Tabling and Answer Subsumption for Reasoning on Logic Programs with Annotated Disjunctions

FABRIZIO RIGUZZI
ENDIF – University of Ferrara
Via Saragat 1, I-44122, Ferrara, Italy
E-mail: fabrizio.riguzzi@unife.it

TERRANCE SWIFT
CENTRIA – Universidade Nova de Lisboa
E-mail: tswift@cs.suysb.edu

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Probabilistic Logic Programming is an active field of research, with many proposals for languages, semantics and reasoning algorithms. One such proposal, Logic Programming with Annotated Disjunctions (LPADs) represents probabilistic information in a sound and simple way.

This paper presents two contributions to the evaluation of LPADs. The first is the definition of a semantics for LPADs with function symbols, which is needed for many domains (including some standard benchmarks used in this paper). The second is the algorithm "Probabilistic Inference with Tabling and Answer subsumption" (PITA) for computing the probability of queries. Answer subsumption is a feature of tabling that allows the combination of different answers for the same subgoal in the case in which a partial order can be defined over them. We have applied it in our case since probabilistic explanations (stored as BDDs in PITA) possess a natural lattice structure.

PITA has been implemented in XSB and compared with the ProbLog, `cplint` and CVE systems. The results show that in almost all cases, PITA is able to solve larger problems and is faster than competing algorithms.

KEYWORDS: Probabilistic Logic Programming, Tabling, Answer Subsumption, Logic Programs with Annotated Disjunction, Program Transformation

1 Introduction

Languages that are able to represent probabilistic information have a long tradition in Logic Programming, dating back to (Shapiro 1983; van Emden 1986; Dantsin 1991; Ng and Subrahmanian 1992; Poole 1993). With these languages it is possible to model domains which contain uncertainty, as many real world domains do. Recently, efficient systems have started to appear for performing reasoning with these languages (De Raedt et al. 2007; Kimmig et al. 2008)

Logic Programs with Annotated Disjunction (LPADs) (Vennekens et al. 2004) are a particularly interesting formalism because of the simplicity of their syntax

and semantics along with their ability to model causation (Vennekens et al. 2009). LPADs share with many other languages a distribution semantics (Sato 1995): a theory defines a probability distribution over logic programs and the probability of a query is given by the sum of the probabilities of the programs where the query is true. In LPADs the distribution over logic programs is defined by means of disjunctive clauses in which the atoms in the head are annotated with a probability. The semantics of LPADs proposed in (Vennekens et al. 2004) requires the programs to be function-free, which is a strong requirement ruling out many interesting programs, including some of those used in our experiments in Section 8. Thus, we propose a version of the semantics that allow function symbols, along the lines of (Sato 1995; Poole 2000).

Various approaches have appeared for performing inference on LPADs. (Riguzzi 2007) proposed an algorithm that first finds all the possible explanations for a query and then makes them mutually exclusive by using Binary Decision Diagrams (BDDs), similarly to what has been proposed for the ProbLog language (De Raedt et al. 2007). (Riguzzi 2008) presented SLGAD resolution that extends SLG resolution by repeatedly branching on disjunctive clauses. (Meert et al. 2009) discusses the CVE algorithm that first transforms an LPAD into an equivalent Bayesian network and then performs inference on the network using the variable elimination algorithm.

In this paper, we present the algorithm "Probabilistic Inference with Tabling and Answer subsumption" (PITA) for computing the probability of queries from LPADs. PITA builds explanations for every subgoal encountered during a derivation of the query. The explanations are compactly represented using BDDs that also allow an efficient computation of the probability. Since all the explanations for a subgoal must be found, it is very useful to store such information so that it can be reused when the subgoal is encountered again. We thus propose to use tabling, which has already been shown useful for probabilistic logic programming in (Kameya and Sato 2000; Riguzzi 2008; Kimmig et al. 2009). This is achieved by transforming the input LPAD into a normal logic programs in which the subgoals have an extra argument storing a BDD that represents the explanations for its answers. Moreover, we also exploit answer subsumption to combine explanations coming from different clauses. PITA is tested on a number of datasets and compared with `cplint`, CVE and ProbLog (Kimmig et al. 2008). The algorithm was able to successfully solve more complex queries than the other algorithms in most cases and it was also almost always faster.

The paper is organized as follows. Section 2 briefly recalls tabling and answer subsumption. Section 3 illustrates the syntax and semantics of LPADs. Section 4 discusses the semantics of LPADs with function symbols. Section 5 defines dynamic stratification for LPADs. Section 6 gives an introduction to BDDs. Section 7 presents PITA and shows its correctness. Section 8 describes the experiments and Section 9 concludes the paper and presents directions for future works.

2 Tabling and Answer Subsumption

The idea behind tabling is to maintain in a table both subgoals encountered in a query evaluation and answers to these subgoals. If a subgoal is encountered more than once, the evaluation reuses information from the table rather than re-performing resolution against program clauses. Although the idea is simple, it has important consequences. First, tabling ensures termination of programs with the *bounded term-size property*. A program P has the bounded term size property if there is a finite function $f : N \rightarrow N$ such that if a query term Q to P has size $size(Q)$ then no term used in the derivation of Q has size greater than $f(size(Q))$. This makes it easier to reason about termination than in basic Prolog. Second, tabling can be used to evaluate programs with negation according to the Well-Founded Semantics (WFS) (van Gelder et al. 1991). Third, for queries to wide classes of programs, such as datalog programs with negation, tabling can achieve the optimal complexity for query evaluation. And finally, tabling integrates closely with Prolog, so that Prolog's familiar programming environment can be used, and no other language is required to build complete systems. As a result, a number of Prologs now support tabling including XSB, YAP, B-Prolog, ALS, and Ciao. In these systems, a predicate p/n is evaluated using SLDNF by default: the predicate is made to use tabling by a declaration such as *table p/n* that is added by the user or compiler.

This paper makes use of a tabling feature called *answer subsumption*. Most formulations of tabling add an answer A to a table for a subgoal S only if A is a not a variant (as a term) of any other answer for S. However, in many applications it may be useful to order answers according to a partial order or (upper semi-)lattice. In the case of a lattice, answer subsumption may be specified by means of a declaration such as *table arc(_,_,or/3 - zero/1)).* which indicates that if a table contains an answer $arc(Arg_1, Arg_2, Arg_{1,3})$, and a new answer $arc(Arg_1, Arg_2, Arg_{2,3})$ is derived, then $arc(Arg_1, Arg_2, Arg_{1,3})$ is replaced by $arc(Arg_1, Arg_2, or(Arg_{1,3}, Arg_{2,3}))$ ($zero/1$ is the bottom element of the lattice). In the PITA algorithm for LPADs presented in Section 7, if a table had an answer $arc(a, b, E_1)$ and a new answer $arc(a, b, E_2)$ were derived, where E_1 and E_2 are probabilistic explanations, the answer $arc(a, b, E_1)$ is replaced by $arc(a, b, E_3)$, where E_3 is the logical disjunction of the first two explanations, as stored in a BDD. Answer subsumption over arbitrary upper semi-lattices is implemented in XSB for stratified programs (Swift 1999b); in addition, the tabled aggregation of B-Prolog can also be seen as a form of answer subsumption.

Section 7 uses the SLG resolution (Chen and Warren 1996) extended with answer subsumption in its proof of Theorem 2, although similar results could be extended to other tabling formalisms that support negation and answer subsumption.

3 Logic Programs with Annotated Disjunctions

A *Logic Program with Annotated Disjunctions* (Vennekens et al. 2004) consists of a finite set of annotated disjunctive clauses of the form

$$h_1 : \alpha_1 \vee \ldots \vee h_n : \alpha_n \leftarrow b_1, \ldots, b_m$$

In such a clause $h_1, \ldots h_n$ are logical atoms and $b_1, \ldots, b_m$ are logical literals, $\{\alpha_1, \ldots, \alpha_n\}$ are real numbers in the interval $[0, 1]$ such that $\sum_{j=1}^{n} \alpha_j \leq 1$. $h_1 : \alpha_1 \vee \ldots \vee h_n : \alpha_n$ is called the *head* and $b_1, \ldots, b_m$ is called the *body*. Note that if $n = 1$ and $\alpha_1 = 1$ a clause corresponds to a normal program clause, sometimes called a *non-disjunctive* clause. If $\sum_{j=1}^{n} \alpha_j < 1$, 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 $1 - \sum_{j=1}^{n} \alpha_j$. For a clause C of the form above, we define $head(C)$ as $\{(h_i : \alpha_i) | 1 \leq i \leq n\}$ if $\sum_{i=1}^{n} \alpha_i = 1$ and as $\{(h_i : \alpha_i) | 1 \leq i \leq n\} \cup \{(null : 1 - \sum_{i=1}^{n} \alpha_i)\}$ otherwise. Moreover, we define $body(C)$ as $\{b_i | 1 \leq i \leq m\}$, $h_i(C)$ as h_i and $\alpha_i(C)$ as α_i.

