Journal of Intelligent Information Systems (2018) 51: 647–672
https://doi.org/10.1007/s10844-018-0531-6

Extending expressivity and flexibility of abductive logic programming

Stefano Ferilli[1,2]

Received: 6 July 2017 / Revised: 29 August 2018 / Accepted: 26 September 2018 /
Published online: 9 October 2018

Abstract
Real-world problems often require purely deductive reasoning to be supported by other techniques that can cope with noise in the form of incomplete and uncertain data. Abductive inference tackles incompleteness by guessing unknown information, provided that it is compliant with given constraints. Probabilistic reasoning tackles uncertainty by weakening the sharp logical approach. This work aims at bringing both together and at further extending the expressive power of the resulting framework, called Probabilistic Expressive Abductive Logic Programming (PEALP). It adopts a Logic Programming perspective, introducing several kinds of constraints and allowing to set a degree of strength on their validity. Procedures to handle both extensions, compatibly with standard abductive and probabilistic frameworks, are also provided.

Keywords Abductive logic programming · Probability · Constraints

1 Introduction

Two, different and complementary, directions have traditionally characterized research in Artificial Intelligence (AI): the numerical/statistical one, on one hand, and the symbolic/logic one, on the other. The former is efficient and tolerant to noise, but unable to capture the complex network of relationships among different events and objects. The latter, using the First-Order Logic (FOL) setting, allows one to handle relational representations of the data, but requires that data are certain and, for purely deductive inference, complete. Unfortunately, the real world often involves, among others, two kinds of noise in the available information: incompleteness and uncertainty. So, reasoning in these contexts requires

✉ Stefano Ferilli
stefano.ferilli@uniba.it

[1] Dipartimento di Informatica – Università di Bari, Bari, Italy

[2] Centro Interdipartimentale per la Logica e sue Applicazioni – Università di Bari, Bari, Italy

purely deductive reasoning to be supported by techniques that can cope with them. Specifically, the former is the domain of abductive reasoning, classically intended as a purely logical approach, while the latter is the domain of probabilistic reasoning, for which several solutions have been proposed in the literature.

As regards uncertainty, since several decades now (Nilsson 1986), much research has been carried out to put together logical and statistical inference, so that the former provides the representation language and high-level reasoning strategies, and the latter enforces robustness. In particular, many works have considered the *Probabilistic Logic Programming* (PLP) setting (De Raedt and Kersting 2008) for the logic side, while others have adopted the *Statistical Relational Learning* (SRL) setting (Getoor 2002).

On the other hand, abductive inference can tackle incompleteness in the data by allowing to guess information that has not been observed. Using a classical example, if someone enters in our room and his shoes are wet, we might guess that it's raining outside. However, there may be many other plausible explanations for this event (e.g., he crossed a lawn while the sprinkler was on), and thus that inference provides no certainty on the real cause of the 'wet shoes' event. While humans are able to distinguish explanations that are consistent with their previous knowledge and discard the others, embedding this capability in machines is not easy. One proposal is known as the Abductive Logic Programming (ALP) framework (Kakas et al. 1992).

This paper builds upon the ALP framework, both because it provides a direct and explicit way of having abductive reasoning and explanations, and because Kakas et al. (1992) provide relevant justifications for which ALP is profitable, and show it has several advantages. Specifically, this work provides two original contributions to ALP:

1. it extends the expressiveness of the framework, by allowing one to use several kinds of integrity constraints and to express a degree of uncertainty on them;
2. it provides suitable procedures to carry out abductive reasoning and to handle uncertainty in both the basic and the extended framework.

While the extensions concerning generalized constraints and uncertainty are independent from each other, and might be introduced separately, an added value of this work is to provide an integrated framework that, leveraging their cooperation, allows the representation and handling of more complex cases.

In the integrated approach, the uncertainty handling part is used to assign a degree of confidence to the alternative abductive explanations of a given goal obtained by the abductive part, and in case to choose the more reliable one(s). The integrated reasoning approach is compatible with other non-probabilistic approaches to abductive reasoning and probabilistic approaches to deductive reasoning. I.e., removing the probabilities and/or the constraints from our framework, and applying the proposed approach, we obtain the same results as other existing frameworks in the literature.

The paper is organized as follows. The next section introduces the basic ALP framework and reviews related work, especially focusing on past attempts to join abductive and probabilistic reasoning. Then, Section 3 proposes an extension of the framework by means of generalized integrity constraints. Section 4 further extends the framework in order to deal with uncertainty. Finally, Section 5 concludes the paper.

2 (Probabilistic) abductive logic programming

Abduction is the inference strategy devoted to cope with missing information. For our purposes, we adopt the Logic Programming setting (Lloyd 1987), which is the traditional machinable fragment of FOL.

2.1 Logic programming basics

This section recalls the basics of Logic Programming that are needed for this work. For more details, abbreviations and theoretical results the reader is referred to Lloyd (1987).

Logic programs consist of *Horn clauses*, i.e. logic formulas of the form $l_0 \leftarrow l_1, \ldots, l_n$, denoting implications $l_1 \wedge \cdots \wedge l_n \Rightarrow l_0$. In FOL all variables appearing in a clause are universally quantified. l_0 is called the *head* of the clause, and $l_1, \ldots, l_n$ is called the *body* of the clause. A clause having both the head and the body is a *rule*; a clause l_0 having only the head is a *fact*; a clause $\leftarrow l_1, \ldots, l_n$ having only the body is a *goal*; the *empty clause* $\square$, having no head nor body, denotes a contradiction. The l_i's ($i = 0, \ldots, n$) are *literals*, i.e. atoms or negated atoms (the negation of an atom a will be denoted, in the following, by $\neg a$ or by $\overline{a}$). In turn, an atom is a predicate applied to terms as arguments. While in FOL, in general, terms may be constants, variables, or n-ary function symbols applied to n terms as arguments, here we will focus on Datalog, a fragment of Horn clause logic where terms can be only constants or variables. Specifically, in the following, variables will be denoted by uppercase symbols, while constants (and predicate names) will be denoted by lowercase symbols. An atom, literal or clause is *ground* if it contains only constants as terms.

Example 1 A logic program in the domain of family relationships might be:

$$
\begin{aligned}
P = \{\ & R_1 : \text{son}(X, Y) \leftarrow \text{parent}(Y, X), \text{male}(X) \\
& R_2 : \text{daughter}(X, Y) \leftarrow \text{parent}(Y, X), \text{female}(X) \\
& F_1 : \text{parent(carl,steve)} \\
& F_2 : \text{parent(ann,steve)} \\
& F_3 : \text{parent(john,ann)} \\
& F_4 : \text{male(steve)}\}
\end{aligned}
$$

where the R_i's are rules, and the F_j's are facts.

Deductive inference in Logic Programming is carried out based on the *SLD-resolution* inference rule. In a nutshell, given two clauses $C' : l'_0 \leftarrow l'_1, \ldots, l'_n$ and $C'' : l''_0 \leftarrow l''_1, \ldots, l''_m$ and a substitution (i.e., a mapping from variables to terms) θ such that $l'_i\theta = l''_0\theta$ for some $i \in \{1, \ldots, n\}$, then the clause $C : (l'_0 \leftarrow l'_1, \ldots, l'_{i-1}, l''_1, \ldots, l''_m, l'_{i+1}, \ldots, l'_n)\theta$, obtained by "*resolving* C' *on* literal l'_i", is a logical consequence of C' and C'' (where applying a substitution θ to a logic formula means simultaneously replacing each variable in the formula with the corresponding term in θ, if any). C is called the *resolvent* of C' and C''. More specifically, proofs proceed by *refutation*: given an existentially quantified conjunction of atoms to be proved $l_1 \wedge \cdots \wedge l_n$ ($n > 0$), it is negated, obtaining a goal $\leftarrow l_1, \ldots, l_n$; if adding this goal to a program P one may obtain the empty clause by repeated application of SLD-resolution steps, then the conjunction is proved in P. In such a case, the cumula-

tive substitution obtained by composing the substitutions obtained in each resolution step is called a *computed answer substitution*, and determines a (set of) instance(s) for which the conjunction is true. The case in which negated atoms appear in the body of clauses is the domain of *general* logic programs. To prove such negated atoms, in this paper we use the *Negation as Failure* (NAF) rule (Clark 1978) (if an atom cannot be proven by refutation, then its negation is considered true), and the corresponding proof procedure (known as *SLD-NF resolution*).

Example 2 In Example 1, *son(steve,carl)* can be proven as follows:

← son(steve,carl) resolved with R_1 yields, by $\theta = \{steve/X, carl/Y\}$
← parent(carl,steve), male(steve) that, resolved with F_1 yields, by the empty substitution $\epsilon = \{\}$,
← male(steve) that, resolved with F_4 yields, by the empty substitution $\epsilon = \{\}$,
□ , the empty clause, which proves the initial goal.

Conversely, due to the missing fact *female(ann)*, the goal *daughter(ann,john)* cannot be proven, and thus is considered false by NAF.

2.2 Abductive logic programming framework

Abductive Logic Programming (Kakas and Mancarella 1990a, b), or *ALP* for short, is a well-known framework for abductive reasoning based on Logic Programming. More precisely, it extends Logic Programming by considering some predicates, called *abducibles*, to be only indirectly and partially defined by means of a set of *constraints*. Problem solving is performed by allowing the reasoner to derive *abductive hypotheses* about instances of these abducible predicates, as long as such hypotheses do not violate the given constraints. Problems can be observations that need to be explained (as in classical abduction) or goals to be proven (as in standard Logic Programming).

Kakas et al. (1992) provide an extensive account of ALP, discussing several semantics and proof procedures, and their connections to other related problems and tasks. In particular, they show how many approaches proposed in the literature for Knowledge Assimilation, Truth Maintenance and, especially, Default Reasoning can be mapped onto ALP. A large part of their discussion focuses on the role and manipulation of negation, introducing several options among which NAF receives a particular emphasis.

