body of clauses. Each clause has an abducible fact in the body. The most complex program has 121 clauses and 16 abducibles. The following is a program with four abducibles:

```
abducible aba0.
abducible aba1.
abducible aba2.
abducible aba3.
a0:0.5:- a1, aba0.
a0:0.5:- \+a1,a2, aba0.
a0:0.5:- \+a1,\+a2,a3, aba0.
a1:0.5:- a2, aba1.
a1:0.5:- \+a2,a3, aba1.
a2:0.5:- a3, aba2.
a3:0.5:- aba3.
```

We are interested in the probability of `a0` since it depends on an increasing number of rules. In the experiment with ICs, we tested the edge case where all the abducibles should be selected. This situation can be represented with:

```
r:- \+aba0.
r:- \+aba1.
r:- \+aba2.
:- r.
```

The *blood* dataset is a set of programs that model the inheritance of blood type. Each program has an increasing number of ancestors (up to five levels in the genealogical tree) identified as mother and father for each person. The most complex program has 67 clauses and 2 abducibles with a variable argument with 20 groundings each. For the experiments with ICs, mother and father should not have the same blood type: this can be implemented using a single denial with variables. Here, we are interested in finding an explanation that maximizes the probability that a person `p` has a certain blood type.

The *graph* dataset represents a set of probabilistic graphs following a Barabási-Albert model generated with the Python `networkx` package[10], with the number of nodes ranging in [50,100] and parameter m_0 (representing the number of edges to attach from a new node to existing nodes) set to 2. Since the generation of the Barabási-Albert model is not deterministic, we created 100 different graph configurations and averaged the resulting inference times. The *complete graph* dataset represents one probabilistic complete graph where each pair of nodes is connected by an edge. In both datasets, every node has a probability of 0.5 of being connected to another node if the abducible representing the edge is selected. Thus, the number of abducibles is the same as the number of edges. The goal is to compute the minimal probabilistic abductive explanation that maximizes the probability of the existence of a path between nodes 1 and N, where N is the size of the graph (number of nodes). In the case of a complete graph, the number of edges, and thus abducibles, is $(N \cdot (N-1))/2$. For the experiments with ICs, we removed paths of length two up to five: if `path(A,B,L)` is the predicate that represents the path between nodes `A` and `B` with length `L`, this constraint can be imposed with `:- path(0,49,L), L < 6`.

To sum up, the datasets have the structure described in Table 4 that lists the number of probabilistic rules (#p), the number of atoms in the head (#h) per clause, the number of atoms in the body (#b), the number of abducibles (#a), the number of ICs (#IC), and the number of atoms in the body of ICs (#bIC) per IC for each of the five datasets, all parametric in n, the size of the program. We considered the datasets with ICs, since the values for the datasets

[10] `https://networkx.github.io/`

without ICs are equal, except for the number of ICs that is obviously 0.

Dataset	#p	#h	#b	#a	#IC	#bIC
blood	27 + n	{3,4}	{2,3}	2	2	2
gh	2n	[1,n]	1	n	1	$\binom{n}{2}+2$
gnb	$n \cdot (n-1)/2+1$	1	[1,n]	n	1	n
graph	$2(n-50)+96$	1	1	$2(n-50)+96$	6	1
complete graph	$n \cdot (n-1)/2$	1	1	$n*(n-1)/2$	3	1

Table 4: Details of the datasets.

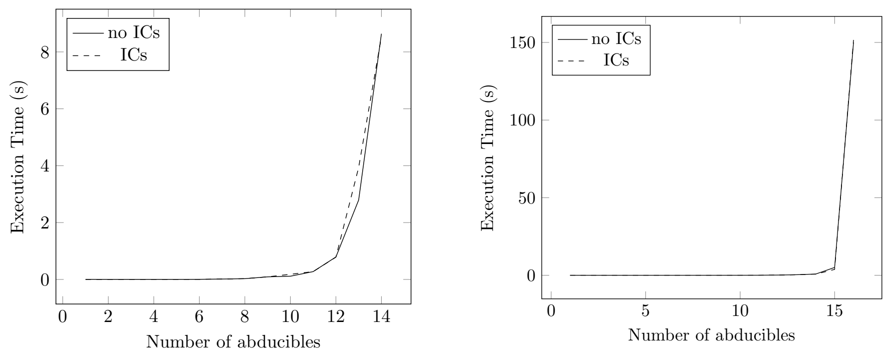
6.2. Discussion of Experiments Results

For *gh*, inference times are shown in Figure 7a. In the experiment without ICs, inference on programs with up to 12 abducibles takes less than 1 second. Starting from 13 abducibles, execution time grows exponentially. With ICs, inference on programs with up to 11 abducibles takes less than 1 second. As for the experiments without ICs, execution time then starts to grow exponentially, but with a steeper slope.