The semantics of LPADs, given in (Vennekens et al. 2004), requires the ground program to be finite, so the program must not contain function symbols if it contains variables. If the LPAD is ground, a clause represents a probabilistic choice between the non-disjunctive clauses obtained by selecting only one atom in the head. As usual, if the LPAD T is not ground, T can be assigned a meaning by computing its grounding, $ground(T)$. By choosing a head atom for each ground clause of an LPAD we get a normal logic program called a *possible world* of the LPAD (called an *instance* of the LPAD in (Vennekens et al. 2004)). A probability distribution is defined over the space of possible worlds by assuming independence between the choices made for each clause.

More specifically, an *atomic choice* is a triple (C, θ, i) where $C \in T$, θ is a substitution that grounds C and $i \in \{1, \ldots, |head(C)|\}$. (C, θ, i) means that, for ground clause $C\theta$, the head $h_i(C) : \alpha_i(C)$ was chosen. A set of atomic choices κ is *consistent* if $(C, \theta, i) \in \kappa, (C, \theta, j) \in \kappa \Rightarrow i = j$, i.e., only one head is selected for a ground clause. A *composite choice* κ is a consistent set of atomic choices.

A *selection* σ is a composite choice that, for each clause $C\theta$ in $ground(T)$, contains an atomic choice (C, θ, i) in σ. We denote the set of all selections σ of a program T by $\mathcal{S}_T$. The *probability* $P(\kappa)$ *of a composite choice* κ is the product of the probabilities of the individual atomic choices, i.e. $P(\kappa) = \prod_{(C,\theta,i) \in \kappa} \alpha_i(C)$. A selection σ identifies a normal logic program w_σ defined as follows $w_\sigma = \{(h_i(C)\theta \leftarrow body(C))\theta | (C, \theta, i) \in \sigma\}$. w_σ is called a *possible world* (or simply *world*) of T. $\mathcal{W}_T$ denotes the set of all the possible worlds of T. Since selections are composite choices, we can assign a probability to possible worlds: $P(w_\sigma) = P(\sigma) = \prod_{(C,\theta,i) \in \sigma} \alpha_i(C)$.

We consider only *sound* LPADs in which every possible world has a total well-founded model. In this way, the uncertainty is modeled only by means of the disjunctions in the head and not by the features of the semantics. In the following we will write $w_\sigma \models \phi$ to mean that the closed formula ϕ is true in the well-founded model of the program w_σ.

The probability of a closed formula ϕ according to an LPAD T is given by the

sum of the probabilities of the possible worlds where the formula is true according to the WFS:

$$P(\phi) = \sum_{\sigma \in \mathcal{S}_T, w_\sigma \models \phi} P(\sigma)$$

It is easy to see that P satisfies the axioms of probability.

Example 1
Consider the dependency of a person's sneezing on his having the flu or hay fever:

$C_1 =$ *strong_sneezing*$(X) : 0{:}3 \vee$ *moderate_sneezing*$(X) : 0{:}5 \leftarrow$ *flu*(X)*:*
$C_2 =$ *strong_sneezing*$(X) : 0{:}2 \vee$ *moderate_sneezing*$(X) : 0{:}6 \leftarrow$ *hay_fever*(X)*:*
$C_3 =$ *flu*(*david*)*:*
$C_4 =$ *hay_fever*(*david*)*:*

This program models the fact that sneezing can be caused by flu or hay fever. Flu causes strong sneezing with probability 0.3, moderate sneezing with probability 0.5 and no sneezing with probability $1-0.3-0.5 = 0.2$; hay fever causes strong sneezing with probability 0.4, moderate sneezing with probability 0.3 and no sneezing with probability $1-0.2-0.6 = 0.2$. *strong_sneezing*(*david*) is true in 5 of the 9 instances of the program and its probability is

$$P_T(\textit{strong\_sneezing}(\textit{david})) = 0.3 \cdot 0.2 + 0.3 \cdot 0.6 + 0.3 \cdot 0.2 + 0.5 \cdot 0.2 + 0.2 \cdot 0.2 = 0.44$$

4 A Semantics for LPADs with Function Symbols

If a non-ground LPAD T contains function symbols, then the semantics given in Section 3 is not well-defined. If T contains function symbols, each possible world w_σ is the result of an infinite number of choices and the probability $P(w_\sigma)$ of w_σ is 0 since it is given by the product of an infinite number of factors all smaller than 1. Thus, the probability of a formula is 0 as well, since it is a sum of terms all equal to 0.

Therefore a new definition of the LPAD semantics is necessary. We provide such a definition following the approach proposed in (Poole 2000) for assigning a semantics to ICL programs with function symbols. A similar result can be obtained using the approach of (Sato 1995).

A composite choice κ identifies a set of possible worlds ω_κ that contains all the worlds relative to a selection that is a superset of κ, i.e.,

$$\omega_\kappa = \{w_\sigma | \sigma \in \mathcal{S}_T, \sigma \supseteq \kappa\}$$

Similarly we can define the set of possible worlds associated to a set of composite choices K:

$$\omega_K = \bigcup_{\kappa \in K} \omega_\kappa$$

Given a closed formula ϕ, we define the notion of explanation, covering set of composite choices and mutually incompatible set of explanations. A finite composite choice κ is an *explanation* for ϕ if ϕ is true in every world of ω_κ. In Example 1, the composite choice

$$\{(C_1, \{X/\textit{david}\}, 1)\}$$

is an explanation for $strong\_sneezing(david)$. A set of choices K is *covering* with respect to ϕ if every world w_σ in which ϕ is true is such that $w_\sigma \in \omega_K$. In Example 1, the set of composite choices

$$L_1 = \{\{(C_1, \{X/david\}, 1)\}, \{(C_2, \{X/david\}, 1)\}\} \tag{1}$$

is covering for $strong\_sneezing(david)$. Two composite choices κ_1 and κ_2 are *incompatible* if their union is inconsistent, i.e., if there exists a clause C and a substitution θ grounding C such that $(C, \theta, j) \in \kappa_1, (C, \theta, k) \in \kappa_2$ and $j \neq k$. A set K of composite choices is *mutually incompatible* if for all $\kappa_1 \in K, \kappa_2 \in K, \kappa_1 \neq \kappa_2 \Rightarrow \kappa_1$ and κ_2 are incompatible. The set of composite choices

$$\begin{aligned} L_2 \quad = \quad & \{\{(C_1, \{X/david\}, 1), (C_2, \{X/david\}, 2)\}, \\ & \{(C_1, \{X/david\}, 1), (C_2, \{X/david\}, 3)\}, \\ & \{(C_2, \{X/david\}, 1)\}\} \end{aligned} \tag{2}$$

is mutually incompatible for the theory of Example 1.

The following results of (Poole 2000) hold also for LPADs.

- Given a finite set K of finite composite choices, there exists a finite set K' of mutually incompatible finite composite choices such that $\omega_K = \omega_{K'}$.
- If K_1 and K_2 are both mutually incompatible sets of composite choices such that $\omega_{K_1} = \omega_{K_2}$ then $\sum_{\kappa \in K_1} P(\kappa) = \sum_{\kappa \in K_2} P(\kappa)$

Thus, we can define a unique probability measure $\mu : \Omega_T \to [0, 1]$ where Ω_T is defined as the set of sets of worlds identified by finite sets of finite composite choices: $\Omega_T = \{\omega_K | K \text{ is a finite set of finite composite choices}\}$. It is easy to see that Ω_T is an algebra over $\mathcal{W}_T$. Then μ is defined by

$$\mu(\omega_K) = \sum_{\kappa \in K'} P(\kappa)$$

where K' is a finite set of finite composite choices that is mutually incompatible and such that $\omega_K = \omega_{K'}$ As for ICL, $\langle \mathcal{W}_T, \Omega_T, \mu \rangle$ is a probability space (Kolmogorov 1950).