Definition 1 (Abductive Logic Program) An *abductive logic program* (or *abductive theory*) consists of a triple $\langle P, A, I \rangle$, where:

P is a general logic program;
A (Abducible predicates) is a set of predicates;[1]

[1]By extension, according to foundational literature (Kakas et al. 1992), literals built on abducible predicates are also called *abducibles*, meaning that they can be abduced. So, an abducible predicate is a kind of claims that may be abduced, while an abducible literal is a specific claim of that kind that may be abduced.

I (Integrity Constraints, ICs for short) is a set of formulas that must be satisfied by the abductive hypotheses.[2]

Again, let us show this through a simple example.

Example 3 An abductive logic program in the family domain might be:
P : as in Example 1
$A = \{male/1, female/1\}$
$I = \{\leftarrow male(X), female(X)\}$ (i.e., any X cannot be both male and female)

The three components of an abductive theory are used to define abductive explanations.

Definition 2 (Abductive explanation) Given an abductive theory $T = \langle P, A, I\rangle$ and a formula G, an *abductive explanation* Δ for G is a set of ground literals of predicates in A s.t. $P \cup \Delta \models G$ (Δ explains G) and $P \cup \Delta \models I$ (Δ is consistent). When it exists, T *abductively entails* G, in symbols $T \models_A G$.

Several proof procedures have been proposed in the literature to obtain abductive explanations for abductive logic programs, also in the presence of negated literals in the body (Denecker and Schreye 1992). Here we follow the one proposed by Kakas and Mancarella (1990c). It takes ICs in the form of plain denials/nogoods, and interleaves *abductive* and *consistency derivations*. An *abductive derivation* is the standard Logic Programming derivation extended in order to consider abducibles. When an abducible literal δ has to be proved, it is added to the current set of hypotheses, provided that no integrity constraint involving δ is violated (i.e., that all such denials fail). To check this, a *consistency derivation* is started. In turn, *consistency derivations* use abductive derivations to solve their goals, which may require/provide additional abductive hypotheses. We recall in the following the two procedures, in a slightly modified version that preserves the same meaning and behavior.

Definition 3 (Abductive derivation) An *abductive derivation* from $(G_1\ \Delta_1)$ to $(G_n\ \Delta_n)$ in $\langle P, A, I\rangle$ is a sequence

$$(G_1\ \Delta_1), (G_2\ \Delta_2), ..., (G_n\ \Delta_n)$$

such that each G_i has the form $\leftarrow l_1, ..., l_{k_i}$ and $(G_{i+1}\Delta_{i+1})$ is obtained by selecting an l_j from G_i and applying one of the following rules:[3]

1. If l_j is not abducible and G_i can be resolved on l_j with some clause in P, obtaining C as the resolvent, then $G_{i+1} = C$ and $\Delta_{i+1} = \Delta_i$;
2. If l_j is abducible and $l_j \in \Delta_i$, then $G_{i+1} = \leftarrow l_1, ..., l_{j-1}, l_{j+1}, ..., l_k$ and $\Delta_{i+1} = \Delta_i$;

[2]While generally defining ICs as formulas, with a few exceptions the discussion in Kakas et al. (1992) only considers ICs in the form of 'plain' *denials* or, using Truth Maintenance terminology, *nogoods*, i.e., negations of conjunctions of literals, possibly due to their straightforward representation as Logic Programming goals ($\leftarrow l_1, \ldots, l_n, \neg l_{n+1}, \ldots, \neg l_{n+m}$). As such, since goals are clauses, and FOL clauses are universally quantified, ICs are universally quantified in FOL, as well.

[3]The original framework requires that abducible predicates have no definition in P. This requirement may be relaxed with a simple representational trick.

3. If l_j is a ground abducible, $l_j \notin \Delta_i$ and $\overline{l_j} \notin \Delta_i$ and there exists a *consistency derivation* from $(l_j\ \Delta_i)$ to $(\square\ \Delta')$ then $G_{i+1} = \leftarrow l_1, ..., l_{j-1}, l_{j+1}, ..., l_k$ and $\Delta_{i+1} = \Delta'$

Rules (1) and (2) apply standard resolution, using (1) a clause in P or (2) a previous abductive hypothesis. Rule (3) extends the current set of hypotheses, provided that a consistency check is passed.

Definition 4 (Consistency derivation) A *consistency derivation* for an abducible α from (α, Δ_0) to $(F_n\ \Delta_n)$ in $\langle P, A, I\rangle$ is a sequence

$$(\alpha\ \Delta_0), (F_1\ \Delta_1), ..., (F_n\ \Delta_n)$$

where:

1. F_1 is the set of all non-empty goals of the form $\leftarrow l_1, ..., l_n$ obtained by resolving the abducible α with the constraints in I, $\Delta_1 = \Delta_0 \cup \{\alpha\}$;
2. for all $i > 1$ let $F_i = \{\leftarrow l_1, ..., l_k\} \cup F'_i$ with $F'_i = F_i \setminus \{\leftarrow l_1, ..., l_k\}$, then for some $j = 1, ..., k$ $(F_{i+1}\ \Delta_{i+1})$ is obtained according to one of the following rules:
   (a) If l_j is not abducible, then $F_{i+1} = C' \cup F'_i$ where C' is the set of all resolvents of clauses in P with $\leftarrow l_1, ..., l_k$ on literal l_j (none of which must be empty), and $\Delta_{i+1} = \Delta_i$
   (b) If l_j is abducible, $l_j \in \Delta_i$ and $k > 1$, then $F_{i+1} = \{\leftarrow l_1, ..., , l_{j-1}, l_{j+1}, ..., l_k\} \cup F'_i$ and $\Delta_{i+1} = \Delta_i$
   (c) If l_j is abducible, $\overline{l_j} \in \Delta_i$ then $F_{i+1} = F'_i$ and $\Delta_{i+1} = \Delta_i$
   (d) If l_j is a ground abducible, $l_j \notin \Delta_i$ and $\overline{l_j} \notin \Delta_i$ and there exists an *abductive derivation* from $(\leftarrow \overline{l_j}\ \Delta_i)$ to $(\square\ \Delta')$ then $F_{i+1} = F'_i$ and $\Delta_{i+1} = \Delta'$;
   (e) If $l_j = \overline{a}$, with a a ground atom, and there exists an *abductive derivation* from $(\leftarrow a\ \Delta_i)$ to $(\square\ \Delta')$ then $F_{i+1} = F'_i$ and $\Delta_{i+1} = \Delta'$.

The first option (2a) replaces the current branch $\leftarrow l_1, ..., l_k$ with all of its resolvents with clauses in P on l_j. If one such resolvent is the empty clause the whole check fails, because the constraint was proved (i.e., the denial could not be falsified). The second option (2b) is similar, but using an already abduced literal. In case (2c) the current branch is falsified on l_i by the current hypotheses in Δ_i, so it is dropped. The last two cases (2d) and (2e) try to falsify (and drop) the current branch through abductive falsification of l_j.

The above procedure returns the minimal abductive explanation set if any, otherwise it fails. It requires the Δs in the abductive explanations to be ground. Thus, non-ground abducibles are resolved with a previously abduced literal, or, if this is not possible, they are grounded using skolem constants.

A goal G can be abductively proven, using hypotheses Δ, if there exists an abductive derivation from $(G\ \emptyset)$ to $(\square\ \Delta)$. Sometimes, an abductive proof must be consistent with a set of previous abductions Δ_I, in which case there must exist an abductive derivation from $(G\ \Delta_I)$ to $(\square\ \Delta)$. Let us define the following function, that returns the set of abductive explanations and associated computed answer substitutions for a goal G in an abductive

theory T, consistent with a set of previous abductions Δ_I: $\text{abd}_T(G, \Delta_I) \rightarrow \{(\theta, \Delta_O)|$ there exists an *abductive derivation* in T from $(G\ \Delta_I)$ to $(\square\ \Delta_O)$ with computed answer substitution θ }

2.3 Related work

While the classical ALP framework considers integrity constraints that are denials, other abductive frameworks proposed different kinds of integrity constraints. E.g., IFF (Fung and Kowalski 1997) and its extensions are based on integrity constraints that are clauses (i.e., implications with conjunctive premises and disjunctive conclusions). Compared to them, here we define a wider set of kinds of integrity constraints, inspired by logical operators.

Probabilistic reasoning has been embedded in logic in different ways in the literature. From a Logic Programming perspective, examples are Stochastic Logic Programs, or SLPs (Muggleton 1996), Logic Programs with Annotated Disjunctions, or LPADs (Vennekens et al. 2004), and ProbLog (De Raedt et al. 2007). Some works explicitly addressed abductive reasoning.

One of the earliest approaches (Poole 1993) focuses on the representation language. A program contains non-probabilistic definite clauses and a set of probabilistic disjoint declarations $\{h_i : p_i\}_{i=1,\ldots,n}$ meaning that an abducible atom h_i is true with probability p_i. It has several shortcomings: no structured constraints (just disjoint declarations); no integrated logic-based abductive proof procedure with statistical learning; no probabilities assigned to constraints (just to ground literals).

PRISM (Sato 2002) is a system based on logic programming with multivalued random variables. It provides no support for integrity constraints but 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. Interestingly, PRISM can learn probabilities from training data.

Two approaches have merged directed and undirected graphical models with logic. The former, by Raghavan (2011), exploits Bayesian Logic Programs (BLPs) (De Raedt and Kersting 2008) as a representation language for abductive reasoning and uses the Expectation Maximization algorithm to learn the parameters associated to the model. The latter, by Kate and Mooney (2009), exploits Markov Logic Networks (MLN) (Richardson and Domingos 2006). 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 expensiveness. Like MLNs, most SRL formalisms use deduction for logical inference, and so they cannot be used effectively for abductive reasoning.

In some solutions, the cooperation between the two approaches aimed at ranking all possible explanations in order to choose the best one.

An approach for probabilistic abductive logic programming with Constraint Handling Rules was proposed by Christiansen (2008). Differently from other approaches to probabilistic logic programming, it provides both interaction with external constraint solvers and integrity constraints. Unfortunately, to optimize the search for explanations using Dijkstra's shortest path algorithm, it always explores the most probable direction, so that the investigation of less probable alternatives is suppressed or postponed. It also has other shortcomings.

