# On Preferring and Inspecting Abductive Models

Luís Moniz Pereira[1], Pierangelo Dell'Acqua[2], and Gonçalo Lopes[1]

[1] Departamento de Informática, Centro de Inteligência Artificial (CENTRIA), Universidade Nova de Lisboa 2829-516 Caparica, Portugal
`lmp@di.fct.unl.pt` `goncaloclopes@gmail.com`
[2] Dept. of Science and Technology - ITN, Linköping University Norrköping, Sweden
`pierangelo.dellacqua@itn.liu.se`

**Abstract.** This work proposes the application of preferences over abductive logic programs as an appealing declarative formalism to model choice situations. In particular, both a priori and a posteriori handling of preferences between abductive extensions of a theory are addressed as complementary and essential mechanisms in a broader framework for abductive reasoning. Furthermore, both of these choice mechanisms are combined with other formalisms for decision making, like economic decision theory, resulting in theories containing the best advantages from both qualitative and quantitative formalisms. Several examples are presented throughout to illustrate the enounced methodologies. These have been tested in our implementation, which we explain in detail.

**Key words.** Abduction, Preferences, Logic Programming, XSB-Prolog, Smodels

## 1 Introduction

Much work in logic program semantics and procedures has focused on preferences between rules of a theory [5] and among theory literals [1, 2], with or without updates. However, the exploration of the application of preferences to abductive extensions of a theory has still much to progress. An abductive extension is, by definition, a defeasible construct, and allows greater flexibility in enforcing preference relations. From our perspective, handling preferences over abductive logic programs has several advantages, and allows for easier and more concise translation into normal logic programs (NLP) than those prescribed by more general and complex rule preference frameworks.

In [5], a preliminary theory of revisable preferences between abducible literals was presented, and applied to theory revision, along with a formal semantics based on the definition of *abductive stable models*, illustrated in the paper with a number of applications and applied also to mutual preference revision in [15]. In [10] pre- and post-preferences on abducibles are employed to prospectively generate and filter likely agent preferred future scenarios. In [12], pre- and post-preferences on abducibles are applied to model and solve classic moral dilemmas.

Here we broaden the framework to account for more flexible and powerful means to express preferences between abducibles, over and above a priori relevancy rules embedded in a program's theory. We show there are many advantages as well to preferring a posteriori, i.e. to enact preferences on the computed models, after the consequences of opting for one or another abducible are known, by means of inspection points that examine specific side-effects of abduction. The advantages of so proceeding stem largely from avoiding combinatory explosions of abductive solutions, by filtering both irrelevant as well as less preferred abducibles. We combine these choice mechanisms with other formalisms for decision making, resulting in theories containing the best advantages from both qualitative and quantitative formalisms. Several examples are presented throughout to illustrate the enounced methodologies. These have been tested in our implementation, which we explain in detail.

## 2 Abductive Framework

### 2.1 Basic Abductive Language

Let $\mathcal{L}$ be a first order propositional language defined as follows. As usual, we study non-ground programs and their declarative semantics in terms of the set of all their ground instances, so that grounded literals can be envisaged as propositional constants. This does not preclude employing non-ground abductive programs for knowledge representation, though their practical use requires of course correct implementation techniques regarding non-ground abduction, an issue shared by constructive negation, so that the two are usefully combined. An inroad into these implementation techniques is to be found in [8].

Assume given an alphabet (set) of propositional atoms containing the reserved atom $\bot$ to denote falsity. A literal in $\mathcal{L}$ is an atom $A$ or its default negation $notA$, the latter expressing that the atom is false by default (CWA).

**Definition 1.** *A rule in $\mathcal{L}$ takes the form $A \leftarrow L_1, \ldots, L_t$ where $A$ is an atom and $L_1, \ldots, L_t$ ($t \geq 0$) are literals.*

We follow the standard convention and call $A$ the head of the rule, and the conjunction $L_1, \ldots, L_t$ its body. When $t = 0$ we write the rule simply as $A$, that is without '$\leftarrow$'. An integrity constraint is a rule whose head is $\bot$.

**Definition 2.** *A goal or query in $\mathcal{L}$ has the form $?{-}L_1, \ldots, L_t$, where $L_1, \ldots, L_t$ ($t \geq 1$) are literals.*

A (logic) program $P$ over $\mathcal{L}$ is a finite (countable) set of rules. We adopt the convention of using ';' to separate rules, thus we write a program $P$ as $\{\text{rule}_1; \ldots; \text{rule}_n\}$.

Every program $P$ is associated with a set of abducibles $\mathcal{A}$ consisting of literals which (without loss of generality) do not appear in any rule head of $P$. Abducibles may be thought of as hypotheses that can be used to extend the current theory in order to provide hypothetical solutions or possible explanations for given queries. Given an abductive solution, to test whether a certain abducible has been abduced, $\mathcal{L}$ contains the reserved abducible $abduced(a)$, for every abducible $a \neq abduced(.)$ in $\mathcal{L}$. Thus, $abduced(a)$ acts as a constraint that is satisfied in the solution if the abducible $a$ is indeed assumed. It can be construed as meta-abduction in the form of abducing to check (or passively verify) that a certain abduction is adopted.

*Example 1.* Let $P = \{p \leftarrow \mathit{abduced}(a), a; q \leftarrow \mathit{abduced}(b)\}$ with set of abducibles $\mathcal{A}_P = \{a, b, \mathit{abduced}(a), \mathit{abduced}(b)\}$. Then, $P$ has four intended models: $M = \{\}$, $M_2 = \{p, a, \mathit{abduced}(a)\}$, $M_3 = \{q, b, \mathit{abduced}(b)\}$, and $M_4 = \{p, q, a, b, \mathit{abduced}(a), \mathit{abduced}(b)\}$. The set $\{q, \mathit{abduced}(b)\}$ is not an intended model since the assumption of $\mathit{abduced}(b)$ requires the assumption of $b$.