Definition 1
The probability of a ground atom ϕ is given by

$$P(\phi) = \mu(\{w_\sigma | w_\sigma \in \mathcal{W}_T \wedge w_\sigma \models \phi\}$$

Theorem 2 in Section 7 shows that, if T is a sound LPAD with bounded term size and ϕ is a ground atom, there is a finite set K of explanations of ϕ such that K is covering. Therefore $P(\phi)$ is well defined.

In the case of Example 1, L_2 shown in equation 2 is a covering set of explanations for $sneezing(david, strong)$ that is mutually incompatible, so

$$P(sneezing(david, strong)) = 0.3 \cdot 0.6 + 0.3 \cdot 0.2 + 0.2 = 0.44$$

5 Dynamic Stratification of LPADs

One of the most important formulations of stratification is that of *dynamic* stratification. (Przymusinski 1989) shows that a program has a 2-valued well-founded model iff it is dynamically stratified, so that it is the weakest notion of stratification that is consistent with the WFS. As presented in (Przymusinski 1989), dynamic stratification computes strata via operators on *3-valued interpretations* – pairs of the form $\langle T; F\rangle$, where T and F are subsets of the Herbrand base H_P of a normal program P.

Definition 2
For a normal program P, sets T and F of ground atoms, and a 3-valued interpretation I we define

$True_I(T) = \{A : val_I(A) \neq \mathtt{t}$ and there is a clause $B \leftarrow L_1, ..., L_n$ in P and a ground substitution θ such that $A = B\theta$ and for every $1 \leq i \leq n$ either $L_i\theta$ is true in I, or $L_i\theta \in T\}$;

$False_I(F) = \{A : val_I(A) \neq \mathtt{f}$ and for every clause $B \leftarrow L_1, ..., L_n$ in P and ground substitution θ such that $A = B\theta$ there is some i $(1 \leq i \leq n)$, such that $L_i\theta$ is false in I or $L_i\theta \in F\}$.

The conditions $val_I(A) \neq \mathtt{t}$ and $val_I(A) \neq \mathtt{f}$ are inessential, but ensure that only *new* facts are included in $True_I(T)$ and $False_I(F)$, and simplify the definition of dynamic strata below. (Przymusinski 1989) shows that $True_I$ and $False_I$ are both monotonic and defines $\mathcal{T}_I$ as the least fixed point of $True_I$ and $\mathcal{F}_I$ as the greatest fixed point of $False_I$. In words, the operator $\mathcal{T}_I$ extends the interpretation I to add the new atomic facts that can be derived from P knowing I; $\mathcal{F}_I$ adds the new negations of atomic facts that can be shown false in P by knowing I (via the uncovering of unfounded sets). An iterated fixed point operator builds up dynamic strata by constructing successive partial interpretations as follows

Definition 3 (*Iterated Fixed Point and Dynamic Strata*)
For a normal program P let

$$\begin{aligned} WFM_0 &= \langle \emptyset; \emptyset \rangle; \\ WFM_{\alpha+1} &= WFM_{\alpha+1} = WFM_\alpha \cup \langle \mathcal{T}_{WFM_\alpha}; \mathcal{F}_{WFM_\alpha} \rangle; \\ WFM_\alpha &= \bigcup_{\beta<\alpha} WFM_\beta, \text{ for limit ordinal} \alpha. \end{aligned}$$

Let $WFM(P)$ denote the fixed point interpretation WFM_δ, where δ is the smallest countable ordinal such that both sets T_{WFM_δ} and F_{WFM_δ} are empty. We refer to δ as the *depth* of program P. The *stratum* of atom A, is the least ordinal β such that $A \in WFM_\beta$ (where A may be either in the true or false component of WFM_β).

(Przymusinski 1989) shows that the iterated fixed point $WFM(P)$ is in fact the well-founded model and that any undefined atoms of the well-founded model do not belong to any stratum – i.e. they are not added to WFM_δ for any ordinal δ.

Dynamic stratification captures the order in which recursive components of a program must be evaluated. Because of this, dynamic stratification is useful for

modeling a variety of operational aspects of program evaluation. Fixed-order dynamic stratification (Sagonas et al. 2000), which will be used in Section 7, replaces the definition of $False_I(F)$ in Definition 2 is by

$False_I(F) = \{A : val_I(A) \neq \mathtt{f}$ and for every clause $B \leftarrow L_1, ..., L_n$ in P and ground substitution θ such that $A = B\theta$ there exists a ***failing prefix***: i.e., there is some i $(1 \leq i \leq n)$, such that $L_i\theta$ is false in I or $L_i\theta \in F$, and for all j $(1 \leq j \leq i-1)$, $L_j\theta$ is true in $I\}$.

(Sagonas et al. 2000) describes how fixed-order dynamic stratification captures those programs that a tabled evaluation can evaluate with a fixed literal selection strategy (i.e. without the SLG operations of SIMPLIFICATION and DELAY). As shown from the following example, fixed-order stratification is a fairly weak condition for a program.

Example 2
The following program has a 2-valued well-founded model and so is dynamically stratified, but does not belong to other stratification classes, such as local, modular, or weak stratification.

$s \leftarrow \neg s,\ p.$ $\qquad$ $s \leftarrow \neg p,\ \neg q,\ \neg r.$
$p \leftarrow q,\ \neg r,\ \neg s.$ $\qquad$ $q \leftarrow r,\ \neg p.$
$r \leftarrow p,\ \neg q.$

p, q, and r all belong to stratum 0, while s belongs to stratum 1. The simple program

$p \leftarrow \neg\ p.$ $\qquad$ $p.$

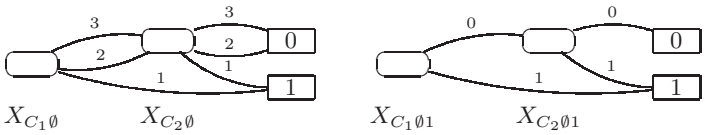
is fixed-order stratified, but not locally, modularly, or weakly stratified. Fixed-order stratification is more general than local stratification, and than modular stratification (since modular stratified programs can be decidably rearranged so that they have failing prefixes). It is neither more nor less general than weak stratification.

The above definitions of (fixed-order) dynamic stratification for normal programs can be straightforwardly adapted to LPADs – an LPAD is *(fixed-order) dynamically stratified* if each $w \in \mathcal{W}_\mathcal{T}$ is (fixed-order) dynamically stratified.

6 Representing Explanations by Means of Decision Diagrams

In order to represent explanations we can use Multivalued Decision Diagrams (Thayse et al. 1978). An MDD represents a function $f(\mathbf{X})$ taking Boolean values on a set of multivalued variables $\mathbf{X}$ by means of a rooted graph that has one level for each variable. Each node has one child for each possible value of the multivalued variable associated to the level of the node. The leaves store either 0 or 1. Given values for all the variables $\mathbf{X}$, an MDD can compute the value of $f(\mathbf{X})$ by traversing the graph starting from the root and returning the value associated to the leaf that is reached.

In order to represent sets of explanations with MDDs, each ground clause $C\theta$ appearing in the set of explanations is associated to a multivalued variable $X_{C\theta}$ with



(a) MDD. (b) BDD.

Fig. 1. Decision diagrams for Example 1.

as many values as atoms in the head of C. Each atomic choice (C, θ, i) is represented by the propositional equation $X_{C\theta} = i$. Equations for a single explanation are conjoined and the conjunctions for the different explanations are disjoined. The resulting function assumes value 1 if the values assumed by the multivalued variables correspond to an explanation for the goal.

The set of explanations in Equation (1) can be represented by the function

$$f(\mathbf{X}) = (X_{C_1\emptyset} = 1) \vee (X_{C_2\emptyset} = 1) \tag{3}$$

The corresponding MDD is shown in Figure 1(a).

If we consider the multivalued variables as random variables, the probability of the goal is given by the probability of $f(\mathbf{X})$ taking value 1. This is equivalent to computing the probability of a DNF formula which is an NP-hard problem.

The advantage of MDDs is that they represent a Boolean function $f(\mathbf{X})$ by means of a generalization of the Shannon's expansion

$$f(\mathbf{X}) = (X_1 = 1) \wedge f_{X_1=1}(\mathbf{X}) \vee \ldots \vee (X_1 = n) \wedge f_{X_1=n}(\mathbf{X})$$

where X_1 is the variable associated to the root node of the diagram and $f_{X_1=i}(\mathbf{X})$ is the function associated to the i-th child of the root node. The expansion can be applied recursively to the functions $f_{X_1=1}(\mathbf{X})$. This expansion allows the probability of $f(\mathbf{X})$ to be expressed by means of the following recursive formula

$$P(f(\mathbf{X})) = P(X_1 = 1) \cdot P(f_{X_1=1}(\mathbf{X})) + \ldots + P(X_1 = n) \cdot P(f_{X_1=n}(\mathbf{X}))$$

because the disjuncts are mutually exclusive due to the presence of the $X_1 = i$ equations. Thus the probability of $f(\mathbf{X})$ can be computed by means of a dynamic programming algorithm that traverses the MDD and sums up probabilities.