First, it cannot handle negation, that must be simulated by normal predicate symbols (e.g., *not_p(X)* for $\neg p(X)$). As a consequence, the definition of the constraints might be tricky.

Arvanitis et al. (2006) adopt SLPs, considering a number of *possible worlds*. Abductive reasoning is carried out by reversing the deductive flow of proof and collecting the probabilities associated to the involved clauses. Although this approach is probabilistically consistent with the SLP language, it is quite hazardous because abductive reasoning by means of deduction without constraints may lead to wrong conclusions.

cProbLog (Fierens et al. 2012) extends regular ProbLog logic programs, where facts in the program can be associated to (independent) probabilities, to consider integrity constraints. It comes with a formal semantics and computational procedures, resulting in a powerful framework that encompasses PLP (ProbLog) and SRL (MLNs), taking the best of both. Since in cProbLog constraints are sharp, it adopts the same probability count as ProbLog, but ignoring all worlds that do not satisfy the constraints.

More recently, the idea of probabilistic integrity constraints is present in the work by Alberti et al. (2016), that also proposes an associated distribution semantics. However, they adopt the same kind of integrity constraints as IFF (i.e., clauses), and consider theories made up of constraints only, while here we still work on logic programs.

The approach proposed in this paper advances the state-of-the-art because it has the following main features, that are never present altogether in the proposals discussed above:

1. it allows to express new and intuitive kinds of integrity constraints, allowing to better specify and enforce domain properties;
2. it computes explanations using purely abductive procedures, in a true (extended) ALP framework;
3. it attaches probabilities also to abducibles and integrity constraints (as well as rules and facts);
4. it computes the most likely abductive explanation based on such probabilities;
5. it is also compatible with established frameworks (standard ALP, standard ProbLog) if removing the probabilities and/or the generalized constraints, respectively.

Very preliminary ideas for this paper appeared in Rotella and Ferilli (2013). However, compared to that paper, this work is completely new. Its novel contributions include a further extension of ALP with 4 additional kinds of generalized integrity constraints, the formal definition of a corrected and refined purely logical proof procedure for the extended framework, some theoretical results about the new framework, and a full account of the extension to the probabilistic setting.

3 Expressive abductive logic programming

As a first extension of the traditional ALP framework, we propose the introduction of several kinds of integrity constraints. This would allow one to (comfortably and easily) express knowledge about the domain that the traditional framework cannot handle, but that can be determinant for finding an appropriate abductive explanation. We will call the new setting *Expressive Abductive Logic Programming* (EALP).

3.1 Generalized integrity constraints

The integrity constraints used in traditional ALP are in the form of denials, due to their straightforward mapping onto goals in Logic Programming. As such, they express the negation of a conjunction of literals, or, more technically, the *nand* logical operation applied to the set of literals in the body. Inspired by this consideration, by extension we propose to consider additional operators, and to define a different kind of integrity constraint after each of them.

The behavior of each kind of constraint directly follows from the definition of the logical operator after which it is named, and specifically:

`nand([`l_1`,...,`l_n`])` at least one among literals l_1,...,l_n must be false (the classical denials considered in ALP);
`xor([`l_1`,...,`l_n`])` exactly one among literals l_1,...,l_n must be true;
`or([`l_1`,...,`l_n`])` at least one among literals l_1,...,l_n must be true;
`if([`l'_1`,...,`l'_n`],[`l''_1`,...,`l''_m`])` if all literals l'_1,...,l'_n are true, then all literals l''_1,...,l''_m must also be true (*modus ponens*); alternatively, if any literal among l''_1,...,l''_m is false, then at least one literal among l'_1,...,l'_n must also be false (*modus tollens*);
`iff([`l'_1`,...,`l'_n`],[`l''_1`,...,`l''_m`])` either all literals l'_1,...,l'_n and l''_1,...,l''_m are true, or at least one literal among l'_1,...,l'_n and at least one literal among l''_1,...,l''_m are false;
`and([`l_1`,...,`l_n`])` all literals l_1,...,l_n must be true;
`nor([`l_1`,...,`l_n`])` all literals l_1,...,l_n must be false.

The intuition behind these kinds of constraints is clear for `nand`, `xor` and `or` operators. Somehow less intuitive may be the others. The following simple examples should help to clarify their meaning and usefulness.

Example 4 Some generalized integrity constraints in the family relationships domain might be:

nand([father(X),mother(X)]) X cannot be both a father and a mother (but it can be neither);
xor([male(X),female(X)]) X must be either male or female (and it cannot be neither);
or([unemployed(X),employee(X),consultant(X),retired(X)]) X must be an unemployed person, an employee, a consultant or a retired person, and can be more than one of these (e.g., be retired and work as a consultant);
and([parent(X, Y),child(Y, X)]) whenever X is a parent of Y, it must also hold that Y is a child of X, and *vice versa*.
nor([¬parent(X, Y),¬child(Y, X)]) if X is a parent of Y, then Y must be a child of X; if Y is a child of X, then X must be a parent of Y.
if([mother(X)],[female(X)]) if X is a mother, then it must be also female; if X is not female, then it cannot be a mother;
iff([parent(X, Y),male(Y)],[son(Y, X)]) if X is a parent of Y and Y is male, then Y must be a son of X; if Y is a son of X, then X must be a parent of Y and Y must be male; if X is not a parent of Y or Y is not male, then Y cannot be a son of X; if Y is not a son of X, then X must not be a parent of Y or Y must not be male.

Note that, while in simple cases some constraints may be expressed using different kinds, this is not true in general. E.g.: *if([mother(X)],[female(X)])* can be expressed as *or([¬mother(X),female(X)])*, but

if(father(X,Y),married(X,Z)],[male(X),female(Z)])

which in usual logic notation is

$father(X,Y) \wedge married(X,Z) \Rightarrow male(X) \wedge female(Z)$,

would yield, by standard logistic manipulation,

$\neg father(X,Y) \vee \neg married(X,Z) \vee (male(X) \wedge female(Z))$

which cannot be expressed by flat constraints as above.

When working in a FOL setting, constraints may involve variables, and their quantification becomes relevant. The quantification approach we propose is based on the following constraint checking procedure.

A constraint is activated when a literal, whose abduction is under consideration, matches one of its components. In such a case, an instance of the constraint is created with the remaining components and checked for consistency, in which the variables of the matching component have been bound to the constants of the literal being abduced. Two cases may take place:

1. all variables in the constraint are bound, so there is no variable left. The constraint must be proven for the specified values. E.g., by activating constraint *nand([father(X),mother(X)])* with *father(joe)*, the consistency procedure must prove the constraint instance *nand([mother(joe)])*, which involves no variables.
2. only some variables in the constraint are bound, in which case the procedure generates a constraint instance involving unbound variables. E.g., by activating constraint *and([p(X,Y),p(Y,Z),q(Z)])* with *p(a,b)*, the consistency procedure must prove the constraint instance *and([p(b,Z),q(Z)])*.

In case 2, the unbound variables in the activated constraint are quantified according to Table 1: basically, constraints involving negation (`nand` and `nor`) must be checked for all possible instantiations, while constraints of kind `and`, `or` and `xor` must be checked for at least one instantiation; constraints of kind `if` and `iff` reduce to other kinds of constraints, and thus follow the corresponding quantification rules. We show the rationale for these choices by means of some examples.

Example 5 Let us consider some cases of constraints and show how they are processed and interpreted.

nand([player(P,T),coach(P,U),president(P,V)]) "a person P cannot be at the same time the player of a team T, the coach of a team U and the president of a team V". Suppose it

Table 1 Quantification of activated constraints

`nand`	`nor`	`and`	`or`	`xor`	`if`	`iff`
∀	∀	∃	∃	∃	see `nand`, `and`	see `if`

is activated by *player(steve,lakers)*; then, *steve* cannot be also the coach and the president of other teams, i.e.:

$$\nexists U, V : coach(steve, U) \wedge president(steve, V) \equiv$$

$$\equiv \forall U, V : \neg(coach(steve, U) \wedge president(steve, V))$$

or, using the IC formalism:

$\forall U, V$: *nand([coach(steve,U),president(steve,V)])*

(as for denials in the standard ALP framework).

xor([rent(P,O,Q,X),buy(P,O,Q,Y),know(P,O,Q,R)]) "either a person P rents an object O from another person Q at price X, or P buys it from Q at price Y, or both P and the owner Q of O know another person R". Suppose it is activated by *rent(steve,car,john,100)*; then, this must be the only true literal in the constraint, and the remaining instantiated constraint instance must be false in all possible cases, i.e.:

$$\nexists Y, R : buy(steve, car, john, Y) \vee know(steve, car, john, R) \equiv$$

$$\equiv \forall Y, R : \neg(buy(steve, car, john, Y) \vee know(steve, car, john, R))$$

or, using the IC formalism:

$\forall Y, R$: *nor([buy(steve,car,john,Y),know(steve,car,john,R)])*

and([red(P,R),green(P,G),blue(P,B)]) "a pixel P must have a value for all basic color components (R for red, G for green, and B for blue)". Suppose it is activated by *red(p,127)*; then, pixel p must have values for color components *green* and *blue*, as well, i.e.:

$$\exists G, B : green(p, G) \wedge blue(p, B)$$

or, using the IC formalism:

$$\exists G, B : and([green(p,G),blue(p,B)])$$

Indeed, were the remaining constraint universally quantified, i.e.:

$$\forall G, B : green(p, G) \wedge blue(p, B)$$

pixel p should have all possible values for both color component *green* and color component *blue*, which is clearly impossible.

or([black_white(I),gray_levels(I,G),colors(I,C)]) "an image I must be in black and white, or use a number G of gray levels, or use a number C of colors (but it may have parts that are just black and white, parts which use gray levels only, and parts which use colors)". Suppose it is activated by $\neg$*black_white(i)*; then, there must be parts of the image that use gray levels and/or colors, i.e.:

$$\exists G, C : gray\_levels(i, G) \vee colors(i, C)$$

or, using the IC formalism:

$$\exists G, C : or([gray\_levels(i,G),colors(i,C)])$$

Indeed, were the remaining constraint universally quantified, i.e.:

$$\forall G, C : gray\_levels(i, G) \vee colors(i, C)$$

image i should use many different numbers of gray values and/or of colors, which is clearly impossible.