Given a set of abducibles $\mathcal{A}$, we write $\mathcal{A}^*$ to indicate the subset of $\mathcal{A}$ consisting of all the abducibles in $\mathcal{A}$ distinct from $\mathit{abduced}(.)$, that is:

$$\mathcal{A}^* = \{\ a\ :\ a \neq \mathit{abduced}(.) \text{ and } a \in \mathcal{A}\}.$$

**Hypotheses Generation** The production of alternative explanations for a query is a central problem in abduction, because of the combinatorial explosion of possible explanations. In our approach, preferences among abducibles can be expressed in order to discard irrelevant assumptions. The notion of expectation is employed to express preconditions for enabling the assumption of an abducible. An abducible can be assumed only if there is an expectation for it, and there is not an expectation to the contrary. In this case, we say that the abducible is *considered*. These expectations are expressed by the following rules, for any given abducible $a \in \mathcal{A}^*$:

$$\begin{aligned} \mathit{expect}(a) &\leftarrow L_1, \ldots, L_t \\ \mathit{expect\_not}(a) &\leftarrow L_1, \ldots, L_t \end{aligned}$$

Note that $\mathcal{L}$ does not contain atoms of the form $\mathit{expect}(\mathit{abduced}(.))$ and of the form $\mathit{expect\_not}(\mathit{abduced}(.))$.

*Example 2.* Let $P = \{p \leftarrow a; q \leftarrow b; \mathit{expect}(a); \mathit{expect}(b); \mathit{expect\_not}(a) \leftarrow q\}$ with set of abducibles $\mathcal{A}_P = \{a, b, \mathit{abduced}(a), \mathit{abduced}(b)\}$. Then, $P$ has three intended models: $M = \{\mathit{expect}(a), \mathit{expect}(b)\}$, $M_2 = \{p, a, \mathit{abduced}(a), \mathit{expect}(a), \mathit{expect}(b)\}$ and $M_3 = \{q, b, \mathit{abduced}(b), \mathit{expect}(a), \mathit{expect}(b)\}$. It is not possible to assume both $a$ and $b$ because the assumption of $b$ makes $q$ true which in turn makes $\mathit{expect\_not}(a)$ true preventing $a$ to be assumed.

This notion of considered abducible allows to divide the abductive process into two distinct moments: the generation of hypotheses and the pruning of the unpreferred ones. Computation of preferences between models is problematic when both generation and comparison get mixed up, as mentioned in [2] which, however, does not introduce abductive preference and defeasibility also into generation like we do, but relegates all preference handling to posterior filtering. The two approaches could be conjoined.

**Enforced Abduction** To express that the assumption of an abducible enforces the assumption of another abducible, $\mathcal{L}$ contains reserved atoms of the form $a \prec b$, for any abducibles $a, b \in \mathcal{A}^*$. The atom $a \prec b$ states that the assumption of $b$ enforces the assumption of $a$, active abduction behavior. That is, if $b$ is assumed, then $a \prec b$ forces $a$ to be assumed provided that $a$ can be considered. Note that the abducibles $a, b$ are both required to be different from $\mathit{abduced}(.)$ since they belong to $\mathcal{A}^*$.

*Example 3.* Let $P = \{p \leftarrow a; a \prec b; b \prec a; \mathit{expect}(a); \mathit{expect}(b)\}$ with set of abducibles $\mathcal{A}_P = \{a, b, \mathit{abduced}(a), \mathit{abduced}(b)\}$. Then, $P$ has two intended models for abduction: $M = \{a \prec b, b \prec a, \mathit{expect}(a), \mathit{expect}(b)\}$ and $M_2 = \{a \prec b, b \prec a, a, b, p, \mathit{expect}(a), \mathit{expect}(b), \mathit{abduced}(a), \mathit{abduced}(b)\}$ due to active abduction behavior of $a \prec b$ and $b \prec a$ preventing intended models containing either $a$ or $b$ alone.

**Conditional Abduction** The assumption of an abducible $a$ can be conditional on the assumption of another abducible $b$. The reserved atom $a \sqsubset b$ in $\mathcal{L}$, for any abducible $a, b \in \mathcal{A}^*$, states that $a$ can be assumed only if $b$ is (without assuming it for the purpose of having $a$), passive abduction behavior. That is, $a \sqsubset b$ acts as a check passively constraining the assumption of $a$ to the assumption of $b$ (passive abduction behavior). Note that the abducibles $a, b$ are required to be different from $\mathit{abduced}(.)$.

*Example 4.* Let $P = \{p \leftarrow a; q \leftarrow b; a \sqsubset b; \mathit{expect}(a)\}$ with set of abducibles $\mathcal{A}_P = \{a, b, \mathit{abduced}(a), \mathit{abduced}(b)\}$. Then, there exists only one intended model of $P$: $M = \{a \sqsubset b, \mathit{expect}(a)\}$. Note that $M_2 = \{a \sqsubset b, b, q, \mathit{expect}(a), \mathit{abduced}(b)\}$ and $M_3 = \{a \sqsubset b, a, p, \mathit{expect}(a), \mathit{abduced}(a)\}$ are not intended models. In fact, $M_2$ is not a model since $b$ cannot be assumed (there is no expectation for it). This fact also prevents the assumption of $a$ (due to $a \sqsubset b$) and consequently $M_3$ is not a model.

**Cardinality Constrained Abduction** To constrain the number of assumed abducibles, $\mathcal{L}$ contains reserved atoms of the form $L\ \{l_1, \ldots, l_n\}\ U$ where $n \geq 1$, every $l_i$ is an abducible in $\mathcal{A}$, and $L$ and $U$ are natural numbers representing, respectively, the lower and upper bounds on the cardinality of abducibles. $L\ \{l_1, \ldots, l_n\}\ U$ states that at least $L$ and at most $U$ abducibles in $\{l_1, \ldots, l_n\}$ must be assumed (active abduction behavior). Since the abducibles $l_i$ belong to $\mathcal{A}$, they can also take the form $\mathit{abduced}(.)$.