For *gnb*, inference times are shown in Figure 7b. In both types of experiments, they are very similar. Until 14 abducibles, inference takes less than 1 second. Starting from 15 abducibles, time grows exponentially. Overall, for both *gh* and *gnb*, experiments with ICs show slightly worse performance with respect to the version without, even if for the latter, results are often comparable.

For the *blood* dataset, execution time for experiments with and without ICs present similar performance (Figure 8). In detail, the execution time exceeds 1 hour for the dataset of size 36 for both programs with and without ICs.

For the *graph* dataset, Figure 9a shows that the execution time generally increases as the number of abducibles increases, reaching an exponential slope. For the *complete graph* dataset, Figure 9b shows that for a number of abducibles up to 6 nodes, inference time is constant and negligible; with 7 nodes, it increases rapidly to approximately 18 (with ICs) and 46 (without ICs) seconds. Finally, it exceeds 8 hours (the time limit) for size $N = 8$. Unlike the other datasets, in this



(a) Results for the *gh* dataset.

(b) Results for the *gnb* dataset.

Figure 7: Inference time as a function of the number of abducibles for *gh* and *gnb* datasets, with and without integrity constraints.

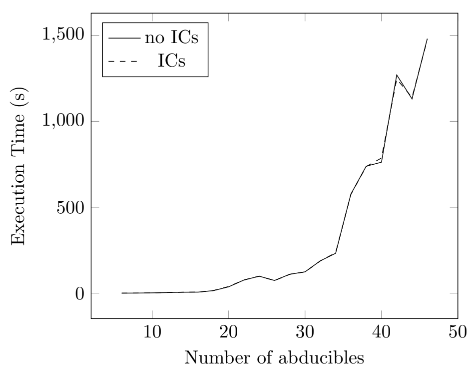
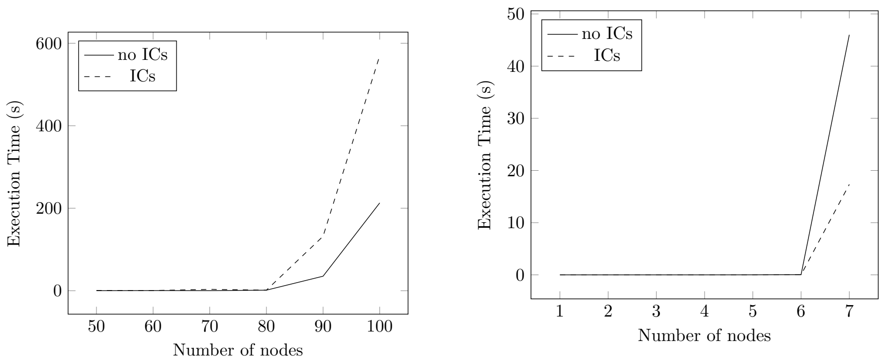


Figure 8: Inference time as a function of the number of abducibles for the *blood* dataset, with and without integrity constraints.

case the dashed curve (programs with ICs) is below the solid curve (programs without ICs).

Overall, the experiments with and without ICs take comparable time. This can be due to the implementation of the constraints, which may discard some of the possible solutions that can be obtained from the BDD without ICs. The experiments with ICs are faster only for the *complete graph* dataset: this happens probably because constraints allowed us to remove some paths.

Clearly, as in most of the applications, scalability is an issue. As the program size increases, the execution time increases, often exponentially. This is



(a) Results for the *graph* dataset.

(b) Results for the *complete graph* dataset.

Figure 9: Inference time as a function of the number of abducibles for the *graph* and *complete graph* datasets, with and without integrity constraints.

unavoidable given the complexity of the problem and the expressivity of the language. Solutions alternative to compiling to BDDs may be investigated, such as the technique of lifted inference: this will be an interesting direction for future work.

7. Related Work

Traditionally defined as *inference to the best explanation*, abduction embeds the implicit assumption that many possible explanations exist and raises the issue about which one should be selected. Adopting a purely logical setting, one may leverage the candidate explanations' complexity, preferring minimal ones. Still, different minimal but incomparable explanations are possible (there is no total ordering on them). Intuitively, one might want to select candidate explanations based on their "reliability", so that non-minimal explanations are not discarded by default. Interpreting "reliability" as (un)certainty opens a connection with the domain of probabilistic reasoning.