Decision diagrams can be built with various software packages that provide highly efficient implementation of Boolean operations. However, most packages are restricted to work on Binary Decision Diagram (BDD), i.e., decision diagrams where all the variables are Boolean. To work on MDD with a BDD package, we must represent multivalued variables by means of binary variables. Various options are possible, we found that the following, proposed in (De Raedt et al. 2008), gives the best performance. For a variable X_1 having n values, we use $n-1$ Boolean variables $X_{11}, \ldots, X_{1n-1}$ and we represent the equation $X_1 = i$ for $i = 1, \ldots n-1$ by means of the conjunction $\overline{X_{11}} \wedge \overline{X_{12}} \wedge \ldots \wedge \overline{X_{1i-1}} \wedge X_{1i}$, and the equation $X_1 = n$ by means of the conjunction $\overline{X_{11}} \wedge \overline{X_{12}} \wedge \ldots \wedge \overline{X_{1n-1}}$. The BDD representation of the function in Equation 3 is given in Figure 1(b). The Boolean variables are associated with the following parameters: $P(X_{11}) = P(X_1) \ldots P(X_{1i}) = \frac{P(X_1=i)}{1-P(X_{1i-1})}$.

7 Program Transformation

The first step of the PITA algorithm is to apply a program transformation to an LPAD to create a normal normal program that contains calls for manipulating BDDs. In our implementation, these calls provide a Prolog interface to the CUDD[1] C library and use the following predicates[2]

- *init, end*: for allocation and deallocation of a BDD manager, a data structure used to keep track of the memory for storing BDD nodes;
- *zero(-BDD), one(-BDD), and(+BDD1,+BDD2,-BDDO), or(+BDD1,+BDD2, -BDDO), not(+BDDI,-BDDO)*: Boolean operations between BDDs;
- *add_var(+N_Val,+Probs,-Var)*: addition of a new multi-valued variable with *N_Val* values and parameters *Probs*;
- *equality(+Var,+Value,-BDD)*: *BDD* represents *Var=Value*, i.e. that the random variable *Var* is assigned *Value* in the BDD;
- *ret_prob(+BDD,-P)*: returns the probability of the formula encoded by *BDD*.

add_var(+N_Val,+Probs,-Var) adds a new random variable associated to a new instantiation of a rule with *N_Val* head atoms and parameters list *Probs*. The auxiliary predicate *get_var_n/4* is used to wrap *add_var/3* and avoid adding a new variable when one already exists for an instantiation. As shown below, a new fact *var(R,S,Var)* is asserted each time a new random variable is created, where R is an identifier for the LPAD clause, S is a list of constants, one for each variables of the clause, and *Var* is a integer that identifies the random variable associated with clause R under a particular grounding. The auxiliary predicates has the following definition

$get\_var\_n(R, S, Probs, Var) \leftarrow$
  $(var(R, S, Var) \rightarrow true;$
  $length(Probs, L), add\_var(L, Probs, Var), assert(var(R, S, Var))).$

where *Probs* is a list of floats that stores the parameters in the head of rule R. R, S and *Probs* are input arguments while *Var* is an output argument.

The PITA transformation applies to clauses, literals and atoms.

- If h is an atom, $PITA_h(h)$ is h with the variable BDD added as the last argument.
- If b_j is an atom, $PITA_b(b_j)$ is b_j with the variable B_j added as the last argument.

In either case for an atom A, $BDD(\text{PITA}(A))$ is the value of the last argument of $PITA(A)$,

- If b_j is negative literal $\neg a_j$, $PITA_b(b_j)$ is the conditional $(PITA'_b(a_j) \rightarrow not(BN_j, B_j); one(B_j))$, where $PITA'_b(a_j)$ is a_j with the variable BN_j added as the last argument.

[1] `http://vlsi.colorado.edu/~fabio/`
[2] BDDs are represented in CUDD as pointers to their root node.

In other words the input BDD, BN_k, is negated if it exists; otherwise the BDD for the constant function 1 is returned.

- A non-disjunctive fact $C_r = h$ is transformed into the clause $PITA(C_r) = PITA_h(h) \leftarrow one(BDD)$.
- A disjunctive fact $C_r = h_1 : \alpha_1 \vee \ldots \vee h_n : \alpha_n.$ where the parameters sum to 1, is transformed into the set of clauses $PITA(C_r)$

  $PITA(C_r, 1) = PITA_h(h_1) \leftarrow \quad get\_var\_n(i, [], [\alpha_1, \ldots, \alpha_n], Var),$
  $equality(Var, 1, BDD).$

  $\ldots$

  $PITA(C_r, n) = PITA_h(h_n) \leftarrow \quad get\_var\_n(r, [], [\alpha_1, \ldots, \alpha_n], Var),$
  $equality(Var, n, BDD).$

  In the case where the parameters do not sum to one, the clause is first transformed into $h_1 : \alpha_1 \vee \ldots \vee h_n : \alpha_n \vee null : 1 - \sum_1^n \alpha_i.$ and then into the clauses above, where the list of parameters is $[\alpha_1, \ldots, \alpha_n, 1 - \sum_1^n \alpha_i]$ but the $(n+1)$-th clause (the one for $null$) is not generated.
- The definite clause $C_r = h \leftarrow b_1, b_2, \ldots, b_m.$ is transformed into the clause

  $PITA(C_r) = PITA_h(h) \leftarrow \quad PITA_b(b_1), PITA_b(b_2), and(B_1, B_2, BB_2),$
  $\ldots, PITA_b(b_m), and(BB_{m-1}, B_m, BDD).$
- The disjunctive clause
  $C_r = h_1 : \alpha_1 \vee \ldots \vee h_n : \alpha_n \leftarrow b_1, b_2, \ldots, b_m.$
  where the parameters sum to 1, is transformed into the set of clauses $PITA(C_r)$

  $PITA(C_r, 1) = PITA_h(h_1) \leftarrow \quad PITA_b(b_1), PITA_b(b_2), and(B_1, B_2, BB_2),$
  $\ldots,$
  $PITA_b(b_m), and(BB_{m-1}, B_m, BB_m),$
  $get\_var\_n(r, VC, [\alpha_1, \ldots, \alpha_n], Var),$
  $equality(Var, 1, B), and(BB_m, B, BDD).$

  $\ldots$

  $PITA(C_r, n) = PITA_h(h_n) \leftarrow \quad PITA_b(b_1), PITA_b(b_2), and(B_1, B_2, BB_2),$
  $\ldots,$
  $PITA_b(b_m), and(BB_{m-1}, B_m, BB_m),$
  $get\_var\_n(r, VC, [\alpha_1, \ldots, \alpha_n], Var),$
  $equality(Var, n, B), and(BB_m, B, BDD).$

  where VC is a list containing each variable appearing in C_r. If the parameters do not sum to 1, the same technique used for disjunctive facts is used.

Example 3

Clause C_1 from the LPAD of Example 1 is translated into

$strong\_sneezing(X, BDD) \quad \leftarrow flu(X, B_1),$
$get\_var\_n(1, [X], [0.3, 0.5, 0.2], Var),$
$equality(Var, 1, B), and(B_1, B, BDD).$

$moderate\_sneezing(X, BDD) \quad \leftarrow flu(X, B_1),$
$get\_var\_n(1, [X], [0.3, 0.5, 0.2], Var),$
$equality(Var, 2, B), and(B_1, B, BDD).$

while clause C_3 is translated into

$flu(david, BDD) \quad \leftarrow one(BDD).$

In order to answer queries, the goal *solve(Goal,P)* is used, which is defined by

$$\begin{aligned} solve(Goal, P) \quad & \leftarrow init, retractall(var(\_,\_,\_)), \\ & add\_bdd\_arg(Goal, BDD, GoalBDD), \\ & (call(GoalBDD) \rightarrow ret\_prob(BDD, P); P = 0.0), \\ & end. \end{aligned}$$

Moreover, various predicates of the LPAD should be declared as tabled. For a predicate p/n, the declaration is *table p(_1,...,_n,or/3-zero/1)*, which indicates that answer subsumption is used to form the disjunct of multiple explanations: At a minimum, the predicate of the goal should be tabled; as in normal programs, tabling may also be used for to ensure termination of recursive predicates, or to reduce the complexity of evaluations.