### 2.2 Declarative Semantics

The declarative semantics of programs over $\mathcal{L}$ is given in terms of abductive stable models. Let $P$ be a program and $\mathcal{A}_P$ the abducibles in $P$. A 2-valued interpretation $M$ of $\mathcal{L}$ is any set of literals from $\mathcal{L}$ that satisfies the condition that, for any atom $A$, precisely one of the literals $A$ or $\mathit{not}\ A$ belongs to $M$. Interpretation $M$ satisfies a conjunction of literals $L_1, \ldots, L_t$, if every literal $L_i$ in the conjunction belongs to $M$. We need the notion of default assumptions of $P$ with respect to an interpretation $M$, where a default literal $\mathit{not}\ A$ is considered an atom, and $\mathit{not}\ \mathit{not}\ A \equiv A$.

$$\begin{aligned}&\mathit{Default}(P, M) = \\ &\quad \{\mathit{not}\ A : \text{there exists no rule } A \leftarrow L_1, \ldots, L_t \text{ in } P \text{ such that } M \models L_1, \ldots, L_t\}\end{aligned}$$

Abducibles are false by default since we made the assumption that abducibles are not defined by any rule in $P$. An interpretation $M$ is a stable model of $P$ iff:

1. $M \not\models \bot$ and
2. $M = \mathit{least}(P \cup \mathit{Default}(P, M))$, where $\mathit{least}$ indicates the least model.

Let $C$ be $L\,\{l_1, \ldots, l_n\}\,U$. Then, we let $W(C, M)$ be the number of abducibles in $\{l_1, \ldots, l_n\}$ satisfied by an interpretation $M$:

$$W(C, M) = |\,\{l : l \in \{l_1, \ldots, l_n\} \text{ and } M \vDash l\}\,|$$

Given a set of abducibles $\Delta$, we write $\Delta^*$ to indicate :

$$\Delta^* = \{a \,:\, a \neq \mathit{abduced}(.) \text{ and } a \in \Delta\}$$

**Definition 3.** *Let $\Delta \subseteq \mathcal{A}_P$ be a set of abducibles. $M$ is an abductive stable model with hypotheses $\Delta$ of $P$ iff:*

1. *$M \nvDash \bot$*
2. *$M = \mathit{least}(Q \cup \mathit{Default}(Q, M))$, where $Q = P \cup \Delta$*
3. *$M \vDash \mathit{expect}(a)$ and $M \nvDash \mathit{expect\_not}(a)$, for every $a \in \Delta^*$*
4. *for every $a \in \Delta^*$, if $M \vDash a$ then $M \vDash \mathit{abduced}(a)$*
5. *for every atom $a \prec b$, if $M \vDash a \prec b$, $M \vDash \mathit{expect}(a)$, $M \nvDash \mathit{expect\_not}(a)$ and $M \vDash b$, then $M \vDash a$*
6. *for every atom $C$ of the form $L\,\{l_1, \ldots, l_n\}\,U$, if $M \models C$ then $L \leq W(C, M)$ $\leq U$*
7. *for every $a \in \Delta^*$, if $M \vDash \mathit{abduced}(a)$ then $M \vDash a$*
8. *for every atom $a \sqsubset b$, if $M \vDash a \sqsubset b$ and $M \vDash a$, then $M \vDash b$*

*Example 5.* Let $P = \{p \leftarrow \mathit{abduced}(a); \mathit{expect}(a)\}$ and $\mathcal{A}_P = \{a, \mathit{abduced}(a)\}$. Then, $M = \{p, a, \mathit{abduced}(a), \mathit{expect}(a)\}$ is an abductive stable model with hypotheses $\Delta = \{a, \mathit{abduced}(a)\}$, while $M_2 = \{p, \mathit{abduced}(a), \mathit{expect}(a)\}$ is not since condition (7) of Def. 3 is not fulfilled.

**Definition 4.** *$\Delta$ is an abductive explanation for goal $G$ in $P$ iff there exists a model $M$ such that $M$ is an abductive stable model with hypotheses $\Delta$ of $P$ and $M \vDash G$.*

**Definition 5.** *$\Delta$ is a strict abductive explanation for goal $G$ in $P$ iff there exists a model $M$ such that:*

1. *$\Delta$ is a minimal set for which:*
   - *$M \vDash G$*
   - *$M \nvDash \bot$*
   - *$M = \mathit{least}(Q \cup \mathit{Default}(Q, M))$, where $Q = P \cup \Delta$*
   - *$M \vDash \mathit{expect}(a)$ and $M \nvDash \mathit{expect\_not}(a)$, for every $a \in \Delta^*$*
   - *for every $a \in \Delta^*$, if $M \vDash a$ then $M \vDash \mathit{abduced}(a)$*
   - *for every atom $a \prec b$, if $M \vDash a \prec b$, $M \vDash \mathit{expect}(a)$, $M \nvDash \mathit{expect\_not}(a)$ and $M \vDash b$, then $M \vDash a$*
   - *for every atom $C$ of the form $L\,\{l_1, \ldots, l_n\}\,U$, if $M \models C$ then $L \leq W(C, M) \leq U$*
2. *for every $a \in \Delta^*$, if $M \vDash \mathit{abduced}(a)$ then $M \vDash a$*
3. *for every atom $a \sqsubset b$, if $M \vDash a \sqsubset b$ and $M \vDash a$, then $M \vDash b$*

Note that in Def. 5 condition (2) is not subject to minimization. The reason for this is clarified by the next example.

*Example 6.* Reconsider $P$ of Example 5. Suppose that the goal $G$ is $?{-}p$. It holds that $\Delta = \{a, \mathit{abduced}(a)\}$ is an abductive explanation for $G$ in $P$, but it is not strict since $\Delta$ is not a minimal set satisfying condition (1) of Def. 5. Indeed, the minimal set is $\Delta_2 = \{\mathit{abduced}(a)\}$. Hence, there exists no strict abductive explanation for $G$ in $P$.