xor([rent(P,O,Q),buy(P,O,Q),know(P,O,Q,R)]) "either a person P rents an object O from another person Q, or P buys it from Q, or both P and the owner Q of O know another person R". Suppose it is activated by $\neg$*rent(steve,car,john)*; then, either *steve* bought the *car* from *john* or both *steve* and *john* know another person. Now, if this claim were universally quantified, i.e.:

$$\forall R : buy(steve, car, john) \oplus know(steve, car, john, R)$$

if both *steve* and *john* know *paul*, but they do not both know *mark*, due to the former *buy(steve,car,john)* should be false, but due to the latter it should be true, which would be an inconsistency. So, there must necessarily be an existential quantification:

$$\exists R : buy(steve, car, john) \oplus know(steve, car, john, R)$$

or, using the IC formalism:
$\exists R$: *xor([buy(steve,car,john),know(steve,car,john,R)])*

3.2 Extended abductive procedure

While the definition of *abductive explanations* is the same as in the standard setting, exploiting the generalized constraints requires some changes in the abductive procedure. Indeed, while in the original procedure denials must be proved false, now constraints must be proved true according to the meaning of their kind. E.g., for *nand* at least one literal must be false, for *xor* exactly one literal must be true, etc. Specifically, the abductive derivation is the same as in the standard framework, while the consistency derivation is changed as follows.

Definition 5 ((Extended) Consistency derivation) An (extended) *consistency derivation* for an abducible α from (α, Δ_0) to $(F_n\ \Delta_n)$ in $\langle P, A, I\rangle$ is a sequence

$$(\alpha\ \Delta_0), (F_1\ \Delta_1), ..., (F_n\ \Delta_n)$$

where F_1 and Δ_1 are $\mathcal{C}$ and Δ as initialized by Algorithm 1 just before the *while* loop, and for all $i > 1$, F_i and Δ_i are the $\mathcal{C}$ and Δ_O obtained by Algorithm 2 at the end of the i-th run of the *while* loop in Algorithm 1.

Algorithm 1 Extended Consistency Derivation

Require α: literal to be abduced ; Δ_I: current abductive assumptions ;
Require $T = \langle P, A, I\rangle$: expressive abductive logic program
  $\mathcal{C} \leftarrow$ priority queue by kind of all instances of integrity constraints in I involving α
  $\Delta \leftarrow \Delta_I \cup \{\alpha\}$
  while $\mathcal{C} \neq \emptyset$ **do**
    $C \leftarrow$ get($\mathcal{C}$)
    $\mathcal{C} \leftarrow \mathcal{C} \setminus \{C\}$
    if handle_constraint($C, \Delta, T, \mathcal{C}$) fails **then**
      FAIL
    else
      $\Delta \leftarrow$ handle_constraint($C, \Delta, T, \mathcal{C}$)
  return Δ

Algorithm 2 handle_constraint(C, Δ_I, T, $\mathcal{C}$): Extended constraints handling

Require C: constraint to be handled ; Δ_I: current abductive assumptions ;
Require $T = \langle P, A, I \rangle$: expressive abductive logic program
Require $\mathcal{C}$: priority queue by kind of instances of integrity constraints
if $C = and([l_1, \ldots, l_n])$ **then**
  if (*) $\exists(\theta, \Delta_O) \in \text{abd}_T(\leftarrow l_1, \Delta_I)$ **then**
    if $n > 1$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{and([l_2, \ldots, l_n])\theta\}$
  else
    FAIL (**)
else if $C = nor([l_1, \ldots, l_n])$ **then**
  if $\exists(\theta, \Delta) \in \text{abd}_T(\leftarrow \overline{l_1}, \Delta_I) \wedge \Delta_O \in \text{mce}(P, \overline{l_1}, \Delta_I)$ **then**
    if $n > 1$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{nor([l_2, \ldots, l_n])\theta \mid (\theta, \Delta) \in \text{abd}_T(\leftarrow \overline{l_1}, \Delta_I)\}$
  else
    FAIL (**)
else if $C = xor([l_1, \ldots, l_n])$ **then**
  if (*) $\exists(\theta, \Delta_O) \in \text{abd}_T(\leftarrow l_1, \Delta_I)$ **then**
    if $n > 1$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{nor([l_2, \ldots, l_n])\theta\}$
  else if $\exists(\theta, \Delta_O) \in \text{abd}_T(\leftarrow \overline{l_1}, \Delta_I)$ **then**
    if $n > 1$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{xor([l_2, \ldots, l_n])\theta\}$ **else** FAIL (**)
else if $C = iff([l'_1, \ldots, l'_n], [l''_1, \ldots, l''_m])$ **then**
  if (*) $\exists(\theta, \Delta_O) \in \text{abd}_T(\leftarrow l'_1, \Delta_I)$ **then**
    if $n > 1$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{iff([l'_2, \ldots, l'_n], [l''_1, \ldots, l''_m])\theta\}$
    else $\mathcal{C} \leftarrow \mathcal{C} \cup \{and([l''_1, \ldots, l''_m])\theta\}$
  else if $\exists(\theta, \Delta) \in \text{abd}_T(\leftarrow \overline{l'_1}, \Delta_I) \wedge \Delta_O \in \text{mce}(P, \overline{l'_1}, \Delta_I)$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{nand([l''_1, \ldots, l''_m])\theta \mid (\theta, \Delta) \in \text{abd}_T(\leftarrow \overline{l'_1}, \Delta_I)\}$
else if $C = if([l'_1, \ldots, l'_n], [l''_1, \ldots, l''_m])$ **then**
  if $\exists(\theta, \Delta) \in \text{abd}_T(\leftarrow \overline{l'_1}, \Delta_I) \wedge \Delta_O \in \text{mce}(P, \overline{l'_1}, \Delta_I)$ **then**
    NOP (*ex falsum quodlibet*)
  else if (*) $\exists(\theta, \Delta_O) \in \text{abd}_T(\leftarrow l_1, \Delta_I)$ **then**
    if $n > 1$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{if([l'_2, \ldots, l'_n], [l''_1, \ldots, l''_m])\theta\}$
    else $\mathcal{C} \leftarrow \mathcal{C} \cup \{and([l''_1, \ldots, l''_m])\theta\}$
else if $C = or([l_1, \ldots, l_n])$ **then**
  if $\exists i \in \{1, \ldots, n\}$ s.t. $\exists(\theta, \Delta_O) \in \text{abd}_T(\leftarrow l_i, \Delta_I)$ **then**
    NOP
  else
    FAIL (**)
else if $C = nand([l_1, \ldots, l_n])$ **then**
  if $\exists i \in \{1, \ldots, n\}$ s.t. $\overline{l_i} \in P \cup \Delta_I$ **then**
    $\Delta_O \leftarrow \Delta_I$
  else if $\exists(\theta, \Delta) \in \text{abd}_T(\leftarrow \overline{l_1}, \Delta_I) \wedge \Delta_O \in \text{mce}(P, \overline{l_1}, \Delta_I)$ **then**
    NOP
  else if $\exists(\theta, \Delta) \in \text{abd}_T(\leftarrow l_1, \Delta_I) \wedge \Delta_O \in \text{mce}(P, l_1, \Delta_I)$ **then**
    if $n > 1$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{nand([l_2, \ldots, l_n])\theta \mid (\theta, \Delta) \in \text{abd}_T(\leftarrow l_1, \Delta_I)\}$
    else FAIL (**)
return Δ_O
Note: The role in backtracking of steps marked (*) or (**) is explained in the running text.

Algorithm 1 takes an abducible literal α and a set of previous abductive assumptions Δ_I, and returns a minimal extended set of abductive assumptions needed to prove the abducible literal in an expressive abductive theory T while satisfying the constraints, or fails if at least one constraint cannot be satisfied. It works by collecting the set $\mathcal{C}$ of all constraint instances involving α, and then suitably selecting and processing them in turn.

Albeit the given constraints can be checked in any order by the abductive proof procedure, efficiency considerations suggest some priorities in their evaluation. *and* and *nor* constraints are to be checked first, since they force the verification (and possible abduction) of some literals, and thus evaluating them first allows to determine the necessary abductions that will bias all subsequent reasoning. Then, *xor*, *if* and *iff* constraints must be checked, since they have a limited set of options about what can be true and what can be false. Finally, *nand* and *or* constraints must be checked last, in order to ensure minimality of the abductive explanation. Indeed, differently from the previous kinds of constraints/operators, they are satisfied by any subset of their arguments being false or true, respectively. Checking them last ensures that all facts that must be true or false according to the other constraints (and the background knowledge) have already been determined, and can be brought to bear, preventing the abduction of additional (unnecessary) literals. So, constraints are organized in a priority queue. Until the queue becomes empty (i.e., all constraints have been satisfied), the constraint instance with highest priority is selected, removed and processed.

Constraint instances are processed by procedure handle_constraint$(\cdot, \cdot, \cdot, \cdot)$, based on Algorithm 2. It selects the first literal in the constraint (smarter selection strategies might be defined, but this is outside the scope of this paper) and checks it using the abductive derivation procedure[4] $\text{abd}_T(\cdot, \cdot)$, returning the set Δ_O of abductive assumptions and suitably updating the constraints queue $\mathcal{C}$. The specific behavior depends on the kind of constraint, but as a general rule:

- if the literal proof agrees with the constraint, then it is removed from the constraint and the rest of the constraint is enqueued for further processing, unless the constraint is satisfied, in which case no action is required.
- if the literal proof disagrees with the constraint, if the constraint is violated, then the procedure fails, otherwise the algorithm abductively proves the negation of the literal, removes the literal from the constraint and enqueues the rest of the constraint for further processing.

More specifically:

and constraints are satisfied when the list of literals becomes empty after all of its members have been proven to be true; they fail as soon as one literal cannot be proved to be true.

nor constraints are satisfied when the list of literals becomes empty after all of its members have been proven to be false; they fail as soon as one literal cannot be proved to be false.