In fact, much research has been carried out aimed at combining logical and statistical inference, from early works [34] to more recent approaches such as *Probabilistic Logic Programming* [1, 2] and *Statistical Relational Learning*

(SRL) [35]. Of course, this also brings about additional problems that are typical of *Probabilistic Graphical Models* (PGM) [36] (parameter and model learning, inference). From a Logic Programming perspective, examples of embedding probabilistic reasoning in logic based on the so-called *distribution semantics* [8] are Logic Programs with Annotated Disjunctions (LPADs) [16], ProbLog [26], CP-Logic [37] and PRISM [8, 38]. Both ProbLog and PRISM allow to set probabilities only on facts, but the former allows two alternatives (true or false) only, one of which is implicit, while the latter allows more than two alternatives. PRISM offers the special predicate $msw(switch, value)$, encoding a random switch (i.e., a random variable), that can be used in the body of clauses to check that the random *switch* takes the value *value*. The possible values of each switch are defined by facts for the $values/2$ predicate, while the probability of each switch is set by calling the predicate $set\_sw/2$. With respect to ProbLog and PRISM, LPADs and CP-Logic offer the most general syntax. They only differ in that CP-Logic deems invalid some programs to which a causal meaning cannot be attached. As said, we considered LPADs.

Some works explicitly addressed probabilistic abductive reasoning: the authors of [39] explicitly addressed the issue of ranking explanations based on their likelihood. Like us, they propose a probabilistic abductive framework, based on the distribution semantics for normal logic programs, that handles negation as failure and integrity constraints in the form of denials. As in our case, the authors realize that in a probabilistic setting, abduction should aim at computing most preferred (i.e., likely), not minimal, solutions. So, they compute the probability of queries. Differently from them, among most preferred solutions, we still look for minimal ones, since we believe that abduced information is only tentative, and should be kept to a minimum. Connected to non-minimality, they propose an open world interpretation of abducibles. A first fundamental difference, and a claimed novel aspect of their approach, is treating ICs as evidence. More specifically, they define evidence as a set of integrity constraints. This is more expressive than traditional definitions of evidence, because denials can express NAND conditions to be fulfilled and using ICs made up of just one literal they can also set the truth (or falsity) of single atoms. Therefore, in their setting, "a query is a conjunction of existentially quantified literals and denials", and their goal is to compute $P(q \mid IC)$, where q is the query and IC is the evidence. Our goal is to compute $P(q, IC \mid \Delta)$. Another fundamental difference is that they consider a probability distribution over the truth values of each (ground) abducible and treat the integrity constraints as hard constraints that can never be violated, envisaging the possibility of viewing denials as a direction to pursue in future work. We addressed this issue in our work, allowing to set probabilities on integrity constraints.

Several proposals embed the Expectation Maximization (EM) algorithm. PRISM [40] is a system based on logic programming with multivalued random variables. While not providing support for integrity constraints, it includes a variety of top-level predicates which can generate abductive explanations. Introducing a probability distribution over abducibles, it chooses the best explanation using a generalized Viterbi algorithm. It can learn probabilities from training data. In essence, it performs what we called Viterbi proof. The authors of [41] extend the SOLAR system [42] with an abductive inference architecture exploiting an EM algorithm working on BDDs to evaluate hypotheses obtained from the process of hypothesis generation. In particular, all the minimal explanations are generated. Then, the EM algorithm working on a BDD representation is used to assign probabilities to atoms in explanations. As the final step, the probability of each hypothesis is computed to find the most probable one. For the comparison of our approach with MAP and Viterbi proofs, see Section 4.1.

Other solutions approached abduction from a deductive reasoning perspective. For example, the one proposed in [43] exploits Markov Logic Networks (MLN) [44]. Since MLNs provide only deductive inference, abduction is carried out by adding reverse implications for each rule in the knowledge base, this way increasing the size and complexity of the model, and its computational requirements. Like MLNs, most SRL formalisms use deduction for logical inference, and so, they cannot be used effectively for abductive reasoning. The authors of [45] adopt Stochastic Logic Programs [46], considering a number of *possi-*

ble worlds. Abduction is carried out by reversing the deductive flow of proof and collecting the probabilities associated with the involved clauses. Compared to our proposal, programs are restricted to SLP, and integrity constraints are not considered. However, the use of deduction without constraints may lead to wrong conclusions. Furthermore, an implementation is currently not available.

The solution presented in [47] describes an original approach to PALP based on Constraint Handling Rules, that allows interaction with external constraint solvers. As for our approach, it can return minimal explanations with their probabilities. Both an implementation returning all the solutions and one returning only the most probable one is provided. Differently from our approach, it attaches probabilities to abducibles only, and has limitations in the use of negation, that must be simulated by normal predicate symbols (e.g., *not_p(X)* for $\neg p(X)$). So, the expressiveness of the constraints is more limited than in our proposal.

In the context of Action-probabilistic logic programs (ap-programs), used for modelling behaviours of entities, in [48] the authors focused on the problem of maximizing the probability that an entity takes a (combination of) action(s), subject to some constraints (known as the Probabilistic Logic Abduction Problem, or PLAP). Specifically, they consider the Basic PLAP setting, where the goal is fixed (a predicate checking reachability of a desired situation from the current situation) and the answer is binary. Differently from our approach, in PLAP the program is ground, and variables and constraints only concern probabilities. Another approach that uses ap-programs for abductive query answering can be found in [49].