The following property relates abductive explanations to strict abductive explanations.

**Proposition 1.** *Let $G$ be a goal and $\Delta$ a strict abductive explanation for $G$ in $P$. Then, $\Delta$ is an abductive explanation for $G$ in $P$.*

## 3 Pragmatics

### 3.1 Constraining Abduction

Quite often in domain problems it happens the assumption of abducibles is subject to the fulfillment of certain conditions, including other assumptions, which must be satisfied. This requirement is expressed by exploiting constrained abduction, $a \sqsubset b$.

*Example 7.* Consider a scenario where there is a pub that is open or closed. If the light is on in the pub then it is open or being cleaned. Late at night, one can assume the pub is open if there are people inside. The pub being located in an entertainment district, there is noise around if there are people in the pub or a party nearby. This scenario is described by the program $P$ with $\mathcal{A}_P = \{\mathit{open}, \mathit{cleaning}, \mathit{party}, \mathit{people}, \mathit{abduced}(\mathit{open}), \mathit{abduced}(\mathit{cleaning}), \mathit{abduced}(\mathit{party}), \mathit{abduced}(\mathit{people})\}$.

$$\begin{array}{l}
\mathit{light} \leftarrow \mathit{open}, \mathit{not\ cleaning} \\
\mathit{light} \leftarrow \mathit{cleaning}, \mathit{not\ open}, \mathit{not\ abduced}(\mathit{people}) \\
\mathit{open} \sqsubset \mathit{people} \leftarrow \mathit{late\_night} \\
\mathit{noise} \leftarrow \mathit{party} \\
\mathit{noise} \leftarrow \mathit{people}
\end{array}$$

$$\mathit{expect}(\mathit{open}) \quad \mathit{expect}(\mathit{cleaning}) \quad \mathit{expect}(\mathit{party}) \quad \mathit{expect}(\mathit{people})$$

Thus, in case it is night (but not late night) and one does observe lights in the pub, then one has two equally plausible explanations for it: $\{\mathit{open}\}$ or $\{\mathit{cleaning}\}$. Otherwise (late night is a given fact), there is a single explanation for the lights being on: $\{\mathit{cleaning}\}$. If instead it is late night and one hears noise too (i.e. the query is $?{-}\mathit{light}, \mathit{noise}$), then one will have three abductive explanations: $\{\mathit{open}, \mathit{people}\}$, $\{\mathit{party}, \mathit{cleaning}\}$ and $\{\mathit{open}, \mathit{party}, \mathit{people}\}$. The last one reflects that the pub may be open with late customers simultaneously with a party nearby, both events producing noise.

### 3.2 Preferring Abducibles

Now we illustrate how to express preferences between considered abducibles. We employ construct $L\,\langle l_1, \ldots, l_n\rangle\,U$ to constrain the number of abducibles possibly assumed. The construct has a passive abduction behavior and it is defined as:

$$L\,\langle l_1, \ldots, l_n\rangle\,U \equiv L\,\{\mathit{abduce}(l_1), \ldots, \mathit{abduce}(l_n)\}\,U$$

for any abducible $l_1, \ldots, l_n$ in $\mathcal{A}^*$. The next example illustrates the difference between $L\,\langle l_1, \ldots, l_n\rangle\,U$ and $L\,\{l_1, \ldots, l_n\}\,U$.

*Example 8.* Consider a situation where Claire drinks either tea or coffee (but not both). Suppose that Claire prefers coffee over tea when sleepy, and doesn't drink coffee when she has high blood pressure. This situation is described by program $P$ over $\mathcal{L}$ with abducibles $\mathcal{A}_P = \{tea, coffee, abduced(tea), abduced(coffee)\}$:

$$\begin{array}{ll}
drink \leftarrow tea & expect(tea) \\
drink \leftarrow coffee & expect(coffee) \\
 & expect\_not(coffee) \leftarrow blood\_pressure\_high \\
0\ \langle tea, coffee \rangle\ 1 & \\
coffee \prec tea \leftarrow sleepy &
\end{array}$$

In the abductive stable model semantics, this program has two models, one with tea the other with coffee. Adding literal $sleepy$, enforced abduction comes into play, defeating the abductive stable model where only $tea$ is present (due to the impossibility of simultaneously abducing coffee). If later we add $blood\_pressure\_high$, coffee is no longer expected, and the transformed preference rule no longer defeats the abduction of $tea$ which then becomes the single abductive stable model, despite the presence of $sleepy$.

## 3.3 Abducible Sets

Often it is desirable not only to include rules about expectations for single abducibles, but also to express contextual information constraining the powerset of abducibles.

*Example 9.* Consider a situation where Claire is deciding what to have for a meal from a limited buffet. The menu has appetizers (which Claire doesn't mind skipping, unless she's very hungry), three main dishes, from which one can select a maximum of two, and drinks, from which she will have a single one. The situation, with all possible choices, can be modelled by program $P$ over $\mathcal{L}$ with set of abducibles $\mathcal{A}_P = \{bread, salad, cheese, fish, meat, veggie, wine, juice, water, abduced(bread), abduced(salad), abduced(cheese), abduced(fish), abduced(meat), abduced(veggie), abduced(wine), abduced(juice), abduced(water)\}$:

$$\begin{array}{ll}
0\ \{bread, salad, cheese\}\ 3 \leftarrow appetizers & main\_dishes \prec appetizers \\
1\ \{fish, meat, veggie\}\ 2 \leftarrow main\_dishes & drinks \prec appetizers \\
1\ \{wine, juice, water\}\ 1 \leftarrow drinks & appetizers \leftarrow very\_hungry \\
2\ \{appetizers, main\_dishes, drinks\}\ 3 &
\end{array}$$

Here we model appetizers as the least preferred set from those available for the meal. It shows we can condition sets of abducibles based on the generation of literals from other cardinality constraints plus preferences among them.