[4]Note that the calls to the abductive derivation are meant to be executed non-deterministically; when successful, they return a suitable partial Δ_O which is combined to other partial Δ_O's found in other derivation steps to give the overall Δ returned by a successful run of the complete algorithm.

xor constraints are satisfied when, after proving one literal to be true, the list of all remaining literals is proved to be false (i.e., a *nor* constraint holds on it); they fail when the list of literals becomes empty after all of its members have been proven to be false.

iff constraints are checked as follows: the *and* of the premise is checked; if it succeeds the *and* of the consequence is enqueued for checking; otherwise, as soon as the premise fails, the *nand* of the consequence is enqueued for checking.

if constraints are checked based on the $X \Rightarrow Y \equiv \neg X \vee Y$ equivalence: the *nand* of the premise is checked; if it succeeds nothing is done (the constraint is satisfied); otherwise, after the *nand* fails (i.e., the *and* of the premise is satisfied), the *and* of the consequence is enqueued for checking.

or constraints are satisfied as soon as one literal is proven to be true; they fail when the list of literals becomes empty after proving all of its members to be false.

nand constraints are satisfied as soon as one literal is proven to be false; they fail when the list of literals becomes empty after proving all of its members to be true.

Note that the case in which an abduced or known literal satisfies an *or* or *nand* constraint, but it is not the first in the constraint, must be purposely checked before starting the constraint evaluation, to prevent the abduction of unnecessary literals needed to prove the literals before it in the constraint.

When processing a literal l of a universally quantified constraint, all possible instances of l must be considered altogether, each of which will generate a new constraint instance to be enqueued. For this purpose, given an expressive abductive theory $T = \langle P, A, I \rangle$ and a set of existing abductive assumptions Δ_I, let us define the set of *cumulative explanations* of l as follows:

$$\mathrm{ce}(P, l, \Delta_I) = \{\Delta_O \mid \forall(\theta, \Delta) \in \mathrm{abd}_T(\leftarrow l, \Delta_I) : P \cup \Delta_I \cup \Delta_O \models l\theta\}$$

i.e., each member of $\mathrm{ce}(P, l, \Delta_I)$ can explain all instances of l in T. Note that members of $\mathrm{ce}(P, l, \Delta_I)$ do not necessarily include all Δ's that explain a given instance $l\theta$. Indeed, if many abductive explanations are available for an instance, including all of them would cause redundancy. However, since we are looking for minimal explanations, we must use only minimal members of this set. So, Algorithm 2 considers the set of *minimal cumulative explanations*, defined as follows:

$$\mathrm{mce}(P, l, \Delta_I) = \{\Delta_O \in \mathrm{ce}(P, l, \Delta_I) \mid \nexists \Delta'_O \in \mathrm{ce}(P, l, \Delta_I) \ni' \Delta'_O \subset \Delta_O\}$$

Operationally, a minimal cumulative explanation can be obtained by sequentially proving the literal instances, feeding each proof with the partial cumulative explanation of the previous ones.

3.3 Properties of EALP

Let us first show that the proposed set of generalized integrity constraints strictly extends the expressiveness of the framework, because their meaning cannot be captured by the traditional integrity constraints used in ALP.

Property 1 The proposed EALP framework is strictly more expressive than standard ALP.

Proof To prove this claim, it is sufficient to show that there exist at least one kind of constraint in EALP that cannot be expressed as a (set of) denial(s). Consider the XOR constraint: $xor([p, q]) \equiv p \oplus q$. By applying standard logistic manipulation we have:

$$p \oplus q \equiv (p \wedge \neg q) \vee (\neg p \wedge q) \equiv \neg(\neg(p \wedge \neg q) \wedge \neg(\neg p \wedge q))$$

which is clearly not a plain denial. Indeed, albeit the *nand* operator alone is functionally complete, nested applications thereof are needed to express the other operator. But *nand* is not associative, and thus the nestings cannot be flattened to plain denials. □

Actually, some constraints in the extended set can be simulated by others, but were introduced only for ease of specification and for efficiency of computation:

- $and([a_1, \ldots, a_n]) \equiv nor([\overline{a_1}, \ldots, \overline{a_n}])$ by De Morgan's laws
- $or([a_1, \ldots, a_n]) \equiv nand([\overline{a_1}, \ldots, \overline{a_n}])$ by De Morgan's laws
- $xor([a_1, \ldots, a_n]) \equiv$
  $\equiv \{and([a_1, \overline{a_2}, \ldots, \overline{a_n}]), and(\overline{a_1}, a_2, \overline{a_3} \ldots, \overline{a_n}), and(\overline{a_1}, \ldots, \overline{a_{n-1}}, a_n)\}$
- $iff([l'_1, \ldots, l'_n], [l''_1, \ldots, l''_m]) \equiv$
  $\equiv \{if([l'_1, \ldots, l'_n], [l''_1, \ldots, l''_m]), if([l''_1, \ldots, l''_m], [l'_1, \ldots, l'_n])\}$

Let us then provide a (semi-formal) proof of the minimality of abductive explanations computed by the proposed procedure.

Theorem 1 (Minimality) *Any abductive explanation Δ associated to an abductive proof obtained using the consistency derivation computed by Algorithm 1 in the abductive derivation computed as per Definition 3 is minimal, i.e., it does not contain unnecessary abduced literals.*

Proof First consider the set of integrity constraints $\mathcal{C}$ used to obtain Δ. Notice that integrity constraints are considered by the proof procedure, and added to $\mathcal{C}$, if and only if they involve a literal that must be abduced. So, no integrity constraint in $\mathcal{C}$ is superfluous, and the only way in which Δ might not be minimal is by abducing unneeded literals from the necessary integrity constraints. Now, only the consistency derivation may add literals to Δ while checking integrity constraints. The evaluation of each single literal is performed by calling the abductive derivation, which succeeds without abducing new literals whenever the literal is already known or abduced. So, a literal is abduced only when unknown and not yet abduced. In particular, only integrity constraints that succeed add literals to Δ. Let us analyze all possible cases:

1. for *and* constraints, all literals must be true, and thus none of them is unnecessary: all literals already known or abduced to be true are skipped, and all remaining literals must necessarily be abduced to be true;
2. for *nor* constraints, all literals must be false, and thus none of them is unnecessary: all literals already known or abduced to be false are skipped, and all remaining literals must necessarily be abduced to be false;
3. for *xor* constraints, all literals are necessary (exactly one must be true, and all the others must be false): as soon as the procedure can prove one literal to be true, the constraint

is dropped and a *nor* constraint is added for the remaining literals, whose minimality is ensured by case (2);[5]

4. for *or* constraints, if at least one of the literals is already known or abduced to be true, no further literal is abduced; otherwise, the procedure stops (i.e., it drops the constraint) as soon as it can prove one literal to be true;
5. for *nand* constraints, if at least one of the literals is already known or abduced to be false, no further literal is abduced; otherwise, the procedure stops (i.e., it drops the constraint) as soon as it can prove one literal to be false;
6. for *if* constraints, evaluation boils down to evaluating either a *nand* constraint (for the premises), in which case minimality is ensured by case (5) and the conclusions are skipped (so no further unnecessary literal is abduced), or an *and* constraint (for the conclusions), in which case minimality is ensured by case (1);
7. for *iff* constraints, evaluation boils down to evaluating two *and* constraints, in which case minimality is ensured by case (1); or two *nand* constraints, in which case minimality is ensured by case (5). □

Note that *minimal* solutions are not necessarily *minimum*, i.e., it is possible that other choices (of Δ_O or of constraint orders) in the non-deterministic steps of the algorithm result in a smaller but uncomparable set of abduced literals.

A further relevant consideration concerns the tractability of EALP, which is the same as for standard ALP:

Property 2 The EALP proof procedure has the same computational complexity as standard ALP, which is linear in the number of involved constraint instances and of literals in them.

Proof Let us first note that the EALP proof procedure differs from the standard ALP proof procedure only in the consistency derivation step. Just like for standard ALP, this step must consider and satisfy all constraint instances that come into play when trying to abduce a literal, and just once each of them. So, the difference is actually in the way the extended procedure proves the additional kinds of constraints other than *nand*. However, constraints of kind *and*, *nor*, *xor*, and *or* just require scanning and satisfying a list of literals, just as for *nand*. In particular, *and*, *xor* and *nor* require to scan and satisfy the whole list, while for *or* the procedure may stop as soon as a true literal is found (just like for *nand* it may stop as soon as a false literal is found). Constraints of kind *if* and *iff* have two ways each of being satisfied. Each such way requires to scan just once the left-hand-side list of literals and the right-hand-side one, so, again, the required effort is linear in the sum of the lengths of these two lists (i.e., as stated by the thesis, in the number of literals that make up these constraints). □

Finally, albeit it is not a formal property, it may nevertheless be interesting to point out that the realistic expectation of having good abductive inference in EALP is not less than in

[5]Actually, this behavior is inefficient: if any literal in the *nor* constraint (say, l_i) was already known or abduced to be true before starting the consistency check on the constraint, all proofs based on the previous abduction of other literals l_j with $j < i$ would fail when encountering l_i, causing backtracking until the turn of l_i itself comes in the consistency check. This might be optimized by first checking if any literal in the *xor* constraint is already known or abduced to be true; if so and it is exactly one, then the constraint is automatically satisfied without any further abduction; if there are several such literals, then the constraint immediately fails; in the other cases the described procedure must be started. Anyway, as said, optimization of the proposed approach is not the subject of this paper.

standard ALP. Instead, it is possibly more than in ALP, because, having proved that EALP is strictly more expressive, the additional kinds of constraints allow one to specify and enforce domain properties that standard ALP cannot express and, thus, enforce.

4 Probabilistic expressive abductive logic programming

After extending the expressiveness of the purely logical abductive framework, we can add flexibility to it by introducing probabilities in order to smooth the classical sharp approach with a statistical one.

