Pre and Post Preferences over Abductive Models

Luís Moniz Pereira
CENTRIA - Centro de Inteligência Artificial
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
lmp@di.fct.unl.pt

Gonçalo Lopes
CENTRIA - Centro de Inteligência Artificial
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
goncaloclopes@gmail.com

Pierangelo Dell'Acqua
ITN - Department of Science and Technology
Linköping University
60174 Norrköping, Sweden
pier@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.

1. INTRODUCTION

Much work in logic program semantics and procedures has focused on preferences between rules of a theory [1] and among theory literals [2, 3], with or without updates. However, the exploration of the application of preferences to abductive extensions of a theory has still much to progress. In 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.

We also argue that preferring among abductive extensions is really a much more appealing formalism than that of hard or soft constraints on program literals, since the latter represent formal conclusions to the program, which are often not defeasible. An abductive extension is, by definition, a defeasible construct, and allows greater flexibility in enforcing preference relations. In [6], a preliminary theory of revisable preferences between abducible literals was presented, along with a formal semantics based on the definition of *abductive stable models*. In this work we extend the theoretical framework thence proposed, addressing many problems and limitations that remained to be solved.

We also propose to broaden the framework to account for more flexible and powerful means to express preferences between abducibles, besides a priori relevancy rules embedded in a program's theory. In fact, we intend to show that there are many advantages as well to prefer a posteriori, i.e. to enact preferences on the computed models, after the consequences of opting for one or another abducible are known. Furthermore, we combine both of these choice mechanisms with other formalisms for decision making, like economic decision theory, resulting in theories containing the best advantages from both qualitative and quantitative formalisms.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.
VLDB '07, September 23-28, 2007, Vienna, Austria.
Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

2. FRAMEWORK

2.1 Language

Let $\mathcal{L}$ be any first order language. A domain literal in $\mathcal{L}$ is a domain atom A or its default negation *not* A, the latter expressing that the atom is false by default (CWA). A domain rule in $\mathcal{L}$ is a rule of the form:

$$A \leftarrow L_1, \ldots, L_t \ \ (t \geq 0)$$

where A is a domain atom and $L_1, \ldots, L_t$ are domain literals. The following convention is used. Given a rule r of the form $L_0 \leftarrow L_1, \ldots, L_t$, we write $H(r)$ to indicate L_0 and $B(r)$ to indicate the conjunction $L_1, \ldots, L_t$. When $t = 0$ we write the rule r simply as L_0.

An integrity constraint in $\mathcal{L}$ is a rule of the form:

$$\bot \leftarrow L_1, \ldots, L_t \ \ (t > 0)$$

where $\bot$ is a domain atom denoting falsity, and $L_1, \ldots, L_t$ are domain literals.

A (logic) program P over $\mathcal{L}$ is a set of domain rules and integrity constraints, standing for all their ground instances. Every program P is associated with a set of abducibles $\mathcal{A} \subseteq \mathcal{L}$, 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.

2.2 Hypotheses Generation

The production of alternative explanations for a query is a central problem in abduction, because of the combinatorial explosion of possible explanations. Thus, it is important to generate only those abductive explanations which are relevant for the problem at hand. Several approaches have thus far been proposed, often based on some global criteria, which has the drawback of generally being domain independent and computationally expensive. An alternative

to global criteria for competing alternative assumptions is to allow the theory to contain rules encoding domain specific information about which particular assumptions are to be considered in a particular situation.

In our approach, preferences among abducibles can be expressed in order to discard unwanted assumptions. Technically, preferences over alternative abducibles will be coded as constraints over even cycles of default negation, under Stable Model semantics [7], and triggering one of the preference rules will break the cycle in favour of one abducible or another. 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 reserved domain rules, for any given abducible $a \in \mathcal{A}$:

$$\begin{aligned} expect(a) &\leftarrow L_1, \ldots, L_t \ \ (t \geq 0) \\ expect\_not(a) &\leftarrow L_1, \ldots, L_t \ \ (t \geq 0) \end{aligned}$$

where every $L_i (1 \leq i \leq t)$ is a domain literal. Consideration of an abducible can thus be expressed, for any given abducible $a \in \mathcal{A}$ by the rule:

$$consider(a) \leftarrow expect(a), not\ expect\_not(a)$$

This notion of considered abducible allows us 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 the generation and comparison get mixed up, as already mentioned in [3], but in our case we introduce a middle-man condition instead of two distinct computations.