Correctness of PITA In this section we consider the correctness of the PITA transformation and its tabled evaluation[3]. For the purposes of our semantics, we consider the BDDs produced as ground terms, and do not specify them further. We first state the correctness of the PITA transformation with respect to the well-founded semantics of LPADs. Because we allow LPADs to have function symbols, care must be taken to ensure that explanations are finite. To accomplish this, we prove correctness for what we term finitary programs, essentially those for which a derivation in the well-founded semantics does not depend on an infinite unfounded set. In the statement of Theorem 1, for a ground atom a for a predicate p/n, $PITA_h(a)$ is an atom of predicate $p/(n+1)$ whose last argument is a variable for the BDD. In the well-founded model, an atom $PITA_h(a)\theta$ has its last argument instantiated to a given BDD: $BDD(PITA_h(a)\theta)$.

Theorem 1 (*Correctness of PITA Transformation*)
Let T be a sound finitary LPAD. Then σ is a finite explanation for a ground atom a iff there is some $PITA_h(a)\theta$ in $WFM(PITA_h(ground(T)))$, such that σ is a path in $BDD(PITA_h(a)\theta)$

Theorem 2 below states the correctness of the tabling implementation of PITA since the BDD returned for a tabled query is the disjunction of all covering explanations for that query. The proof uses an extension of SLG evaluation that includes answer subsumption but that is restricted to fixed-order dynamically stratified programs (Swift 1999b), a formalism that models the implementation tested in Section 8. Note that unlike Theorem 1, Theorem 2 does not require the program T to be grounded. However, Theorem 2 does require T to be range-restricted in order to ensure that tabled evaluation grounds answers. A normal program/LPAD is *range restricted* if all the variables appearing in the head of each clause appear also in the body. If a normal program is range restricted, every successful SLDNF-derivation for G completely grounds G (Muggleton 2000), a result that can be straightforwardly extended to tabled evaluations. In addition, Theorem 2 requires T to have the bounded-term-size property (cf. Section 2).

[3] Due to space limitations, our presentation is somewhat informal: a formal presentation with all proofs and supporting definitions can be found at `http://www.ing.unife.it/docenti/FabrizioRiguzzi/Papers/RigSwi09-TR.pdf`.

Theorem 2 (*Correctness of PITA Evaluation*)
Let T be a range-restricted, bounded-term-depth, fixed-order dynamically stratified LPAD and a a ground atom. Let $\mathcal{E}$ be an SLG evaluation of $PITA(a)$ against $PITA(T)$, such that answer subsumption is declared on $PITA(a)$ using BDD-disjunction. Then $\mathcal{E}$ terminates with an answer ans for $PITA(a)$ and $BDD(ans)$ represents the set of all covering explanation for a.

8 Experiments

PITA was tested on programs encoding biological networks from (De Raedt et al. 2007), a game of dice from (Vennekens et al. 2004) and the four testbeds of (Meert et al. 2009). PITA was compared with the exact version of ProbLog (De Raedt et al. 2007) available in the git version of Yap as of 19/12/2009, with the version of `cplint` (Riguzzi 2007) available in Yap 6.0 and with version CVE (Meert et al. 2009) available in ACE-ilProlog 1.2.20[4]. The biological network problems compute the probability of a path in a large graph in which the nodes encode biological entities and the links represents conceptual relations among them. Each programs in this dataset contains a definition of path plus a number of links represented by probabilistic facts. The programs have been sampled from a very large graph and contain 200, 400, ..., 5000 edges. Sampling has been repeated ten times, so overall we have 10 series of programs of increasing size. In each test we queried the probability that the two genes HGNC_620 and HGNC_983 are related[5].

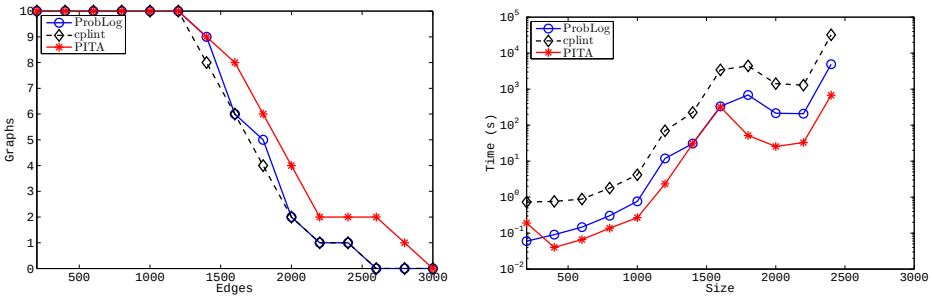
We ran PITA, ProbLog and `cplint` on the graphs in sequence starting from the smallest program and in each case we stopped after one day or at the first graph for which the program ended for lack of memory[6]. In PITA, we used the group sift method for automatic reordering of BDDs variables. Figure 2(a) shows the number of subgraphs for which each algorithm was able to answer the query as a function of the size of the subgraphs, while Figure 2(b) shows the execution time averaged over all and only the subgraphs for which all the algorithms succeeded. PITA was able to solve more subgraphs and in a shorter time than `cplint` and ProbLog. For PITA the vast majority of time for larger graphs was spent on BDD maintenance.

The second problem[7] models a game in which a die with three faces is repeatedly thrown until a 3 is obtained. This problem is encoded by the program

on(0,1):1/3 ; on(0,2):1/3 ; on(0,3):1/3.
on(T,1):1/3 ; on(T,2):1/3 ; on(T,3):1/3 ←
T1 is T-1, T1¿=0, on(T1,F), on(T1,3).

For this problem, we query the probability of *on(T,F)* for increasing values of *N*. In

[4] All experiments were performed on Linux machines with an Intel Core 2 Duo E6550 (2333 MHz) processor and 4 GB of RAM.
[5] The definition of path that is used is the one in (Kimmig et al. 2008) that performs loop checking explicitly by keeping the list of visited nodes.
[6] CVE was not applied to this dataset because the current version can not handle graph cycles.
[7] In the remaining problems, ProbLog was not considered because the publicly available version is not yet able to deal with non-binary variables



(a) Number of successes. (b) Execution times.

Fig. 2. Biological graph experiments.

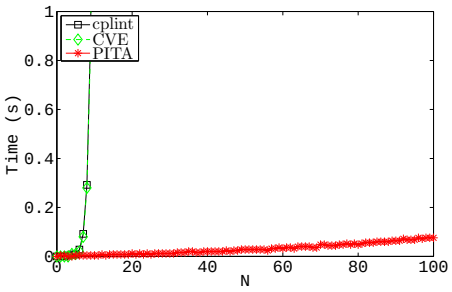


Fig. 3. Three sided die.

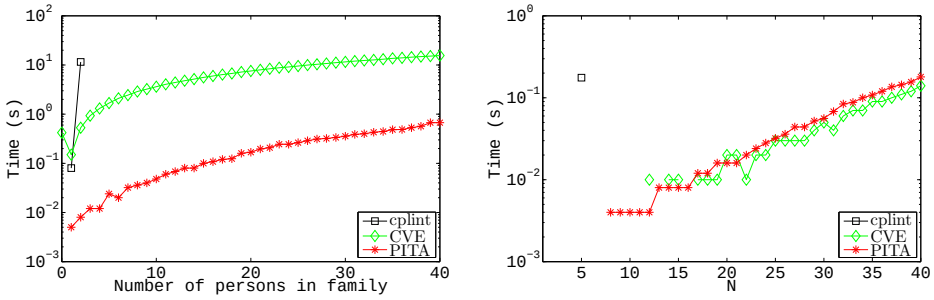
PITA, we disabled automatic reordering of BDDs variables. The execution times of PITA, CVE and `cplint` are shown in Figure 3. In this problem, tabling provides an impressive speedup, since computations can be reused often.

The four datasets of (Meert et al. 2009), containing programs of increasing size. served as a final suite of benchmarks. `bloodtype` encodes the genetic inheritance of the blood type, `growingbody` contains programs with growing bodies, `growinghead` contains programs with growing heads and `uwcse` encodes a university domain. In PITA we disabled automatic reordering of BDDs variables for all datasets except for `uwcse` where we used automatic reordering with the group sift method. The execution times of `cplint`, CVE and PITA are shown respectively in Figures 4(a), 4(b), 5(a) and 5(b)[8]. PITA was faster than `cplint` in all domains and faster than CVE in all domains except `growingbody`. This is notable, as (Meert et al. 2009) found that the variable-elimination based CVS performed better than the BDD-based implementations `cplint` and ProbLog.