Some proposals approached probabilistic reasoning in abduction but did not make all the ALP components probabilistic. In [50], programs contain non-probabilistic definite clauses and probabilities are attached to abducible atoms. So, there are no structured constraints, and no integrated logic-based abductive proof procedure. cProbLog [51] extends regular ProbLog logic programs, where facts in the program can be associated with probabilities, to consider integrity constraints. It comes with a formal semantics and computational procedures, resulting in a powerful framework that encompasses the advantages of both PLP (ProbLog) and SRL (MLNs). Differently from our proposal, constraints are sharp, and thus all worlds that do not satisfy the constraints are ignored.

The discussion in [52] only considers ICs in the form of (universally quantified) *denials*, i.e., negations of conjunctions of literals. Other abductive frameworks proposed different kinds of integrity constraints: IFF [53] and its extensions, CIFF [54] and SCIFF [55], are based on integrity constraints that are clauses (i.e., implications with conjunctive premises and disjunctive conclusions). The solution proposed in [56] considers an ALP program enriched with integrity constraints à la IFF, possibly annotated with a probability value, that makes it possible to handle uncertainty of real-world domains. This language is also made richer by allowing for probabilistic abduction with variables, extending this way the answer capabilities of the proof-procedure. These probabilistic integrity constraints were defined in [57, 58], where programs containing such constraints are called Probabilistic Constraint Logic Theories (PCLTs) and may be learned directly from data by means of the PASCAL ("ProbAbiliStic inductive ConstrAint Logic") system. PCLTs however are theories only made up of constraints.

A recent proposal [59] extended traditional ALP by providing for several types of integrity constraints inspired by logic operators and allowing to attach probabilities to all components in the program (logic program, abducibles, and integrity constraints). Differently from this work, it allows ranking candidate explanations by likelihood but does not compute their exact probability.

While not explicitly computing with abduction, other systems may have a relationship to our work in that they merge logic programs, constraints, and probabilities. Specifically, Answer Set Programming (ASP) [60] may express denials and choice rules. There is a stream of works on probabilistic extensions of ASP that can deal with abduction through choice rules. Usually these works propose specific systems, implementations, or optimizations.

P-log [61] extends ASP by adding "random attributes" (that can be considered as random variables) of the form $a(X)$ where probabilistic information (understood as a measure of the degree of an agent's belief) about possible values of a is given through so-called 'pr-atoms'. The logical part of a program represents knowledge which determines the possible worlds of the program, while pr-atoms determine the probabilities of these worlds. LPMLN [62] extends ASPs by allowing weighted rules based on the Markov Logic weight scheme. LPMLN programs can be turned into P-log programs or into answer set MLN programs, to use their reasoning engines. As to the former, the translation of non-ground LPMLN programs yields unsafe ASPs. As to the latter, the straightforward implementation of a translation of an LPMLN program into an equivalent MLN results in effective computation. PrASP [63] is a probabilistic inductive logic programming (PILP) language and an uncertainty reasoning and statistical relational machine learning software, based on ASP. It includes limited support for inference with probabilistic normal logic programs under non-ASP-based semantics.

8. Conclusions

In this paper, we extended the PITA system to perform abductive reasoning on probabilistic abductive logic programs: given a probabilistic logic program, a set of abducible facts, and a set of (possibly probabilistic) integrity constraints, we want to compute minimal sets of abductive explanations (the probabilistic abductive explanation) such that the joint probability of the query and the constraints is maximized. The algorithm is based on Binary Decision Diagrams and was tested on several datasets, by including or not the constraints. Empirical results show that often the versions with and without constraints have comparable execution times: this may be due to the constraint implementation that discards some of the solutions. The code is available online and integrated in a publicly accessible web application at `https://cplint.eu` [11]. As future work, we plan to apply approximate inference [64] to speed up the computation: for example, if we consider the routing problem exposed in Section 1 and the *graph* experiments in Section 6, approximate inference will allow us to manage bigger graphs and handle real-world networks.

Acknowledgments

This research was partly supported by the “National Group of Computing Science (GNCS-INDAM)”.

References

[1] L. De Raedt, P. Frasconi, K. Kersting, S. Muggleton (Eds.), Probabilistic Inductive Logic Programming, Vol. 4911 of LNCS, Springer, 2008.

[2] F. Riguzzi, Foundations of Probabilistic Logic Programming: Languages, semantics, inference and learning, River Publishers, Gistrup, Denmark, 2018.

[3] D. Azzolini, F. Riguzzi, E. Lamma, Studying transaction fees in the bitcoin blockchain with probabilistic logic programming, Information 10 (11) (2019) 335. `doi:10.3390/info10110335`.