## 3.4 Modeling Inspection Points

When finding an abductive solution for a query, one may want to check whether some other literals become true or false strictly within the abductive solution found, without performing additional abductions, and without having to produce a complete model to do so. Pereira and Pinto [11] argue this type of reasoning requires a new mechanism. To achieve it, they introduce the concept of inspection point, and show how to employ it to investigate side-effects of interest. Procedurally, inspection points are construed as a form of meta-abduction, by "meta-abducing" the specific abduction of checking (i.e. passively verifying) that a corresponding concrete abduction is indeed adopted. That is, one abduces the checking of some abducible $A$, and the check consists in confirming that $A$ is part of the abductive solution by matching it with the object of the abduced check.

In their approach the side-effects of interest are explicitly indicated by the user wrapping the corresponding goals with a reserved construct $inspect/1$. Procedurally, $inspect$ goals must be solved without abducing regular abducibles, only "meta-abducibles" of the form $abduced/1$.

*Example 10.* Consider the following program taken from [11], where $tear\_gas$, $fire$, and $water\_cannon$ are abducibles.

$$\begin{array}{ll}
\bot \leftarrow police, riot, not\, contain & \\
contain \leftarrow tear\_gas & contain \leftarrow water\_cannon \\
smoke \leftarrow fire & smoke \leftarrow inspect(tear\_gas) \\
police & riot
\end{array}$$

Note the two rules for $smoke$. The first states one explanation for smoke is fire, when assuming the hypothesis $fire$. The second states $tear\_gas$ is also a possible explanation for smoke. However, the presence of tear gas is a much more unlikely situation than the presence of fire; after all, tear gas is only used by police to contain riots and that is truly an exceptional situation. Fires are much more common and spontaneous than riots. For this reason, $fire$ is a much more plausible explanation for $smoke$ and, therefore, in order to let the explanation for $smoke$ be $tear\_gas$, there must be a plausible reason – imposed by some other likely phenomenon. This is represented by $inspect(tear\_gas)$ instead of simply $tear\_gas$. The $inspect$ construct disallows regular abduction – only meta-abduction – to be performed whilst trying to solve $tear\_gas$. I.e., if we take tear gas as an abductive solution for fire, this rule imposes that the step where we abduce $tear\_gas$ is performed elsewhere, not under the derivation tree for $smoke$. The integrity constraint, since there is $police$ and a $riot$, forces $contain$ to be true, and hence, $tear\_gas$ or $water\_cannon$ or both, must be abduced. $smoke$ is only explained if, at the end of the day, $tear\_gas$ is abduced to enact containment. Abductive solutions should be plausible, and $smoke$ is explained by $tear\_gas$ if there is a reason, a best explanation, that makes the presence of tear gas plausible; in this case the riot and the police. Plausibility is an important concept in science in lending credibility to hypotheses.

## 4 A posteriori preferences

A desirable result of encoding abduction semantics over models of a program is that we immediately obtain the consequences of committing to any one hypotheses set. Rules which contain abducibles in their bodies can account for the side-effect derivation of certain positive literals in some models, but not others, possibly triggering integrity constraints or indirectly deriving interesting consequences simply as a result of accepting a hypothesis.

Preferring a posteriori, only after model generation is achieved, is thus needed. However, cause and effect is not enough to draw conclusions and decide, due to the problem of imperfect information and uncertain conditions. To resolve these problems, combining causal models with probabilistic information about models is required.

### 4.1 Utility Theory

Abduction can be seen as a mechanism to enable the generation of the possible futures of an agent, with each abductive stable model representing a possibly reachable scenario of interest. Preferring over abducibles enacts preferences over the imagined future of the agent. In this context, it is unavoidable to deal with uncertainty, a problem decision theory is ready to address using probabilities coupled with utility functions.

*Example 11.* Suppose Claire is spending a day at the beach and she is deciding what means of transportation to adopt. She knows it is usually faster and more comfortable to go by car, but she also knows, because it is hot, there is possibility of a traffic jam. It is also possible to use public transportation (by train), but it will take longer, though it meets her wishes of being more environment friendly. The situation can be modeled by the abductive logic program:

$$\begin{array}{ll} go\_to(beach) \leftarrow car & expect(car) \\ go\_to(beach) \leftarrow train & expect(train) \\ hot & 1\ \{car, train\}\ 1 \end{array}$$

$$\begin{array}{l} probability(traffic\_jam, 0.7) \leftarrow hot \\ probability(not\ traffic\_jam, 0.3) \leftarrow hot \end{array}$$

$$\begin{array}{l} utility(stuck\_in\_traffic, -8) \\ utility(wasting\_time, -4) \end{array}$$

$$\begin{array}{l} utility(comfort, 10) \\ utility(environment\_friendly, 3) \end{array}$$

By assuming each of the abductive hypotheses, the general utility of going to the beach can be computed for each particular scenario:

**Assume car**
Probability of being stuck in traffic = 0.7
Probability of a comfortable ride = 0.3
Expected utility = 10 * 0.3 + 0.7 * -8 = -2.6

**Assume train**
Expected utility = -4 + 3 = -1

It should be clear that enacting preferential reasoning over the utilities computed for each model has to be performed after the scenarios are available, with an a posteriori meta-reasoning over the models and their respective utilities.

## 5 Implementation

The abductive framework described above has been implemented in the ACORDA [10] logic programming system, designed to accomodate abduction in evolving scenarios.

### 5.1 XSB-XASP Interface

Prolog is the most accepted means to codify and execute logic programs, and a useful tool for research and application development in logic programming. Several stable implementations were developed and refined over the years, with plenty of working solutions to pragmatic issues, ranging from efficiency and portability to explorations of language extensions. XSB-Prolog[1] is one of the most sophisticated, powerful, efficient and versatile, focusing on execution efficiency and interaction with external systems, implementing logic program evaluation following the Well-Founded Semantics (WFS).