4.1 Probabilistic framework

While deductive reasoning is truth preserving (given true premises, the consequences of a sound inference rule must necessarily be true), abductive reasoning is valid only to the extent that the abductive explanation is valid. Thus, the idea of associating a degree of 'validity' (or 'reliability', or 'likelihood') to the explanation immediately comes to mind, in order to have an indication of how trustworthy it is. This would also open new, interesting perspectives. Indeed, one may consider an abductive explanation as shaping a *possible world*, in which the assumptions it makes (abduced literals) are true. Having different sets of consistent assumptions that explain a goal is like considering different possible worlds, in each of which a different set of abduced facts holds. In principle, each of these worlds might be the correct one, as far as we know. But while, from a strictly logical viewpoint, any consistent explanation is as good as any other, once explanations are associated to a degree of validity, some of them might turn out to be more or less likely than others. In such a case, of course, we might prefer to choose the more likely ones.

Now, since the validity of abductive explanations is supported only by the integrity constraints they satisfy, it follows that the degree of their validity would be directly affected by different degrees of 'validity' (or 'reliability', or 'likelihood') of those integrity constraints. This perspective has been so far almost neglected by the literature, that focused on the logic program part of the abductive framework. Instead, we posit that, in a truly and fully probabilistic abductive setting, not only the clauses in P, but also the integrity constraints in I must be probabilistic. In a sense, this is also reflected in the real world, where constraints are not always sharp. Even when they express universally valid laws (such as temporal or physical ones) they might have to be flexible to handle noise and uncertainty. Quite often, they just represent domain-specific restrictions, that hold only to some degree of certainty, reflecting a degree of personal belief or confidence in their being true, or a statistical frequency.

So, in our extended abductive framework we associate each integrity constraint to a probabilistic value, that we call its *strength*, expressing its degree of validity in the given domain.[6]

[6]We prefer calling it a 'strength', using a more neutral term than 'likelihood', because often these values express an intuitive degree of validity of the constraint, rather than a theoretically founded computation of its likelihood of being true. Indeed, as quite usual in the probabilistic logic programming setting, these values are manually set by the programmers of the logic theory. Of course, it might be possible to devise procedures that try to assess these values from the available data, but it is a line of research on its own and is outside of the scope of this paper, which focuses instead on the exploitation of the given values.

Definition 6 A *Probabilistic Expressive Abductive Logic Program* (PEALP for short) consists of a 4-tuple $\langle P, A, I, p \rangle$, where:

- P is a (standard or probabilistic) logic program;
- A (Abducible predicates) is a set of predicates;
- I (Integrity Constraints) is a set of generalized integrity constraints, as defined in Section 3.1, that must be satisfied by abductive hypotheses;
- $p : P \cup \text{ground}(A) \cup I \rightarrow [0, 1]$ is a function associating a likelihood to each element in P, in I (called the *strength* of a constraint) and in the set ground(A) of ground literals built on predicates in A.[7]

For the sake of compactness, in the following all items for which $p(\cdot)$ is not specified will be assumed to have likelihood equal to 1.0.

As usual in the literature, we base our probabilistic approach on the notion of *possible worlds*, re-defined as follows in the abductive setting.

Definition 7 Given a goal G and a PEALP $T = \langle P, A, I, p \rangle$, a *(probabilistic) abductive explanation* of (or *possible world* for) G in T is a triple $E = (L, \Delta, \mathcal{C})$, where L, Δ and $\mathcal{C}$ are, respectively, the set of clauses in P, the abductive explanation (i.e., the set of ground literals abduced), and the set of instances of (probabilistic) integrity constraints in I, involved in an abductive proof of G in T. Specifically, we denote with $C \in \mathcal{C}$ the instances of constraints that are satisfied by Δ, and by $\overline{C} \in \mathcal{C}$ the instances of constraints that are violated by Δ. We denote by $\mathcal{W}_G$ the set of all possible worlds associated to G in T.

Differently from traditional explanations, probabilistic explanations must report the involved integrity constraints, because the probability of these constraints affects the probability of the explanation, just like that of the program elements (L) or abduced literals (Δ). Moreover, when integrity constraints are probabilistic, a world that violates them is not anymore impossible, it is just differently probable. For this reason, also explanations that violate constraints are valid, and the information about violated constraints must be reported in the explanation, just like information about satisfied constraints. This information can be collected and returned by Algorithm 1 with trivial modifications. Also note that, in a possible world $(L, \Delta, \mathcal{C})$, Δ and/or $\mathcal{C}$ may be empty. When $\Delta = \emptyset$ (in which case also $\mathcal{C}$ must be empty), the explanation is deductive, and as such it is intrinsically minimal (and minimum).

4.2 Computational procedures

So, while the basic procedure returns one minimal abductive explanation of the goal (any minimal explanation is as reliable as any other), in a probabilistic setting we need to generate all different (minimal) abductive explanations, and extend the logical proof procedure given in Algorithm 1 to consider all possible worlds. This means that, whenever the abductive procedure has a choice, it must explore the worlds associated to all different options. One such choice is in the abductive derivation, when different clauses can be applied to resolve the selected literal (using different rules leads to different explanations, involving different

[7] The assessment of the likelihood $p(\delta)$ that abducible δ is true is outside the scope of this paper. E.g., as a naive approach, one might assume that this is the a priori probability that the predicate on which δ is built is true.

abducibles and different constraints to be satisfied). Another choice is in the consistency derivation, when it might abduce either a literal or its negation (all constraints, except *and* and *nor*, admit different ways for being satisfied). The former branches the current world in as many possible worlds as many clauses are applicable; the latter branches the current world in two possible worlds (one where the literal is true and another where it is false). Each of these choices leads to a different world, and acts as a bias for the abductive assumptions that may be made next. Note that, since *nand* constraints (i.e., the classical denials) can be satisfied in several ways, these considerations are applicable also to the standard ALP setting.

The exploration of all possible worlds is associated to a tree in which nodes represent sets of possible worlds: the root is the set of all possible worlds; each branching corresponds to a choice (as described above), whose offspring represents worlds that are specializations of the parent node; internal nodes are sets of possible worlds associated to partial abductive proofs; and leaves represent the worlds associated to abductive explanations. Operationally, such a tree can be generated using a depth-first approach, collecting the leaves when they are reached and forcing backtracking in all possible choice points until all possible worlds associated to (minimal) abductive explanations have been explored. This is the same as the backtracking provided for by the classical procedure to return all minimal abductive explanations. The choice points are in case (1) of the abductive derivation (the selection of the clause in P for resolution), and in statements marked by (*) in Algorithm 2 for the consistency derivation. It is important to note that failing branches, marked by (**) in Algorithm 2, do not fail anymore, but collect the violated constraints to build $\mathcal{C}$ and continue. So, case (1) of the abductive derivation collects the clauses in P to build L, while (*) and (**) in the consistency derivation collect the instances of constraints to build $\mathcal{C}$.

Now, the probability of each possible world must be assessed. The solution in the literature that is closest to our needs is cProbLog. However, in cProbLog constraints are sharp, while in our framework they are probabilistic, as well. So, we have to modify the cProbLog approach for computing the likelihood of a world. It is beyond the scope of this paper discussing in details the underlying semantics that we use to cope with this. It suffices to say that we must take into account the likelihood of the (satisfied or violated) constraints that are involved in a possible world, and that, while cProbLog removes from the probability count all worlds that do not satisfy the constraints, we must consider again all worlds (as in ProbLog), because a world that violates a constraint is only less probable, not impossible.

Given a PEALP $T = \langle P, A, I, p \rangle$, where P is a probabilistic logic (ProbLog) program, and a (probabilistic) abductive explanation $E = (L, \Delta, \mathcal{C})$ associated to a goal G, the (absolute) likelihood of E is:

$$\overline{p}(E) = p(L|P) \cdot \prod_{\delta \in \Delta} p(\delta) \cdot \prod_{C \in \mathcal{C}} p(C) \tag{1}$$

$$= \prod_{l \in L} p(l) \cdot \prod_{l \in P \setminus L} (1 - p(l)) \cdot \prod_{\delta \in \Delta} p(\delta) \cdot \prod_{C \in \mathcal{C}} p(C) \tag{2}$$

where $p(L|P)$ is as in ProbLog, and the rest of the formula throws in the computation the likelihood assessment for the abduced literals (with the probability of a negated literal $\overline{\delta} \in \Delta$ being equal to $1 - p(\delta)$) and constraints (with the probability of a violated constraint $\overline{C} \in \mathcal{C}$ being equal to $1 - p(C)$) involved, respectively.

Actually, for our purposes of comparing alternate abductive explanations, in order to choose the most likely one(s), we may use just a pseudo-likelihood, based on a simpler version of the formula:[8]

$$p(E) = \prod_{l \in L} p(l) \cdot \prod_{\delta \in \Delta} p(\delta) \cdot \prod_{C \in \mathcal{C}} p(C) \tag{3}$$

where the initial part expresses the likelihood that all program elements or abduced literals in an explanation are true in that particular world, and the final part expresses the overall reliability of the constraints involved in the abductive explanation. Intuitively, (3) is based on the fact that any element (rule, fact, abduced literal, or constraint) in an explanation must be either true or false, which corresponds to a choice between two alternative sets of possible worlds, while the value of elements that are not involved does not care. If one choice has probability p, the opposite choice would lead to a disjoint set of worlds with probability $1 - p$. Note that if any component in T is not probabilistic, the corresponding product equals 1 and so only probabilistic components concur in differentiating the likelihood of the explanations, as expected. In particular, the standard (non-probabilistic) setting, in which all components of T are not probabilistic, is accounted for. Moreover, for sharp constraints, violated constraints have probability 0, and the whole product becomes 0, meaning that the world is impossible and must be discarded.

After determining all possible worlds/explanations of a goal G, and their associated probabilities, the issue of selecting the best one arises. Indeed, while in standard PLP the set of true facts is known and fixed, and different proofs just use a different subset thereof, in an abductive setting each explanation may introduce new facts, and facts in different explanations might be inconsistent with each other. We propose to select the *most likely explanation*, i.e., the one with maximum (pseudo-)likelihood among all $\mathcal{W}_G$ possible worlds:

$$\overline{\Delta} \text{ s.t. } (\overline{L}, \overline{\Delta}, \overline{C}) = \overline{E} = \arg \max_{E \in \mathcal{W}_G} p(E) \tag{4}$$

As said, for this purpose, there is no need to normalize the probabilities of the explanations over all possible worlds.