[4] A. Nguembang Fadja, F. Riguzzi, Probabilistic logic programming in action, in: A. Holzinger, R. Goebel, M. Ferri, V. Palade (Eds.), Towards Integrative Machine Learning and Knowledge Extraction, Vol. 10344 of Lecture Notes in Computer Science, Springer, 2017, pp. 89–116. `doi:10.1007/978-3-319-69775-8\_5`.

[5] R. Manhaeve, S. Dumancic, A. Kimmig, T. Demeester, L. De Raedt, Deepproblog: Neural probabilistic logic programming, in: Advances in Neural Information Processing Systems, 2018, pp. 3749–3759.

[6] A. C. Kakas, P. Mancarella, Abductive logic programming, in: Proceedings of NACLP Workshop on Non-Monotonic Reasoning and Logic Programming, 1990.

[7] A. C. Kakas, P. Mancarella, Database updates through abduction, in: Proceedings of the 16th VLDB, Morgan Kaufmann, 1990, pp. 650–661.

[8] T. Sato, A statistical learning method for logic programs with distribution semantics, in: L. Sterling (Ed.), Logic Programming, Proceedings of the Twelfth International Conference on Logic Programming, Tokyo, Japan, June 13-16, 1995, MIT Press, 1995, pp. 715–729.

[9] F. Riguzzi, T. Swift, Tabling and answer subsumption for reasoning on logic programs with annotated disjunctions, in: Technical Communications of the 26th International Conference on Logic Programming (ICLP 2010), Vol. 7 of LIPIcs, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010, pp. 162–171. `doi:10.4230/LIPIcs.ICLP.2010.162`.

[10] F. Riguzzi, E. Bellodi, E. Lamma, R. Zese, G. Cota, Probabilistic logic programming on the web, Softw. Pract. Exper. 46 (10) (2016) 1381–1396. `doi:10.1002/spe.2386`.

[11] M. Alberti, E. Bellodi, G. Cota, F. Riguzzi, R. Zese, `cplint` on SWISH: Probabilistic logical inference with a web browser, Intell. Artif. 11 (1) (2017) 47–64. `doi:10.3233/IA-170105`.

[12] J. W. Lloyd, Foundations of Logic Programming, 2nd Edition, Springer, 1987.

[13] A. Van Gelder, K. A. Ross, J. S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (3) (1991) 620–650.

[14] T. C. Przymusinski, Every logic program has a natural stratification and an iterated least fixed point model, in: Proceedings of the Eighth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '89, Association for Computing Machinery, New York, NY, USA, 1989, pp. 11–21. `doi:10.1145/73721.73723`.

[15] A. Van Gelder, K. Ross, J. S. Schlipf, Unfounded sets and well-founded semantics for general logic programs, in: Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Sys-

tems, PODS '88, Association for Computing Machinery, New York, NY, USA, 1988, pp. 221–230. doi:10.1145/308386.308444.

[16] J. Vennekens, S. Verbaeten, M. Bruynooghe, Logic programs with annotated disjunctions, in: B. Demoen, V. Lifschitz (Eds.), 20th International Conference on Logic Programming (ICLP 2004), Vol. 3131 of Lecture Notes in Computer Science, Springer, 2004, pp. 431–445. doi:10.1007/978-3-540-27775-0_30.

[17] R. Zese, E. Bellodi, F. Riguzzi, G. Cota, E. Lamma, Tableau reasoning for description logics and its extension to probabilities, Ann. Math. Artif. Intell. 82 (1–3) (2018) 101–130. doi:10.1007/s10472-016-9529-3.

[18] E. Bellodi, F. Riguzzi, Structure learning of probabilistic logic programs by searching the clause space, Theor. Pract. Log. Prog. 15 (2) (2015) 169–212. doi:10.1017/S1471068413000689.

[19] F. Riguzzi, N. Di Mauro, Applying the information bottleneck to statistical relational learning, Mach. Learn. 86 (1) (2012) 89–114. doi:10.1007/s10994-011-5247-6.

[20] D. Poole, Abducing through negation as failure: Stable models within the independent choice logic, J. Logic Program. 44 (1–3) (2000) 5–35. doi:10.1016/S0743-1066(99)00071-0.

[21] F. Riguzzi, T. Swift, Well-definedness and efficient inference for probabilistic logic programming under the distribution semantics, Theor. Pract. Log. Prog. 13 (2) (2013) 279–302. doi:10.1017/S1471068411000664.

[22] F. Riguzzi, The distribution semantics for normal programs with function symbols, Int. J. Approx. Reason. 77 (2016) 1–19. doi:10.1016/j.ijar.2016.05.005.

[23] L. G. Valiant, The complexity of enumeration and reliability problems, SIAM J. Comput. 8 (3) (1979) 410–421.

[24] A. Darwiche, P. Marquis, A knowledge compilation map, J. Artif. Intell. Res. 17 (2002) 229–264. doi:10.1613/jair.989.