9 Conclusion and Future Works

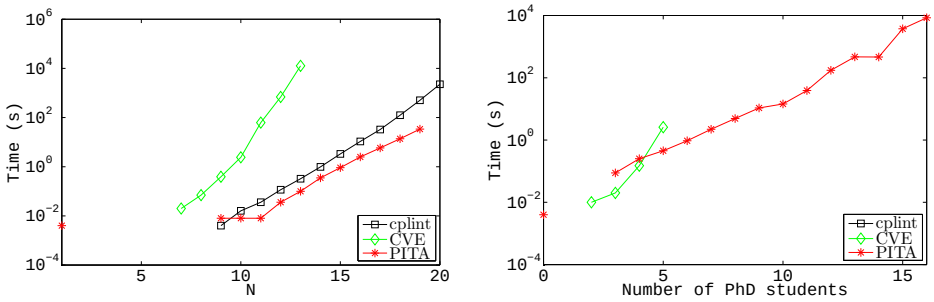
This paper has made two contributions. The first, semantic, contribution extends LPADs to include functions. By way of proving correctness of the PITA transfor-

[8] For the missing points at the beginning of the lines a time smaller than 10^-6 was recorded. For the missing points at the end of the lines the algorithm exhausted the available memory.



(a) `bloodtype`. (b) `growingbody`.

Fig. 4. Datasets from (Meert et al. 2009).



(a) `growinghead`. (b) `uwcse`.

Fig. 5. Datasets from (Meert et al. 2009).

mation we also characterize those extended LPAD programs whose derived atoms all have explanations (finitary LPADs); by way of proving correctness of PITA evaluation we characterize those that have only finite sets of explanations (LPADs with the bounded term depth property). Such results may be used to guide termination and other analysis for tabled evaluation of LPADs with functions.

The extension of LPADs to functions was directly used in the experiments of Section 8. The experiments substantiate the PITA approach which uses BDDs together with tabling with answer subsumption. PITA outperformed `cplint`, CVE and ProbLog in expressiveness, scalability or speed in almost all domains considered. The implementation of PITA is greatly simplified by its use of answer subsumption, which is a comparatively easy extension to an engine that already performs tabling. Accordingly PITA programs should be easily portable to other tabling engines such as that of YAP, Ciao and B Prolog if they support answer subsumption over general semi-lattices.

In the future, we plan to extend PITA to the whole class of sound LPADs by implementing the SLG DELAYING and SIMPLIFICATION operations for answer subsumption. In addition, we plan to develop a version of PITA that is able to answer queries in an approximate way, similarly to (Kimmig et al. 2008).

References

Chen, W. and Warren, D. S. 1996. Tabled evaluation with delaying for general logic programs. *J. ACM 43,* 1, 20–74.

Dantsin, E. 1991. Probabilistic logic programs and their semantics. In *Russian Conference on Logic Programming*. LNCS, vol. 592. Springer, 152–164.

De Raedt, L., Demoen, B., Fierens, D., Gutmann, B., Janssens, G., Kimmig, A., Landwehr, N., Mantadelis, T., Meert, W., Rocha, R., Santos Costa, V., Thon, I., and Vennekens, J. 2008. Towards digesting the alphabet-soup of statistical relational learning. In *NIPS*2008 Workshop on Probabilistic Programming*.

De Raedt, L., Kimmig, A., and Toivonen, H. 2007. ProbLog: A probabilistic Prolog and its application in link discovery. In *International Joint Conference on Artificial Intelligence*. 2462–2467.

Kameya, Y. and Sato, T. 2000. Efficient EM learning with tabulation for parameterized logic programs. In *Computational Logic*. LNCS, vol. 1861. Springer, 269–284.

Kimmig, A., Gutmann, B., and Santos Costa, V. 2009. Trading memory for answers: Towards tabling ProbLog. In *International Workshop on Statistical Relational Learning*. KU Leuven, Leuven, Belgium.

Kimmig, A., Santos Costa, V., Rocha, R., Demoen, B., and De Raedt, L. 2008. On the efficient execution of ProbLog programs. In *International Conference on Logic Programming*. LNCS, vol. 5366. Springer, 175–189.

Kolmogorov, A. N. 1950. *Foundations of the Theory of Probability*. Chelsea Publishing Company, New York.

Meert, W., Struyf, J., and Blockeel, H. 2009. CP-Logic theory inference with contextual variable elimination and comparison to BDD based inference methods. In *International Conference on Inductive Logic Programming*. KU LEuven, Leuven, Belgium.

Muggleton, S. 2000. Learning stochastic logic programs. *Electron. Trans. Artif. Intell. 4,* B, 141–153.

Ng, R. and Subrahmanian, V. S. 1992. Probabilistic logic programming. *Inf. Comput. 101,* 2, 150–201.

Poole, D. 1993. Probabilistic horn abduction and Bayesian networks. *Artif. Intell. 64,* 1, 81–129.

Poole, D. 2000. Abducing through negation as failure: stable models within the independent choice logic. *J. Log. Program. 44,* 1-3, 5–35.

Przymusinski, T. 1989. Every logic program has a natural stratification and an iterated least fixed point model. In *Symposium on Principles of Database Systems*. ACM Press, 11–21.

Riguzzi, F. 2007. A top down interpreter for LPAD and CP-logic. In *Congress of the Italian Association for Artificial Intelligence*. Number 4733 in LNAI. Springer, 109–120.

Riguzzi, F. 2008. Inference with logic programs with annotated disjunctions under the well founded semantics. In *International Conference on Logic Programming*. Number 5366 in LNCS. Springer, 667–771.

Sagonas, K., Swift, T., and Warren, D. S. 2000. The limits of fixed-order computation. *Theor. Comput. Sci. 254,* 1-2, 465–499.

Sato, T. 1995. A statistical learning method for logic programs with distribution semantics. In *International Conference on Logic Programming*. 715–729.

Shapiro, E. Y. 1983. Logic programs with uncertainties: a tool for implementing rule-based systems. In *International Joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., 529–532.

Swift, T. 1999a. A new formulation of tabled resolution with delay. In *Recent Advances in Artifiial Intelligence*. LNAI, vol. 1695. 163–177.

Swift, T. 1999b. Tabling for non-monotonic programming. *Ann. Math. Artif. Intell. 25,* 3-4, 201–240.

Thayse, A., Davio, M., and Deschamps, J. P. 1978. Optimization of multivalued decision algorithms. In *International Symposium on Multiple-Valued Logic*. IEEE Computer Society Press, Los Alamitos, CA, USA, 171–178.

van Emden, M. H. 1986. Quantitative deduction and its fixpoint theory. *J. Log. Program. 30,* 1, 37–53.

van Gelder, A., Ross, K., and Schlipf, J. 1991. Unfounded sets and well-founded semantics for general logic programs. *J. ACM 38,* 3, 620–650.

Vennekens, J., Denecker, M., and Bruynooghe, M. 2009. CP-logic: A language of causal probabilistic events and its relation to logic programming. *Theory Pract. Log. Program. 9,* 3, 245–308.

Vennekens, J., Verbaeten, S., and Bruynooghe, M. 2004. Logic programs with annotated disjunctions. In *International Conference on Logic Programming*. LNCS, vol. 3131. Springer, 195–209.

Appendix A Proof of Theorems 1 and 2

Before proving the theorems, we discuss some of the concepts on which they are based. First, the theorems below use the notion of a finitary program.

Definition 4 (*Finitary Programs*)
A *finitary* normal program P is one in which any atom in $WFM(P)$ (Definition 3) has a finite stratum δ, and for which each stratum $\beta \leq \delta$ was computed by a finite number of applications of the of $\mathcal{T}_I$ and $\mathcal{F}_I$ operators (Definition 2).

An LPAD T is finitary if each of its worlds ($w \in \mathcal{W}_\mathcal{T}$) is finitary.

While the property of being finitary is clearly undecidable, note that a finitary program P need not have a finite ground instantiation nor a finite depth (cf. Definition 3). However in a finitary program, a ground atom a can be seen to be true or false after a finite bottom-up derivation. Since we consider only (subsets of) dynamically stratified programs, the question of determining that an atom is undefined in a finitary program does not arise here.