Consideration of an abducible could also be encoded as an integrity constraint, such as:

$$\bot \leftarrow expect(a), not\ expect\_not(a), not\ consider(a)$$

but this would prevent our framework from benefitting from query relevance, since all abducibles would have to be evaluated for consideration, even if some of them are irrelevant for the problem.

Although we will base our approach on Stable Models, the lack of the relevance property in this semantics makes it more difficult to constrain abduction to just the relevant abducibles for a query. Results in recent years have provided a partial solution for this issue, by combining Stable Models evaluation with the Well-Founded Semantics [16] in order to guarantee goal-directed evaluation [5]. These approaches guarantee goal-directedness for *partial stable models* and even guarantee equivalence to total stable models for *call-consistent* programs [8, 5]. As goal-directedness is, in our perspective, critical for practical abduction over large knowledge bases, we consider evaluation of abductive logic programs as computation of partial stable models, obtained by restricting evaluation to just the query-relevant part of the program. As such, abduction is effectively restricted to just those literals which are relevant for the query at hand.

2.3 Preferring Abducibles

In order to enact preferences between the considered abducibles, we present the language $\mathcal{L}^*$, an extension of $\mathcal{L}$ including preference rules expressing a relevancy preorder. A preference atom is one of the form $a \lhd b$, where a and b are abducibles, i.e. $a, b \in \mathcal{A}$. $a \lhd b$ means that the abducible a is more relevant than the abducible b. A preference rule in $\mathcal{L}^*$ is one of the form:

$$a \lhd b \leftarrow L_1, \ldots, L_t \ \ (t \geq 0)$$

where $a \lhd b$ is a preference atom and every $L_i (1 \leq i \leq t)$ is a domain literal or a preference literal (i.e. a preference atom or its default negation). A (logic) program P over $\mathcal{L}^*$ is a set of domain rules, integrity constraints and preference rules, standing for their ground instances.

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 $not\ A$ belongs to M. We say that an interpretation M satisfies a conjunction of literals $L_1, \ldots, L_t$, if every literal L_i in the conjunction belongs to M. For each preference rule r having $H(r)$ of the form $a \lhd b$ we say that an interpretation M satisfies r iff whenever b belongs to M and M satisfies $B(r)$, a also belongs to M.

Example 1. Consider a situation where agent Claire will drink either tea or coffee (but not both). Suppose that Claire prefers coffee over tea when sleepy. This situation can be represented by a program Q over $\mathcal{L}^*$ with set of abducibles $\mathcal{A}_Q = \{tea, coffee\}$:

$$\begin{aligned} &drink \leftarrow tea \\ &drink \leftarrow coffee \\ \\ &expect(tea) \\ &expect(coffee) \\ &expect\_not(coffee) \leftarrow blood\_pressure\_high \\ \\ &coffee \lhd tea \leftarrow sleepy \end{aligned}$$

2.4 Abducible Sets

In many situations it is desirable not only to include rules about the expectations for single abducibles, but also to express contextual information constraining the *powerset* of abducibles. For instance, in the previous example we expressed that abducing tea or coffee was mutually exclusive (i.e. only one of them could be abduced), but it is easy to imagine similar choice situations where it would be possible, indeed even desirable, to abduce both, or neither. The behaviour of abducibles over different sets is highly context-dependent, and as such, should also be embedded over rules in the theory.

Overall, the problem is analogous to the ones addressed by *cardinality* and *weight constraint rules* for the Stable Model semantics [10], and below we present how one can nicely import these results to work with abduction of sets, and also hierarchies of sets. We now present the language $\mathcal{L}^+$, an extension of $\mathcal{L}^*$ allowing for cardinality constraints on the domain literals of $\mathcal{L}^*$. A *cardinality constraint* C has the form:

$$L\ \{a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m\}\ U \ \ (m, n \geq 0)$$

where every $a_i, not\ b_i$ are domain literals in $\mathcal{L}^*$ and L, U represent, respectively, the lower and upper bounds on the cardinality of (abduced) literals. Cardinality constraints can also appear in the heads of rules, meaning that in case the body of the rule is satisfied, the cardinality constraint should be enforced. In this way, we can form constrained powersets of abducibles, possibly conditioned on theory literals

With this framework, a simple way to prefer among abducible sets is to extend the preference rules in $\mathcal{L}^*$ to also apply to literals appearing in bodies of cardinality constraint rules. Generally preferring over theory literals is not desirable, as we have argued in Section 1, but in this case, we are merely using the condition literals as a means to *identify* the abducible set, and to allow us the specification of preferences over these identifiers.