[25] A. Thayse, M. Davio, J. P. Deschamps, Optimization of multivalued decision algorithms, in: 8th International Symposium on Multiple-Valued Logic, IEEE Computer Society Press, 1978, pp. 171–178.

[26] L. De Raedt, A. Kimmig, H. Toivonen, ProbLog: A probabilistic Prolog and its application in link discovery, in: M. M. Veloso (Ed.), 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), Vol. 7, AAAI Press/IJCAI, 2007, pp. 2462–2467.

[27] T. Sang, P. Beame, H. A. Kautz, Performing bayesian inference by weighted model counting, in: 20th National Conference on Artificial Intelligence (AAAI 2005), AAAI Press, Palo Alto, California USA, 2005, pp. 475–482.

[28] E. Bellodi, M. Alberti, F. Riguzzi, R. Zese, MAP inference for probabilistic logic programming, Theor. Pract. Log. Prog. 20 (5) (2020) 641–655. doi:10.1017/S1471068420000174.

[29] D. Poole, Logic programming, abduction and probability - a top-down anytime algorithm for estimating prior and posterior probabilities, New Generat. Comput. 11 (3) (1993) 377–400.

[30] T. Sato, Y. Kameya, A viterbi-like algorithm and em learning for statistical abduction, in: Proceedings of UAI2000 Workshop on Fusion of Domain Knowledge with Data for Decision Support, 2000.

[31] D. S. Shterionov, J. Renkens, J. Vlasselaer, A. Kimmig, W. Meert, G. Janssens, The most probable explanation for probabilistic logic programs with annotated disjunctions, in: J. Davis, J. Ramon (Eds.), 24th International Conference on Inductive Logic Programming (ILP 2014), Vol. 9046 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2015, pp. 139–153. doi:10.1007/978-3-319-23708-4_10.

[32] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, Adaptive computation and machine learning, MIT Press, Cambridge, MA, 2009.

[33] C. Jiang, J. Babar, G. Ciardo, A. S. Miner, B. Smith, Variable reordering in binary decision diagrams, in: 26th International Workshop on Logic and Synthesis, 2017, pp. 1–8.

[34] N. J. Nilsson, Probabilistic logic, Artif. Intell. 28 (1) (1986) 71–87.

[35] L. C. Getoor, Learning statistical models from relational data, Ph.D. thesis, Stanford, CA, USA, aAI3038093 (2002).

[36] J. Pearl, Graphical models for probabilistic and causal reasoning, in: The Computer Science and Engineering Handbook, 1997, pp. 697–714.

[37] J. Vennekens, M. Denecker, M. Bruynooghe, CP-logic: A language of causal probabilistic events and its relation to logic programming, Theor. Pract. Log. Prog. 9 (3) (2009) 245–308. doi:10.1017/S1471068409003767.

[38] T. Sato, Y. Kameya, PRISM: a language for symbolic-statistical modeling, in: 15th International Joint Conference on Artificial Intelligence (IJCAI 1997), Vol. 97, 1997, pp. 1330–1339.

[39] T. Calin-Rares, M. Nataly, R. Alessandra, B. Krysia, On minimality and integrity constraints in probabilistic abduction, in: Logic for Programming, Artificial Intelligence, and Reasoning, Springer, 2013, pp. 759–775.

[40] T. Sato, EM learning for symbolic-statistical models in statistical abduction, in: Progress in Discovery Science, Final Report of the Japanese Discovery Science Project, Springer, 2002, pp. 189–200.

[41] K. Inoue, T. Sato, M. Ishihata, Y. Kameya, H. Nabeshima, Evaluating abductive hypotheses using an EM algorithm on BDDs, in: 21st International Joint Conference on Artificial Intelligence (IJCAI 2009), Morgan Kaufmann Publishers Inc., 2009, pp. 810–815.

[42] H. Nabeshima, K. Iwanuma, K. Inoue, Solar: a consequence finding system for advanced reasoning, in: International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Springer, 2003, pp. 257–263.

[43] R. J. Kate, R. J. Mooney, Probabilistic abduction using markov logic networks, in: Proceedings of the IJCAI-09 Workshop on Plan, Activity, and Intent Recognition (PAIR-09), Pasadena, CA, 2009.

[44] M. Richardson, P. Domingos, Markov logic networks, Mach. Learn. 62 (1-2) (2006) 107–136.

[45] A. Arvanitis, S. H. Muggleton, J. Chen, H. Watanabe, Abduction with stochastic logic programs based on a possible worlds semantics, in: Short Paper Proceedings of the 16th International Conference on Inductive Logic Programming (ILP-06), University of Coruña, 2006.

[46] S. Muggleton, et al., Stochastic logic programs, Advances in inductive logic programming 32 (1996) 254–264.