4.3 An example

Consider the following logic program P, encoding in predicate *printable/1* a policy to decide whether a document can be sent to print:

$$\begin{aligned} P = \{\ & R_1 : printable(X) \leftarrow a4(X), \neg image(X), black\_white(X) \\ & R_2 : printable(X) \leftarrow a4(X), table(X) \\ & R_3 : printable(X) \leftarrow a4(X), image(X), color(X) \\ & F_1 : a4(d) \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \} \end{aligned}$$

and the goal $G = \leftarrow printable(d)$.

[8] Using (3), the actual probability of an abductive explanation $\overline{E} \in \mathcal{W}_G$ relative to the set $\mathcal{W}_G$ of all possible worlds can be computed using the following normalization:

$$p'(\overline{E}) = \frac{p(\overline{E})}{\sum_{E \in \mathcal{W}_G} p(E)}$$

Future work will deal specifically with the definition of a formal probability distribution for the assessment of the likelihood of possible worlds.

In a purely deductive setting, G cannot be proven in P, since the available definitions for concept $printable(X)$ need facts that are not in P to succeed (only $a4(d)$ holds in P). An abductive setting is needed to provide an explanation for $printable(d)$. Let us consider the following:

$$
\begin{aligned}
A &= \{image/1, text/1, black\_white/1, table/1\} \\
I &= \{ \quad C_1 = or([table(X), text(X), image(X)]) \\
&\qquad C_2 = nand([text(X), color(X)]) \\
&\qquad C_3 = and([image(X), color(X)]) \\
&\qquad C_4 = xor([black\_white(X), color(X)]) \qquad \} \\
p(R_1) &= p(R_2) = p(R_3) = p(F_1) = 1.0 \\
\forall X: \quad & p(\text{image}(X)) = 0.4,\ p(\text{text}(X)) = 0.8,\ p(\text{table}(X)) = 0.3, \\
& p(\text{color}(X)) = 0.2,\ p(\text{black\_white}(X)) = 0.7 \\
p(C_1) &= 0.9,\ p(C_2) = 0.6,\ p(C_3) = 0.7,\ p(C_4) = 0.8
\end{aligned}
$$

The reported strengths mean that in 90% of the cases a document must include at least one among tables, images or text (constraint C_1); in 80% of the cases a document must be either in black and white or in color (constraint C_4); etc.

To prove G, an *abductive derivation* is started, described in details in the next paragraphs and schematically summarized in Table 2 as follows. Each row reports a possible explanation, identified by the integer reported in the first column. Column P reports the rule used in the explanation (each explanation used just one rule from P, plus fact F_1). Then, section A reports, with obvious abbreviations, the abduced literals that make up the explanation, where: a T_i means that the literal is positive, a F_i means that the literal is negated, and the subscript expresses the order in which the literals are generated by the associated proof. Then, section I reports the integrity constraints, specifying if they are satisfied (+) or violated (−) by the explanation. A blank cell in A or I means that the literal or constraint was not involved in the explanation. Finally, the last column reports the probability of the world associated to the explanation. For the sake of readability, the probability of each element in

Table 2 Explanations (possible worlds) for query $G = \leftarrow printable(d)$

	P	A					I				
#W		img	tab	txt	col	b_w	C_1	C_2	C_3	C_4	
	1.0	0.4	0.3	0.8	0.2	0.7	0.9	0.6	0.7	0.8	p
1	R_1	F_1	T_2		F_4	T_3	+	+		+	0.0435456
2	R_1	F_1	T_2	F_5	T_4	T_3	+	+	−	−	0.000653184
3	R_1	F_1	T_2	T_5	T_4	T_3	+	−	−	−	0.000435456
4	R_1	F_1	F_2	T_3	F_4	T_5	+	+		+	0.08128512
5	R_1	F_1	F_2	T_3	T_4	T_5	+	−	−	−	0.01016064
6	R_1	F_1	F_2	F_3	F_5	T_4	−	+		+	0.0150528
7	R_1	F_1	F_2	F_3	T_5	T_4	−	+	−	−	0.000028224
8	R_2		T_1				+				0.27
9	R_3	T_1		F_3	T_2	F_4	+	+	+	+	0.00145152
10	R_3	T_1		F_3	T_2	T_4	+	+	+	−	0.00084672
11	R_3	T_1		T_3	T_2		+	−	+		0.06912

P, A and I is also reported in the table heading; since all elements in P have probability 1.0, they do not affect the final probability of the explanation.

Suppose case (1) selects rule R_1 in P. $a4(d)$ holds in P, so the abductive procedure proceeds with the next literal $\neg image(d)$. Since $image(d)$ is not in P, and $image/1$ is abducible, its negation might be assumed by the abductive procedure, provided that all integrity constraints involving it (only C_1 in this case) are satisfied using the *consistency derivation*. C_1 may be satisfied by abducing $table(d)$ (which is possible because $table/1 \in A$), that does not involve further constraints. To complete the explanation of the goal using rule R_1, $black\_white(d)$ must be abduced (indeed, $black\_white/1 \in A$), involving constraint C_4. It is of kind *xor*, so the only true literal must be $black\_white(d)$ and the other $color(d)$ must be false. $color/1$ is abducible, and abducing $\neg color(d)$ in turn involves constraint C_2, of kind *nand*, which is automatically satisfied because $\neg color(d)$ itself falsifies the *and*. Overall, this explanation (#1) involved (and satisfied) constraints C_1, C_2 and C_4.

To find other possible worlds, backtracking is forced. The last choice point was $\neg color(d)$, so $color(d)$ is abduced, which involves constraints C_2, C_3 and C_4. It violates constraints C_4 (its negation was abduced to satisfy it) and C_3 (because $image(d)$ cannot be abduced to satisfy the *and* constraint, having already abduced its negation). Since $text/1 \in A$, the *nand* constraint C_2 can be satisfied by abucing $\neg text(d)$, which still satisfies constraint C_1. So, this explanation (#2) satisfies constraints C_1 and C_2, and violates constraints C_3 and C_4.

The latest choice point to backtrack on is the abduction of $\neg text(d)$. Abducing $text(d)$ instead violates constraint C_2 (for which $\neg text(d)$ had been abduced), but still satisfies constraint C_1. This explanation (#3) satisfies constraint C_1 but violates constraints C_2, C_3 and C_4.

The next backtracking is on the abduction of $table(d)$. Abducing $\neg table(d)$ instead involves the *or* constraint C_1 that, having already abduced $\neg image(d)$, must be satisfied by abducing $text(d)$. This in turn involves the *nand* constraint C_2, satisfied by abducing $\neg color(d)$, which involves the *xor* constraint C_4, satisfied by abducing $black\_white(d)$. Now the abductive proof for $\neg image(d)$ is complete, and proof of rule R_1 proceeds to $black\_white(d)$, which was just abduced. This completes the explanation (#4), which involved (and satisfied) constraints C_1, C_2 and C_4.

Backtracking on the latest choice point, i.e. the abduction of $black\_white(d)$, the procedure now abduces its negation $\neg black\_white(d)$, violating constraint C_4 but also preventing the proof of R_1. So, this world is discarded.

The procedure backtracks on $\neg color(d)$, now abducing $color(d)$, which violates constraint C_2 and also involves constraints C_3 and C_4. C_3 (an *and* constraint) is also violated, because it would require abduction of $image(d)$ but $\neg image(d)$ was already abduced. C_4, an *xor* constraint, would be satisfied by abducing $\neg black\_white(d)$, but this would prevent the proof of R_1, so this world is discarded, as well.

Abducing (by backtracking) $black\_white(d)$ instead violates constraint C_4, but completes the proof of R_1, returning an explanation (#5) that satisfies constraint C_1 but violates constraints C_2, C_3 and C_4.

Backtracking now goes to $text(d)$, changing the abduction to $\neg text(d)$ and thus violating constraint C_1 (because all components in the *or* are now false), but automatically satisfying constraint C_2 (because at least one component in the *nand* is false). No more constraints are involved, so the proof proceeds with the next literal in R_1, which is $black\_white(d)$. Abducing this literal involves the *xor* constraint C_4, satisfied by abducing

$\neg color(d)$, which would involve the *nand* constraint C_2 which, however, is already satisfied. The resulting explanation (#6) satisfies constraints C_2 and C_4 but violates constraint C_1.

Backtracking on $\neg color(d)$ and abducing $color(d)$ violates constraint C_4, and further involves constraints C_2 and C_3. The *and* constraint C_3 has priority, and would be satisfied by abducing $image(d)$, but this is impossible, since its negation $\neg image(d)$ was already abduced in this world, so C_3 is violated. On the other hand, C_2 was already satisfied by $\neg text(d)$. This explanation (#7) satisfies constraint C_2 and violates constraints C_1, C_3 and C_4.

All possible backtrackings in the proof of rule R_1 have been explored, so the procedure backtracks on the rule itself and selects R_2. Again, $a4(d)$ holds in P, so the abductive procedure proceeds with the next literal $table(d)$. Since $table/1 \in A$, it can be abduced. It involves only the *or* constraint C_1, and is also sufficent to satisfy it, so the proof of R_2 is complete, returning an explanation (#8) that involves and satisfies only constraint C_1.

No backtracking is possible on R_2, since backtracking on $table(d)$ and abducing $\neg table(d)$ the proof would fail, and the corresponding world would be impossible.