The semantics of Stable Models is a cornerstone for the definition of some of the most important results in logic programming of the past decade, providing an increase in logic program declarativity and a new paradigm for program evaluation. However, the lack of some important properties of previous language semantics, like relevancy and cumulativity, somewhat reduces its applicability in practice, namely regarding abduction.

The XASP interface [3, 4] (XSB Answer Set Programming), part of XSB-Prolog, is a practical programming interface to Smodels[9]. XASP allows to compute the models of a Normal LP, and also to effectively combine 3- with 2-valued reasoning.

This is gotten by using Smodels to compute the stable models of the residual program, one that results from a query evaluation in XSB using tabling[16]. The residual program is formed by delay lists, sets of undefined literals for which XSB could not find a complete proof, due to mutual loops over default negation in a set, as detected by the tabling mechanism. This method allows obtaining 2-valued models, by completion of the 3-valued ones of XSB. The integration maintains the relevance property for queries over our programs, something Stable Model semantics does not enjoy. In Stable Models, by its very definition, it is necessary to compute whole models of a given program. In the ACORDA implementation framework we sidestep this issue, using XASP to compute the query relevant residual program on demand, usually after some degree of transformation. Only the resulting residual program is sent to Smodels for computation of its abductive stable models. This is one of the main problems which abduction over stable models has been facing: it always needs to consider each abducible in a program, and then progressively defeat those irrelevant for the problem at hand. It is not so in our framework, since we can begin evaluation with a top-down derivation for a query, which immediately constrains the set of abducibles to those relevant to the satisfaction and proof of that query. Each query is conjoined with $not\,\bot$ to ensure Integrity Constraints satisfaction.

An important consideration in computing consequences, cf. Section 4, is not having to compute whole models of the program to obtain just a specific consequences subset,

[1] Both the XSB Logic Programming system and Smodels are freely available at: `http://xsb.sourceforge.net` and `http://www.tcs.hut.fi/Software/smodels`

the one useful to enact a posteriori preferences. This is avoided by computing models just for the residual program corresponding to the consequences we wish to observe. Consequences which are significant for model preference are thus computed on the XSB side, and their residual program is then sent to Smodels. A posteriori preference rules are evaluated over the computed models, and are just consumers of considered abducibles that have indeed been produced, meaning that any additional abductions are disallowed.

### 5.2 Top-Down Proof Procedure

In our implementation we aim for query-driven evaluation of abductive stable models, so that only the relevant part of the program is considered for computation. Computation of such models is performed in two stages. First XSB computes the Well-Founded Model (WFM) of the program w.r.t. the query, by supporting goal derivations on any considered expected abducibles, that are not otherwise being defeated; cf. Section 2.1. We aim to dynamically collect only those abducibles necessary to prove the query. We can assume them neither true nor false at this stage, so they end up undefined in the derivation tree. This is achieved by coding considered abducibles thus:

$$\begin{aligned} consider(A) &\leftarrow expect(A), not\ expect\_not(A), abduce(A) \\ abduce(A) &\leftarrow not\ abduce\_not(A) \\ abduce\_not(A) &\leftarrow not\ abduce(A) \end{aligned}$$

The latter two rules encode an abducible as an even-loop over default negation. They warrant any considered abducible is undefined in the WFM of the program, and hence contained in the residual program computed by the XSB Prolog meta-interpreter; cf. Section 5.1. In this way, any query which is supported on abductive literals will also have its entire derivation tree contained in the XSB's residual program. We can then use this tree as a partial program which is sent to Smodels for model computation. After the stable models of these partial programs are computed, all even-loops will be solved by either assuming the abduction, or assuming its negation.

### 5.3 Program Transformation

Now we consider a first-order propositional language $\mathcal{L}^{\#}$ containing rules as defined in Def. 1. Programs over $\mathcal{L}^{\#}$ are constrained to satisfy given properties.

**Definition 6.** *Let $\Gamma$ and $\Sigma$ be sets of rules over $\mathcal{L}^{\#}$. Then $(\Gamma, \Sigma)$ is a restricted program.*

The basic idea is that $\Gamma$ contains the rules formalizing the application domain while $\Sigma$ formalizes the properties $\Gamma$ must satisfy. Every restricted program is associated with a set of abducibles $\mathcal{A}_{(\Gamma,\Sigma)}$. In the sequel let $\Delta \subseteq \mathcal{A}_{(\Gamma,\Sigma)}$ be some subset.

**Definition 7.** *$M$ is a valid stable model with hypotheses $\Delta$ of $(\Gamma, \Sigma)$ iff $M \nvDash \bot$, $M = least(\Gamma^+ \cup Default(\Gamma^+, M))$, where $\Gamma^+ = \Gamma \cup \Delta$ and $M \vDash \Sigma$.*

A valid stable model is an abductive stable model (conditions (1) and (2)) satisfying the wffs in $\Sigma$ (condition (3)).

**Definition 8.** *$\Delta$ is a valid explanation for goal $G$ in $(\Gamma, \Sigma)$ iff $M$ is a valid stable model with hypotheses $\Delta$ of $(\Gamma, \Sigma)$ and $M \vDash G$.*

**Definition 9.** *$\Delta$ is a strict valid explanation for goal $G$ in $(\Gamma, \Sigma)$ iff:*

1. *$\Delta$ is a minimal set for which:*
   - $M \nvDash \bot$
   - $M = least(\Gamma^+ \cup Default(\Gamma^+, M))$*, where* $\Gamma^+ = \Gamma \cup \Delta$
   - $M \vDash G$
2. $M \vDash \Sigma$

We define transformation $\gamma$ mapping programs over $\mathcal{L}$ to restricted ones over $\mathcal{L}^{\#}$.