[47] H. Christiansen, Implementing probabilistic abductive logic programming with constraint handling rules, in: T. Schrijvers, T. Frühwirth (Eds.), Constraint Handling Rules, Vol. 5388 of Lecture Notes in Computer Science, Springer, 2008, pp. 85–118. doi:10.1007/978-3-540-92243-8_5.

[48] G. Simari, V. S. Subrahmanian, Abductive Inference in Probabilistic Logic Programs, in: M. Hermenegildo, T. Schaub (Eds.), Technical Communications of the 26th International Conference on Logic Programming, Vol. 7 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2010, pp. 192–201. doi:10.4230/LIPIcs.ICLP.2010.192.

[49] G. I. Simari, J. P. Dickerson, A. Sliva, V. S. Subrahmanian, Parallel abductive query answering in probabilistic logic programs, ACM Trans. Comput. Logic 14 (2) (Jun. 2013). doi:10.1145/2480759.2480764.

[50] D. Poole, Probabilistic Horn abduction and Bayesian networks, Artif. Intell. 64 (1) (1993) 81–129.

[51] D. Fierens, G. V. den Broeck, M. Bruynooghe, L. D. Raedt, Constraints for probabilistic logic programming, in: D. Roy, V. Mansinghka, N. Goodman (Eds.), Proceedings of the NIPS Probabilistic Programming Workshop, 2012.

[52] A. C. Kakas, R. A. Kowalski, F. Toni, Abductive Logic Programming, J. Logic Comput. 2 (6) (1993) 719–770. `doi:10.1093/logcom/2.6.719`.

[53] T. H. Fung, R. A. Kowalski, The IFF proof procedure for abductive logic programming, J. Logic Program. 33 (2) (1997) 151–165.

[54] E. Ulrich, M. Paolo, S. Fariba, T. Giacomo, T. Francesca, Abductive logic programming with CIFF: System description, in: J. Alferes, J. Leite (Eds.), Logics in Artificial Intelligence. JELIA 2004, Vol. 3229 of Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2004. `doi:10.1007/978-3-540-30227-8\_56`.

[55] M. Alberti, F. Chesani, M. Gavanelli, E. Lamma, P. Mello, P. Torroni, Verifiable agent interaction in abductive logic programming: the SCIFF framework, ACM T. Comput. Log. 9 (4) (2008) 29:1–29:43. `doi:10.1145/1380572.1380578`.

[56] E. Bellodi, M. Gavanelli, R. Zese, E. Lamma, F. Riguzzi, Nonground abductive logic programming with probabilistic integrity constraints, Theory and Practice of Logic Programming 21 (5) (2021) 557–574. `doi:10.1017/S1471068421000417`.

[57] M. Alberti, E. Bellodi, G. Cota, E. Lamma, F. Riguzzi, R. Zese, Probabilistic constraint logic theories, in: A. Hommersom, S. Abdallah (Eds.), Proceedings of the 3rd International Workshop on Probabilistic Logic Programming (PLP), Vol. 1661 of CEUR Workshop Proceedings, Sun SITE Central Europe, Aachen, Germany, 2016, pp. 15–28.

[58] F. Riguzzi, E. Bellodi, R. Zese, M. Alberti, E. Lamma, Probabilistic inductive constraint logic, Machine Learning 110 (2021) 1–32. doi:10.1007/s10994-020-05911-6.

[59] S. Ferilli, Extending expressivity and flexibility of abductive logic programming, Journal of Intelligent Information Systems 51 (2018) 647–672.

[60] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, Commun. ACM 54 (12) (2011) 92–103. doi:10.1145/2043174.2043195.

[61] C. Baral, M. Gelfond, N. Rushton, Probabilistic reasoning with answer sets, Theor. Pract. Log. Prog. 9 (1) (2009) 57–144. doi:10.1017/S1471068408003645.

[62] J. Lee, S. Talsania, Y. Wang, Computing lpmln using asp and mln solvers, Theory and Practice of Logic Programming 17 (5-6) (2017) 942–960. doi:10.1017/S1471068417000400.

[63] M. Nickles, A tool for probabilistic reasoning based on logic programming and first-order theories under stable model semantics, in: L. Michael, A. Kakas (Eds.), Logics in Artificial Intelligence, Springer International Publishing, Cham, 2016, pp. 369–384.

[64] D. Azzolini, F. Riguzzi, E. Lamma, F. Masotti, A comparison of MCMC sampling for probabilistic logic programming, in: M. Alviano, G. Greco, F. Scarcello (Eds.), Proceedings of the 18th Conference of the Italian Association for Artificial Intelligence (AI*IA2019), Rende, Italy 19-22 November 2019, Vol. 11946 of Lecture Notes in Computer Science, Springer, Heidelberg, Germany, 2019. doi:10.1007/978-3-030-35166-3_2.

[65] W. Chen, D. S. Warren, Tabled evaluation with delaying for general logic programs, J. ACM 43 (1) (1996) 20–74. doi:10.1145/227595.227597.