The procedure selects now R_3, where $a4(d)$ is true in P. The next literal to be proven is $image(d)$, which can be done because $image/1 \in A$. This abduction involves (and satisfies) constraints C_1 and C_3. C_1 is an *or* constraint, and thus its evaluation is postponed. The *and* constraint C_3 can be satisfied by abducing $color(d)$. This can be done because $color/1 \in A$, and further involves constraints C_2 and C_4. They have the same priority, so they are processed in this order. To satisfy C_2, which is a *nand* constraint, literal $text(d)$ must be falsified, by abducing $\neg text(d)$. To satisfy C_4, which is an *xor* constraint, $black\_white(d)$ must be falsified, by abducing $\neg black\_white(d)$. It is now the turn of C_1, which is already satisfied by $image(d)$. The proof of $image(d)$ is now complete, and the proof of rule R_3 proceeds to prove $color(d)$. Actually, it was already abduced, so the proof is complete and returns an explanation (#9) in which all constraints C_1, C_2, C_3 and C_4 were involved and satisfied.

The latest choice point is on the abduction of $\neg black\_white(d)$. The procedure backtracks and abduces $black\_white(d)$, which now violates constraint C_4 and does not involve any additional constraint. So, the proof of R_1 continues and needs $color(d)$, which was already abduced. This completes the proof, and yields a further explanation (#10) that satisfies constraints C_1, C_2 and C_3 and violates constraint C_4.

The next choice point is on $\neg text(d)$. Changing it to $text(d)$ violates constraint C_2 and involves the *or* constraint C_1, also satisfying it. Again, the proof of R_3 continues and terminates positively thanks to the already abduced literal $color(d)$. The explanation (#11) associated to this proof satisfied constraints C_1 and C_3 and violated constraint C_2.

The last backtracking occurs on the abduced literal $color(d)$. Changing it to $\neg color(d)$ violates constraint C_3, but also prevents the proof of the last literal in R_3, making this world impossible.

So, there are overall 11 possible worlds in which G is proven, whose probability can be computed using (3) as follows. In the first world $E_1 = (L_1, \Delta_1, \mathcal{C}_1)$, we have that

$L_1 = \{R_1, F_1\}$,

$\Delta_1 = \{\neg image(d), table(d), black\_white(d), \neg color(d)\}$,

$$\mathcal{C}_1 = \{C_1, C_2, C_4\},$$

$$\begin{aligned} p(L_1, \Delta_1, \mathcal{C}_1) &= p(R_1) \cdot p(F_1) \cdot \\ &\quad \cdot p(\neg image(d)) \cdot p(table(d)) \cdot p(black\_white(d)) \cdot p(\neg color(d)) \cdot \\ &\quad \cdot p(C_1) \cdot p(C_2) \cdot p(C_4) = \\ &= 1.0 \cdot 1.0 \cdot (1 - 0.4) \cdot 0.3 \cdot 0.7 \cdot (1 - 0.2) \cdot 0.9 \cdot 0.6 \cdot 0.8 = \\ &= 1.0 \cdot 1.0 \cdot 0.6 \cdot 0.3 \cdot 0.7 \cdot 0.8 \cdot 0.9 \cdot 0.6 \cdot 0.8 = 0.0435456 \end{aligned}$$

In the second world $E_2 = (L_2, \Delta_2, \mathcal{C}_2)$, we have that
$L_2 = \{R_1, F_1\}$,
$\Delta_2 = \{\neg image(d), table(d), black\_white(d), color(d), \neg text(d)\}$,
$\mathcal{C}_2 = \{C_1, C_2, \overline{C_3}, \overline{C_4}\}$,

$$\begin{aligned} p(L_2, \Delta_2, \mathcal{C}_2) &= p(R_1) \cdot p(F_1) \cdot \\ &\quad \cdot p(\neg image(d)) \cdot p(table(d)) \cdot p(black\_white(d)) \cdot p(color(d)) \\ &\quad \cdot p(\neg text(d)) \cdot \\ &\quad \cdot p(C_1) \cdot p(C_2) \cdot p(\overline{C_3}) \cdot p(\overline{C_4}) = \\ &= 1.0 \cdot 1.0 \cdot (1 - 0.4) \cdot 0.3 \cdot 0.7 \cdot 0.2 \cdot (1 - 0.8) \cdot 0.9 \cdot 0.6 \\ &\quad \cdot (1 - 0.7) \cdot (1 - 0.8) = \\ &= 1.0 \cdot 1.0 \cdot 0.6 \cdot 0.3 \cdot 0.7 \cdot 0.2 \cdot 0.2 \cdot 0.9 \cdot 0.6 \cdot 0.3 \cdot 0.2 = 0.000653184 \end{aligned}$$

and so on, as reported in Table 2. Thus, the abductive explanation having maximum likelihood is E_8, since $p(E_8) = 0.27 \geq p(E_i), i = 1, \ldots, 11$.

5 Conclusions

Reasoning in complex contexts often requires pure deductive reasoning to be supported by techniques that can cope with incomplete and uncertain data. Abductive inference allows to guess information that has not been observed, as long as it is consistent with given constraints. The abductive setting naturally calls for probability handling: first, because there is a need to assess the reliability of the abduced information; second, because often the constraints are not universally valid laws, but hold with some degree of certainty.

This paper introduced the PEALP (Probabilistic Expressive Abductive Logic Programming) framework, an extension of the traditional ALP framework that can handle more complex kinds of constraints and allows a fully probabilistic approach involving all of its components. It provided computational procedures to handle the extended framework. Based on a 'possible worlds' perspective, these procedures return the most reliable explanation of a given goal.

Future work will concern an investigation of the properties of the proposed proof procedure, a more formal definition of its semantics (especially concerning the probabilistic setting), and running experiments aimed at assessing its practical performance. Also, it would be interesting to further extend the approach to integrity constraints of any kind, obtained by nested combinations of the generalized constraints proposed in this paper, in order to obtain constraints that are full-fledged logical expressions. Another direction for future work is the optimization of the proposed procedure in order to reduce the computational burden required to carry out abduction. Finally, we will investigate the possibility

of learning PEALPs using supervised Machine Learning approaches, and exploiting the learned models for classification tasks.

References

Alberti, M., Bellodi, E., Cota, G., Lamma, E., Riguzzi, F., Zese, R. (2016). Probabilistic constraint logic theories. In Hommersom, A., & Abdallah, S. (Eds.) *Proceedings of the 3rd International Workshop on Probabilistic Logic Programming (PLP-2016), co-located with 26th International Conference on Inductive Logic Programming (ILP 2016), CEUR Workshop Proceedings*, (Vol. 1661 pp. 15-28).

Arvanitis, A., Muggleton, S.H., Chen, J., Watanabe, H. (2006). 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*.

Christiansen, H. (2008). Implementing probabilistic abductive logic programming with constraint handling rules. In Schrijvers, T., & Frühwirth, T. (Eds.) *Constraint handling rules, lecture notes in computer science, vol 5388, springer* (pp. 85-118).

Clark, K.L. (1978). Negation as failure. In Gallaire, H., & Minker, J. (Eds.) *Logic and Databases, Plenum Press* (pp. 293-322).

De Raedt, L., & Kersting, K. (2008). Probabilistic inductive logic programming. In *Probabilistic inductive logic programming, springer, lecture notes in artificial intelligence*, (Vol. 4911 pp. 1-27).

De Raedt, L., Kimmig, A., Toivonen, H. (2007). Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)* (pp. 2462–2467).

Denecker, M., & Schreye, D.D. (1992). Sldnfa: An abductive procedure for normal abductive programs. In *Proceedings of ICSLP, MIT Press* (pp. 700–868).

Fierens, D., Van den Broeck, G., Bruynooghe, M., De Raedt, L. (2012). Constraints for probabilistic logic programming. In Roy, D., Mansinghka, V., Goodman, N. (Eds.) *Proceedings of the NIPS probabilistic programming workshop*.

Fung, T.H., & Kowalski, R.A. (1997). The IFF proof procedure for abductive logic programming. *The Journal of Logic Programming*, *33*, 151–165.

Getoor, L.C. (2002). Learning statistical models from relational data. PhD thesis, Stanford, CA, USA, aAI3038093.

Kakas, A., & Mancarella, P. (1990a). Abductive logic programming. In *Proceedings of NACLP workshop on non-monotonic reasoning and logic programming*.

Kakas, A., & Mancarella, P. (1990b). Database updates through abduction. In *Proceedings of the 16th VLDB, Morgan Kaufmann* (pp. 650–661).

Kakas, A.C., & Mancarella, P. (1990c). On the relation of truth maintenance and abduction. In *Proceedings of the 1st pacific rim international conference on artificial intelligence*.

Kakas, A.C., Kowalski, R.A., Toni, F. (1992). Abductive logic programming. *Journal of Logic and Computation*, *2*, 719–770.

Kate, R.J., & Mooney, R.J. (2009). Probabilistic abduction using markov logic networks. In *Proceedings of the IJCAI-09 Workshop on Plan, Activity and Intent Recognition (PAIR-09), Pasadena, CA*.

Lloyd, J. (1987). Foundations of Logic Programming, 2nd edn. Springer.

Muggleton, S. (1996). Stochastic logic programs De Raedt, L. (Ed.), (Vol. 32.

Nilsson, N. (1986). Probabilistic logic. *Artificial Intelligence*, *28*, 71–87.

Poole, D. (1993). Probabilistic horn abduction and bayesian networks. *Artificial Intelligence*, *64*(1), 81–129.

Raghavan, S.V. (2011). Bayesian abductive logic programs: a probabilistic logic for abductive reasoning. In Walsh, T. (Ed.) *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)* (pp. 2840-2841).

Richardson, M., & Domingos, P. (2006). Markov logic networks. *Machine Learning*, *62*, 107–136.

Rotella, F., & Ferilli, S. (2013). Probabilistic abductive logic programming using possible worlds. In *Proceedings of the 10th Italian Convention on Computational Logic (CILC-2013), Central Europe (CEUR) Workshop Proceedings*, (Vol. 1068 pp. 131-145).

Sato, T. (2002). EM Learning for symbolic-statistical models in statistical abduction. In *Progress in discovery science ,final report of the japanese discovery science project, Springer* (pp. 189-200).

Vennekens, J., Verbaeten, S., Bruynooghe, M. (2004). Logic programs with annotated disjunctions. In Demoen, B., & Lifschitz, V. (Eds.) *Programming, Logic* (pp. 431-445). Berlin: Springer.