**Definition 10.** *Let $P$ be a program over $\mathcal{L}$ with set of abducibles $\mathcal{A}_P$. The restricted program $\gamma(P) = (\Gamma, \Sigma)$ over $\mathcal{L}^{\#}$ with set of abducibles $\mathcal{A}_{(\Gamma,\Sigma)} = \mathcal{A}_P$ is defined as follows.*

*$\Gamma$ consists of:*

1. *all the rules in $P$*
2. $\bot \leftarrow a, not\ expect(a)$
   $\bot \leftarrow a, expect\_not(a)$
   *for every abducible $a \in \mathcal{A}_P^*$*
3. $\bot \leftarrow a, not\ abduced(a)$
   *for every abducible $a \in \mathcal{A}_P^*$*
4. $\bot \leftarrow a \prec b, expect(a), not\ expect\_not(a), b, not\ a$
   *for every rule $a \prec b \leftarrow L_1, \ldots, L_t$ in $P$*
5. $\bot \leftarrow L\,\{l_1, \ldots, l_n\}\,U, count([l_1, \ldots, l_n], N), N \leq L$
   $\bot \leftarrow L\,\{l_1, \ldots, l_n\}\,U, count([l_1, \ldots, l_n], N), N \geq U$
   *for every rule $L\,\{l_1, \ldots, l_n\}\,U \leftarrow L_1, \ldots L_t$ in $P$*

*$\Sigma$ consists of:*

6. $\bot \leftarrow abduced(a), not\ a$
   *for every abducible $a \in \mathcal{A}_P^*$*
7. $\bot \leftarrow a \sqsubset b, a, not\ b$
   *for every rule $a \sqsubset b \leftarrow L_1, \ldots, L_t$ in $P$*

*Remark 1.* We assume given the atom $count([l_1, \ldots, l_n], m)$ that holds if $m$ is the number of abducibles belonging to $[l_1, \ldots, l_n]$ that are assumed. That is, if $C$ is the atom $L\,\{l_1, \ldots, l_n\}\,U$, then we have that $M \vDash count([l_1, \ldots, l_n], m)$ iff $W(C, M) = m$, for any interpretation $M$.

*Remark 2.* If $P$ contains inspection points, then apply the transformation $\gamma$ to $\Pi(P)$ (cf. Definition 6 of [11]) instead of to $P$ directly.

**Theorem 1.** *Let $P$ be a program over $\mathcal{L}$ with set of abducibles $\mathcal{A}_P$. $M$ is an abductive stable model with hypotheses $\Delta \subseteq \mathcal{A}_P$ of $P$ iff $M$ is a valid stable model with hypotheses $\Delta$ of $\gamma(P)$.*

**Theorem 2.** *Let $P$ be a program over $\mathcal{L}$. $\Delta \subseteq \mathcal{A}_P$ is a strict abductive explanation for goal $G$ in $P$ iff $\Delta$ is a strict valid explanation for $G$ in $\gamma(P)$.*

## 5.4 Computation of Abductive Stable Models

In this second stage, Smodels will thus be used to compute abductive stable models from the residual program obtained from top-down goal derivation. Determination of relevant abducibles is performed by examining the residual program for ground literals which are arguments to *consider/1* clauses. Relevant preference rules are pre-evaluated at this stage as well, by querying for any such rules involving any pair of the relevant abducible set.

The XASP package [3] allows the programmer to collect rules in an XSB clause store. When the programmer has determined enough clauses were added to the store to form a semantically complete sub-program, it is then *committed*. This means information in the clauses is copied to Smodels, coded using Smodels data structures, so that stable models of those clauses can be computed and returned to XSB to be examined.

When both the relevant abducibles and preference rules are determined, a variation of transformation $\gamma$ is applied, with every encoded clause being sent to the XASP store, reset beforehand in preparation for the stable models computation. Once the residual program, transformed to enact preferences, is actually committed to Smodels, we obtain through XASP the set of abductive stable models, and identify each one by their choice of abducibles, i.e. those *consider/1* literals collected beforehand in the residual program.

## 5.5 Inspection Points

Given the top-down proof procedure for abduction, implementing inspection points becomes just a matter of adapting the evaluation of derivation subtrees falling under $inspect/1$ literals at meta-interpreter level. Basically, considered abducibles evaluated under $inspect/1$ subtrees are codified thus:

$$\begin{aligned} consider(A) &\leftarrow abduced(A) \\ abduced(A) &\leftarrow not\ abduced\_not(A) \\ abduced\_not(A) &\leftarrow not\ abduced(A) \end{aligned}$$

All $abduced/1$ predicates are collected during computation of the residual program and later checked against the abductive stable models themselves. Every $abduced(a)$ predicate must pair with a corresponding abducible $a$ for the model to be accepted; cf. transformation $\gamma$.

Let $\mathcal{L}^*$ be a first order propositional language defined as $\mathcal{L}$ except that $\mathcal{L}^*$ contains atoms of the form $inspect(.)$, $ea(.,.)$ and $ca(.,.)$, while it does not contain any atom of the form $a \prec b$, $a \sqsubset b$ and $abduced(.)$. The transformation below maps programs over $\mathcal{L}$ into programs over $\mathcal{L}^*$.

**Definition 11.** *Let $P$ be a program over $\mathcal{L}$. Let $Q$ be the set of all the rules obtained by the rules in $P$ by replacing $abduced(a)$ with $inspect(a)$, $a \prec b$ with $ea(a,b)$ and $a \sqsubset b$ with $ca(a,b)$. Program $\Pi(P)$ over $\mathcal{L}^*$ consists of all rules in $Q$ together with rules:*
$\perp \leftarrow not\ a, inspect(b), L_1, \ldots, L_t$ *for every* $ea(a,b) \leftarrow L_1, \ldots, L_t$ *in $Q$,*
$\perp \leftarrow inspect(a), not\ inspect(b), L_1, \ldots, L_t$ *for every* $ca(a,b) \leftarrow L_1, \ldots, L_t$ *in $Q$.*

### 5.6 A Posteriori Choice Mechanisms

If a single model emerges from computation of the abductive stable models, goal evaluation can terminate. When multiple models still remain, there is opportunity to introduce a posteriori choice mechanisms, which are domain-specific for a program. We account for this specificity by providing an implementation hook the user can adopt for introducing specific code for this final selection process, in addition to the a posteriori choice mechanisms in Section 4.