[66] T. Swift, D. S. Warren, XSB: Extending prolog with tabled logic programming, Theor. Pract. Log. Prog. 12 (1-2) (2012) 157–187. doi:10.1017/S1471068411000500.

Appendix A. The PITA System

PITA (Probabilistic Inference with Tabling and Answer subsumption) [9, 21] computes the probability of a query from a probabilistic logic program in the form of an LPAD by first transforming the LPAD into a normal program containing calls for manipulating BDDs. The idea is to add an extra argument to each subgoal to store a BDD encoding the explanations for the answers of the subgoal. The values of the subgoals' extra arguments are combined using a set of general library functions:

- `init, end`: initializes and terminates the data structures for manipulating BDDs;
- `zero(-D), one(-D)`: `D` is the BDD representing the Boolean constants 0 or 1 respectively;
- `and(+D1,+D2,-DO), or(+D1,+D2,-DO), not(+D1,-DO)`: Boolean operations among BDDs `D1` and `D2`;
- `equality(+Var,+Value,-D)`: `D` is the BDD representing `Var=Value`, i.e., the multi-valued random variable `Var` is assigned `Value`;
- `ret_prob(+D,-P)`: returns the probability `P` of the BDD `D`.

As usual, `+` denotes input variables that must be instantiated when the predicate is called, while `-` is used for output variables that should not be instantiated when the predicate is called. These functions are implemented in C as an interface to the CUDD library for manipulating Binary Decision Diagrams. A BDD is represented in Prolog as an integer that is a pointer in memory to its root node. Moreover, the predicate `get_var_n(+R,+S,+Probs,-Var)` is implemented in Prolog and returns the multi-valued random variable `Var` associated with rule `R` with grounding substitution `S` and list of probabilities `Probs` in its head.

The PITA transformation applies to atoms, literals and clauses. The transformation for an atom `h` and a variable `D`, `PITA(h,D)`, is `h` with the variable `D` added as the last argument. The transformation for a negative literal `b = \+ a` and a variable `D`, `PITA(b,D)`, is the Prolog conditional

```
(PITA(a,DN)->
  not(DN,D)
;
  one(D)
).
```

In other words, the data structure `DN` is negated if `a` has some explanations; otherwise, the data structure for the constant function 1 is returned.

The disjunctive clause

```
cr = h1:p1 ; ... ; hn:pn :- b1,...,bm.
```

where the parameters `pi`, $i = 1, ..., n$ sum to 1, is transformed into the set of clauses `PITA(cr)`:

```
PITA(cr,i)=PITA(hi,D):- one(DD0),
                        PITA(b1,D1),and(DD0,D1,DD1),....,
                        PITA(bm,Dm),and(DDm-1,Dm,DDm),
                        get_var_n(r,V,[p1,...,pn],Var),
                        equality(Var,i,DD),and(DDm,DD,D).
```

for $i = 1, ..., n$, where `V` is a list containing all the variables appearing in `cr` and `r` is a unique identifier for `cr`. If the parameters do not sum up to 1, then $n - 1$ rules are generated as the last head atom, `null`, does not influence the query since it does not appear in any body. In the case of empty bodies or non-disjunctive clauses (a single head with probability 1), the transformation can be optimized.

The PITA transformation applied to Example 1 yields

```
PITA(c1,1) = eruption(D) :-
             one(DD0),sudden_er(D1),and(DD0,D1,DD1),
             fault_rupture(X,D2),and(DD1,D2,DD2),
```

```
                get_var_n(1,[X],[0.6,0.3,0.1],Var),
                equality(Var,1,DD),and(DD2,DD,D).
PITA(c1,2) = earthquake(D) :-
                one(DD0),sudden_er(D1),and(DD0,D1,DD1),
                fault_rupture(X,D2),and(DD1,D2,DD2),
                get_var_n(1,[X],[0.6,0.3,0.1],Var),
                equality(Var,2,DD),and(DD2,DD,D).
PITA(c2,1) = sudden_er(D) :-
                one(DD0), get_var_n(2,[],[0.7,0.3],Var),
                equality(Var,1,DD),and(DD0,DD,D).
PITA(c3,1) = fault_rupture(southwest_northeast,D) :- one(D).
PITA(c4,1) = fault_rupture(east_west,D) :- one(D).
```

Clause C_1 has three alternatives in the head but the last one is the `null` atom so only two clauses are generated. Clauses C_3 and C_4 are definite facts so their transformation is optimized as shown above.

PITA uses tabling [65] to ensure that, when a goal is asked again, the already computed answers for it are retrieved rather than recomputed. That saves time because explanations for different goals are memorized. Moreover, it also avoids non-termination in many cases. PITA also exploits the answer subsumption feature [66] such that, when a new answer for a tabled subgoal is found, it combines old answers with the new one according to a partial order or lattice. See [2] for further details.