Second, the theorems make explicit mention of BDD data structures. Note that there are a number of ways to represent BDDs as ground terms, although of course a ground term representation of a BDD will be less efficient than that used by CUDD or other packages. While the proofs below all assume that a BDD is a ground term, they do not need to further specify its form. Accordingly the BDD operations used in the PITA transformation: *and/3*, *or/3*, *not/2*, *one/1*, *zero/1*, and *equality/3* are all taken as (infinite) relations on terms, so that these predicates can be made part of a program's ground instantiation in the normal way. As a result, the ground instantiation of $PITA(T)$ instantiates all variables in T with all BDD terms.

Finally, the definition of *get_var_n/4* from Section 7:

$$get\_var\_n(RuleName, Vars, Probs, RV) \leftarrow$$
$$(var(RuleName, Vars, RV) \rightarrow true;$$
$$length(Probs, L), add\_var(L, Probs, RV), assert(var(RuleName, Vars, RV))).$$

uses a non-logical update of the program, and so without modifications, it is not suitable for our proofs below. Alternately, we assume that $ground(T)$ is augmented with a (potentially infinite) number of facts of the form $var(RuleName, [], RV)$ for each ground rule $RuleName$ (note that no variable instantiation is needed in the second argument of $var/3$ if it is indexed on ground rule names). Clearly, the augmentation of T by such facts has the same meaning as *get_var_n/4*, but is simply done by a priori program extension rather than during the computation as in the implementation.

Lemma 1
If T is a finitary LPAD, then $PITA(T)$ is finitary, and for each $a \in T$, if a is true in $WFM(T)$, a has an explanation.

Proof
If T is finitary, each world in $\mathcal{W}_T$ is finitary, by Definition 4. By Definition 2 there are interpretations I' and T' such that $a \in TRUE_{I'}(T')$. Since I' an T' were both produced by a finite number of iterations of the $TRUE_I$ and $FALSE_I$ operators,

a "depends" (e.g. in the sense of local stratification) on the truth value of a finite set $\mathcal{S}$ of atoms. Since any world in which $\mathcal{S}$ is true implies that a is true, $\mathcal{S}$ is an explanation for a.

By the preceding discussion, the extra literals introduced into the bodies of rules by the $PITA$ transformation are all facts. Accordingly, for e.g. $and/3$, $and/3$ atoms that are in the $and/3$ relation are true at all points in the well-founded computation after $TRUE_\emptyset(\emptyset)$, and all $and/3$ atoms that are in *not* the $and/3$ relation are false at all points after $FALSE_\emptyset(\emptyset)$. Accordingly, derivation of an atom $PITA(a)$ will take a finite number of steps (i.e. applications of the $TRUE$, $FALSE$ and WFM operators) in the computation of $PITA(T)$ if a takes a finite number of steps in the computation of T, so that $PITA(T)$ is finitary. □

Theorem 1 *Let T be a sound finitary LPAD. Then σ is an explanation for $a \in ground(T)$ iff there is some $PITA(a)\theta$ in $WFM(PITA(ground(T)))$, such that σ is a path in $BDD(PITA(a)\theta)$.*

Proof
(Soundness $\Leftarrow$) The proof is by outer induction on the operator WFM (Definition 3) and inner induction on the operators $TRUE_I$ and $FALSE_I$ (Definition 2). Within each stratum there is induction on the structure of formulas. We note that since T is finitary, we do not need to consider transfinite induction.

1. *Base Case for Outer Induction (WFM Operator).* This case consists of induction for $TRUE_\emptyset$ and $FALSE_\emptyset$

   (a) *Base Case for Inner Induction ($TRUE_\emptyset$ and $FALSE_\emptyset$)* In the case of $TRUE_\emptyset$, a ground atom a is proven using a non-disjunctive fact or disjunctive fact in the T. If a unifies with a non-disjunctive fact f,

   $$PITA(f) = PITA_h(f) \leftarrow one(BDD).$$

   so the BDD associated with a represents the constant 1, i.e., an empty composite choice that is an explanation for a. Alternately, if a unifies with head $h_i(C_r)$ of a disjunctive fact C_r,

   $$PITA(C_r, i) = PITA_h(h_1(C_r)) \leftarrow var(C_r : i, [], Var), equality(Var, i, BDD).$$

   the rule "creates" a new variable Var and a new BDD BDD indicating that the value of Var is i to represent the primitive choice $\{(C_r, \emptyset, i)\}$, which is an explanation for a.
   In the base case of $FALSE_\emptyset$, those atoms *removed* from the greatest fixed point are non-disjunctive facts contained in $ground(T)$.

   (b) *Inductive Case (Inner)* In the case of $TRUE_\emptyset$, the atom a is proven via a rule C_r in $PITA(T)$ that does not contain negation in its body. If C_r is non-disjunctive

   $$\begin{aligned} PITA(C_r) = PITA_h(h) \leftarrow\ & PITA_b(b_1), PITA_b(b_2), \\ & and(BDD(PITA_b(b_1)), BDD(PITA_b(b_2)), BDD_2), \ldots \\ & PITA_b(b_m), and(BDD_{m-1}, BDD(PITA_b(b_m)), BDD). \end{aligned}$$

By the induction assumption, the statement holds for $PITA_b(b_i) : 1 \leq i \leq m$. For $j : 1 \leq j < m$, let BDD_j be the BDD assocated with the $\bigwedge_{1 \leq i \leq j} PITA_b(b_i)$. Then the literal $and(BDD_j, BDD(PITA_b(b_{j+1})), BDD_{j+1})$ produces the BDD BDD_{j+1} that is either the first argument of the next $and/3$ literal in the body, or is the final BDD produced in the body. We now perform structural induction on the body of C_r. For the base case, each path in BDD_1 is a finite explanation by the inner fixed point inductive assumption. For the inductive case, by the structural induction assumption, BDD_k represents a finite explanation for $\bigwedge_{1 \leq i \leq k} PITA_b(b_i)$, and BDD_{k+1} by the inner fixed point inductive case, so that clearly $and(BDD_k, BDD(PITA_b(b_{k+1})), BDD_{k+1})$ produces a finite explanation for $\bigwedge_{1 \leq i \leq k+1} PITA_b(b_i)$.
Alternately, if a is derived via a selection of the i^{th} disjunct of a ground disjunctive clause C_r,

$$\begin{aligned} PITA(C_r, i) = PITA_h(h_i) \leftarrow \quad & PITA_b(b_1), PITA_b(b_2), \\ & and(BDD(PITA_b(b_1)), BDD(PITA_b(b_2)), BDD_2), \ldots, \\ & PITA_b(b_m), and(BDD_{m-1}, BDD(PITA_b(b_m)), BDD_m), \\ & var(C_r : i, [], Var), \\ & equality(Var, i, B), and(BDD_m, B, BDD). \end{aligned}$$

the transformed rule propagates conjunction as for non-disjunctive rules within the present iteration case (1b), and as with disjunctive facts (iteration case 1a) "creates" a new variable Var and a new BDD B indicating that the value of Var is i to represent the primitive choice $\{(C_r, \emptyset, i)\}$, and conjoins B to BDD_m to produce the final BDD for the rule, which is an explanation for a.
In the case of $FALSE_\emptyset$, an atom a removed from the greatest fixed point is such that at least one of one of its clauses does not have any positive body literal in the previous iteration of $FALSE_\emptyset$. As a result a is not an unfounded atom with respect to the interpretation $\emptyset$. This condition is not affected by the PITA transformation.

2. *Inductive Case Outer Induction (WFM Operator)* This case consists of induction for $FALSE_I$ and $FALSE_I$, where I is the interpretation produced by the previous iteration of the WFM operator.