## 6 Conclusions

We have addressed issues of formalizing the combination of abduction with preferences, and its implementation, in an original way. Namely, we introduced the notion of expectable defeasible abducibles, enabled by the situation at hand and guided by the query, and catered for a priori preferences amongst abducibles; all mechanisms designed to cut down on the combinatory explosion of untoward abduction, that is relying only on a posteriori preferences to filter what should not have been generated in the first place. In regard to a posteriori preferences, we defined the notion of inspection points to cater for observing relevant side-effects of abductive solutions, and enable a posteriori choices without generating complete models which show all side-effects. In all, we showed how a combination of cardinality constraints, preferences and inspection points can be used to govern and constrain abduction, and how these mechanisms, coupled with a posteriori ones, are an important tool for knowledge representation in modeling choice situations where agents need to consider present and future contexts to decide about them. Finally, we showed the advantages of implementing our framework as a hybrid state-of-art declarative XSB-Prolog/Smodels system, to efficiently combine top-down query-oriented abductive backward chaining, and side-effect model generating forwards-chaining, respectively for constraining abducibles to preferred ones relevant for a goal, and to compute the consequences of assuming them in each scenario.

We cannot presume to survey and compare the present work with the by now formidable stock of separate works on abduction and on preferences in logic programming, though we have referred to some along the exposition. Our intent has been the rather original introduction of new features mentioned above, and their combination and implementation, and in that respect we know not of competitive similar attempts. However, regarding inspection points, in [14], a technical problem is detected with the IFF abductive proof procedure [7], in what concerns the treatment of negated abducibles in integrity constraints (e.g. their examples 2 and 3). They then specialize IFF to avoid such problems and prove correctness of the new procedure. The problems detected refer to the active use an IC of some not A, where A is an abducible, whereas the intended use should be a passive one, simply checking whether A is proved in the abductive solution found, as in our inspection points (though these more generally apply to any literal). To that effect they replace such occurrences of not A by not provable(A), in order to ensure that no new abductions are allowed during the checking.

In the way of future developments and application topics, in [13], arguments are given as to how epistemic entrenchment can be explicitly expressed as preferential reasoning. And, moreover, how preferences can be employed to determine believe revisions, or, conversely, how belief contractions can lead to the explicit expression of preferences. [6] provides a stimulating survey of opportunities and problems in the use of preferences, reliant on AI techniques.

On a more general note, it appears to us that the practical use and implementation of abduction in knowledge representation and reasoning, by means of declarative languages and systems, has reached a point of maturity, and of opportunity for development, worthy the calling of attention of a wider community of potential practitioners.

## References

1. G. Brewka. Logic programming with ordered disjunction. In M. Kaufmann, editor, *Proc. 18th National Conference on Artificial Intelligence, AAAI-02*, 2002.
2. G. Brewka, I. Niemelä, and M. Truszczynski. Answer set optimization. In *Proc. IJCAI-2003*, pages 867–872, 2003.
3. L. Castro, T. Swift, and D. S. Warren. *XASP: Answer Set Programming with XSB and Smodels*. http://xsb.sourceforge.net/packages/xasp.pdf.
4. L. F. Castro and D. S. Warren. An environment for the exploration of non monotonic logic programs. In A. Kusalik, editor, *Proc. of the 11th Intl. Workshop on Logic Programming Environments (WLPE'01)*, 2001.
5. P. Dell'Acqua and L. M. Pereira. Preferential theory revision. *J. of Applied Logic*, 5(4):586–601, 2007.
6. J. Doyle. Prospects for preferences. *Computational Intelligence*, 20(3):111–136, 2004.
7. T. H. Fung and R. Kowalski. The IFF Proof Procedure for Abductive Logic Programming. *The J. of Logic Programming*, 33(2):151–165, 1997.
8. Neg-Abdual. Constructive Negation with Abduction. Available at: http://centria.di.fct.unl.pt/∼lmp/software/contrNeg.rar.
9. I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In J. D. et al., editor, *4th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning*, LNAI 1265, pages 420–429. Springer, 1997.
10. L. M. Pereira and G. Lopes. Prospective logic agents. In M. Neves and J. et al., editors, *Progress in AI, Procs. 13th Portuguese Intl.Conf. on AI (EPIA'07)*, LNAI 4874, pages 73–86. Springer, 2007.
11. L. M. Pereira and A. M. Pinto. Notes on Inspection Points and Meta-Abduction in Logic Programs. Work in progress at http://centria.di.fct.unl.pt/∼lmp/publications/online-papers/IP08.pdf, 2008.
12. L. M. Pereira and A. Saptawijaya. Modelling morality with prospective logic. In J. M. Neves, M. F. Santos, and J. M. Machado, editors, *Progress in AI, Procs. 13th Portuguese Intl.Conf. on AI (EPIA'07)*, LNAI 4874, pages 99–111. Springer, 2007.
13. H. Rott. *Change, Choice and Inference*. Oxford University Press, Oxford, 2001.
14. F. Sadri and F. Toni. Abduction with Negation as Failure for Active and Reactive Rules. In E. Lamma and P. Mello, editors, *An Implementation for Abductive Logic Agents*, LNAI 1792, pages 49–60. Springer-Verlag, 1999.
15. P. Santana and L. M. Pereira. Emergence of cooperation through mutual preference revision. In M. Ali and D. Richard, editors, *Procs. The 19th Intl.Conf. on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE'06)*, LNAI. Springer, 2006.
16. T. Swift. Tabling for non-monotonic programming. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):201–240, 1999.