   (a) *Base Case Inner Induction ($TRUE_I$ and $FALSE_I$ Operators)*
   For $TRUE_I$, in the case of non-disjunctive rules, atoms may be added to the interpretation by rules of the form

   $$r = h \leftarrow b_1, \ldots, b_k, not(b_{k+1}), \ldots, not(b_m).$$

   where $b_1, \ldots, b_k$ are true in I, and $b_{k+1}, \ldots, b_m$ are false in I. Such rules are transformed into

   $$\begin{aligned} PITA(C_r) = PITA_h(h) \leftarrow & PITA_b(b_1), \ldots, PITA_b(b_k), \\ & and(BDD_{k-1}, BDD(PITA_b(b_k)), BDD_k) \\ & (PITA_b(b_{k+1}) \rightarrow not(BDD_{b_{k+1}}, B_{l_{k+1}}); one(B_{l_{k+1}})), \\ & and(BDD_k, B_{l_{k+1}}, BDD_{k+1}), \ldots, and(BDD_{m-1}, B_{l_m}, BDD). \end{aligned}$$

   This is similar to non-disjunctive rules seen in inductive case 1b with the

addition of the negative literals that have been transformed into conditionals. If a negative literal $PITA_b(b_{k+1})$ fails, conjoining the BDD 1 with BDD_k in $and(BDD_k, B_{k+1}, BDD_{k+1})$ has no effect on the semantics of BDD_k. If $PITA_b(b_{k+1})$ succeeds, its complement is taken for the conjunction to produce a new BDD whose paths to 1 indicate the consistent composite choices [9].
In the case of disjunctive rules, an atom is added by a rule whose i^{th} disjunct is translated into

$$\begin{aligned} PITA(C_r, i) = PITA_h(h) \leftarrow\ & PITA_b(b_1), \ldots, PITA_b(b_k), \\ & and(BDD_{k-1}, BDD(PITA_b(b_k)), BDD_k) \\ & (PITA_b(b_{k+1}) -> not(BDD_{b_{k+1}}, B_{l_{k+1}}); one(B_{l_{k+1}})), \\ & and(BDD_k, B_{l_{k+1}}, BDD_{k+1}), \ldots, \\ & var(C_r : i, [], Var), equality(Var, i, B), and(BB_m, B, BDD). \end{aligned}$$

This is similar to the previous case, but with the addition of "adding" a new variable Var and a new BDD B indicating that the value of Var is i to represent the primitive choice $\{(C_r, \emptyset, i)\}$, just as for disjunctive rules for induction case (1b). As in that case B is conjoined to BDD_m to produce the final BDD for the rule, which is an explanation for a.
In the base case of $FALSE_I$, those atoms removed from the greatest fixed point are those all of whose rules have at least one body literal that is false in I. This is unaffected by the PITA transformation.

(b) *Inductive Case (Outer)*
For $TRUE_I$, from the perspective of the PITA transformation, the actions in this case are essentially the same as in inductive case 2a.
In the case of $FALSE_I$, an atom a removed from the greatest fixed point is such that at least one of one of its clauses does not have either any positive body literal in the previous iteration of $FALSE_I$ or a body literal that is false in I. As a result a is not an unfounded atom with respect to the interpretation I This condition is not affected by the PITA transformation.

(Completeness $\Rightarrow$) Sketch

The completeness proof of Theorem 1 maps interpretations over a LPAD T to $PITA(T)$. To accomplish this, note that the PITA transformation corresponds to the identity transformation for atoms whose predicate symbol is *and/3*, *or/3*, *not/2*, *one/1*, *zero/1*, *equality/3* and *var/3*.

For completeness we must prove that if σ is an explanation for $a \in ground(T)$ then there is some instantiation $PITA(a)\theta \in WFM(PITA(ground(T)))$ of $PITA(a)$ such that such that σ is a path in $BDD(PITA(a)\theta)$ [10].

Let $h \leftarrow body$ be a non-disjunctive clause in T. If $body$ is such that $TRUE_I(T')$ for interpetations I and T', then it is straightforward to show that $Pita(body)$ is true in $PITA(I)$ and $PITA(T)$ – since the extra literals in $Pita(body)$ such as *and/3*, *one/1*, etc. are added to these interpretations (cf the proof of Lemma 1).

[9] Technically speaking, the Prolog conditional $(A \rightarrow B; C)$ must first be translated to $(A, B; \neg A, C)$ and this form translated into clauses without body disjuncts.
[10] The θ substitution binds the variable introduced by $PIA_h(a)$ to a BDD-term.

The same argument holds for the body of a disjunctive clause

$$C = h_1 : \alpha_1 \vee \ldots \vee h_n : \alpha_n \leftarrow b_1, \ldots, b_m.$$

In addition, note that if (C, θ, i) is a primitive selection used to construct σ, then the PITA transformation ensures that a rule is generated for h_i which is then instantiated to all possible instantiations, including the instantiation θ mentioned above.

Extending this to a full proof would involve placing these observations within the structure of a double induction like that used for Soundness. Note that because of Lemma 1 no transfinite induction would be needed for this induction. However the main ideas used in the cases of such an induction are simply those just mentioned combined with the analysis of the different cases of the PITA transformation shown for Soundness. □

Definition 5
A normal program P has the *bounded term size property* if there is a finite function $f : N \rightarrow N$ such that any if a query term Q to P has size $size(Q)$ then no term used in the derivation of Q has size greater than $f(Q)$. An LPAD T has the bounded term size property if each $w \in W_T$ has the bounded term size property.

Note that the query Q in Definition 5 need not be ground. From Definition 4 a program that is finitary is such that for each *ground* query Q a bounded-term-size proof could be constructed (simply by computing the well-founded model until Q is seen to be true or false). Thus a finitary program is a bounded-term-size program, but the converse is false.

Theorem 2 *Let T be a range-restricted, bounded-term-depth, fixed-order dynamically stratified LPAD and a a ground atom. Let $\mathcal{E}$ be an SLG evaluation of $PITA(a)$ against $PITA(T)$, such that answer subsumption is declared on $PITA(a)$ using BDD-disjunction. Then $\mathcal{E}$ terminates with an answer ans for $PITA(a)$ and $BDD(ans)$ represents the set of all covering explanation for a.*

Proof
(Sketch) The proof uses the forest-of-trees model (Swift 1999a) of SLG (Chen and Warren 1996).

Because T is fixed-order dynamically stratified, queries to T can be evaluated using SLG without the DELAYING, SIMPLIFICATION or ANSWER COMPLETION oprations.. Instead, as (Sagonas et al. 2000) shows, only the SLG operations NEW SUBGOAL, PROGRAM CLAUSE RESOLUTION, ANSWER RETURN and NEGATIVE RETURN are needed. However, because $\mathcal{E}$ uses answer subsumption, the SLG an extension to the ANSWER RETURN operation must also be defined. The operation assumes that answer subsumption on an upper semi-lattice L (implicitly on the last argument of the selected literal).

- SUBSUMING ANSWER RETURN: Let an SLG forest $\mathcal{F}_n$ contain a non-root node

$$N = Ans \leftarrow S, GoalList$$

whose selected literal S is positive and has been declared to use answer subsumption over a lattice L. Further, assume that S is ground in all arguments except its last, V_n, which is unbound. Let $\mathcal{A}$ be the set of all answers for S in $\mathcal{F}_n$, and $Join$ be the L-join of all the final arguments of all answers in $\mathcal{A}$. Assume that in $\mathcal{F}_n$, N does not have a child $S\{V_n/Join\}$. Then add $S\{V_n/Join\}$ as a child of N.

For the proof, the first item to note is that since T is range-restricted, it is straightforward to show that $PITA(T)$ is also range-restricted; that any subgoal $PITA(a)$ will be ground except for its last argument, which is free; and that all answers in $PITA(a)$ will be ground (cf. (Muggleton 2000)). Accordingly the operation SUBSUMING ANSWER RETURN will be applicable to any subgoal with a non-empty set of answers.

Next, since T has the bounded term size property, any SLG evalation of a query Q to T will terminate (Chen and Warren 1996), and it is straightforward to extend this result to SLG evaluations extended with answer subsumption.

It remains to show that answer ans for $PITA(a)$ such that $BDD(ans)$ represents the set of all covering explanations for a. The bounded term size property of T implies that there will only be a finite set of explanations (otherwise $f(Q)$ would be unbounded). Furthermore, each explanation in the set is finite, as follows from the fact that T is finitary (as implied by the bounded term size property).

That $BDD(ans)$ contains all and only covering explanations can be shown by induction on the number of join (BDD-or) operations. For the inductive assumption, BDD_{i-1} and BDD_i both represent finite sets of covering finite explanations covering for b_{i-1} and h_I respectively. Let F_{i-1}, F_i, and $F_{i'}$ be the formulas expressed by BDD_{i-1}, BDD_i, and $BDD_{i'}$ respectively. These formulas can be represented in disjunctive normal form, in which every disjunct represents an explanation. $F_{i'}$ is obtained by multiplying F_{i-1} and $F_{i'}$ so, by algebric manipulation, we can obtain a formula in disjunctive normal form in which every disjunct is the conjunction of two disjuncts, one from F_{i-1} and one from F_i. Every disjunct is thus an explanation for $F_{i'}$ – the body prefix upto and including b_i. Moreover, every explanation for $F_{i'}$ is obtained by conjoining an explanations for F_{i-1} with an explanation for F_i, so it is represented by a disjunct of $F_{i'}$. □