To account for this, we extend the preference operator $\lhd$ to work on literals in the body of cardinality constraint rules in $\mathcal{L}^+$, as this will provide us with a logic programming methodology to prefer among abducible sets, as in the following example.

Example 2. Consider a situation where agent 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 the following program R over $\mathcal{L}^+$ with set of abducibles $\mathcal{A}_\mathcal{R} = \{bread, salad, cheese, fish, meat, veggie, wine, juice, water\}$:

$$0\ \{bread, salad, cheese\}\ 3 \leftarrow appetizers$$
$$1\ \{fish, meat, veggie\}\ 2 \leftarrow main\_dishes$$
$$1\ \{wine, juice, water\}\ 1 \leftarrow drinks$$

$$2\ \{appetizers, main\_dishes, drinks\}\ 3$$

$$main\_dishes \lhd appetizers$$
$$drinks \lhd appetizers$$
$$appetizers \leftarrow very\_hungry$$

In this situation we model appetizers as being the least preferred set from those available for the meal. This shows how we can condition sets of abducibles based on the generation of literals from other cardinality constraints along with preferences among such literals.

2.5 Declarative Semantics

In the remainder of this section we let P be a program over $\mathcal{L}^+$, $\mathcal{A}_P$ the set of abducibles of P, and M a 2-valued interpretation of $\mathcal{L}^+$. We write $least(P)$ to indicate the least model of P. We adopt the following two definitions from [1], definition 3 from [6] and definition 4 from [10] and refer the reader to those sources for a detailed exposition.

Definition 1. The set of default assumptions of P with respect to M is:

$$Default(P, M) =$$
$$\{not\ A : \neg\exists r \in P\ such\ that\ H(r) = A\ and\ M \vDash B(r)\}$$

Definition 2. M is a stable model of P iff $M = least(P \cup Default(P, M))$

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

$$M = least(P^+ \cup Default(P^+, M)),\ where\ P^+ = P \cup \Delta$$

Definition 4. M satisfies a cardinality constraint C, written as $M \vDash C$, iff $L \leq W(C, M) \leq U$ where

$$W(C, M) = |\{l \in lit(C) : M \vDash l\}|$$

is the number of literals in C satisfied by M

Definition 5. Let L, B, C be literals in $\mathcal{L}^+$. We say L directly depends on B iff B occurs in the body of some rule in P with head L. We say L depends on B iff L directly depends on B or there is some C such that L directly depends on C and C depends on B. We say that $Rel_L(P)$, the relevant part of P, is the logic program constituted by the set of all rules of P with head L or some B on which L depends on.

Definition 6. Let Q be a program over $\mathcal{L}^+$ with set of abducibles $\mathcal{A}_Q$, M an interpretation of $\mathcal{L}^+$ and G a ground instance of a given query on the program. Let $\Delta \subseteq \mathcal{A}_Q$. M is a preferred relevant abductive partial stable model of Q with hypotheses Δ, total on the set of literals of $Rel_G(Q)$ iff:

1. M is relevant and consistent
2. M defines a preorder over the abducibles in $\mathcal{A}_Q$
3. $M \nvDash \bot$
4. $\forall x\ if M \vDash x$, then $x \in Rel_G(Q)$
5. for every $x, y \in \mathcal{A}_Q$, if $M \vDash x \lhd y$ then $M \nvDash y \lhd x$
6. for every $x, y, z \in \mathcal{A}_Q$, if $M \vDash x \lhd y$ and $M \vDash y \lhd z$, then $M \vDash x \lhd z$
7. for every cardinality constraint rule $r \in Q$, if $M \vDash B(r)$ then $M \vDash H(r)$
8. $M = least(Q^+ \cup Default(Q^+, M))$, with $Q^+ = Q \cup \Delta$
9. for every $a \in \Delta$, $M \vDash expect(a)$ and $M \nvDash expect\_not(a)$
10. if $M \vDash a$ where $a \in \Delta$ or $a \in B(c)$, for some cardinality constraint rule c, then there exists no preference rule r in Q such that:
    - $H(r)$ is $x \lhd a$
    - $M \vDash H(r)$
    - $M \vDash expect(x)$ and $M \nvDash expect\_not(x)$ if $x \in \mathcal{A}_Q$
    - $x \notin \Delta$ if $x \in \mathcal{A}_Q$

In other words, if x is preferred to a and is expected with no contrary expectation, then it must be an assumed abducible if a is.

Note that Δ is any possible subset of $\mathcal{A}_Q$, subject to any cardinality constraints in Q. In this way we allow for specific abducible sets to be defined on Q, and preference rules can be applied to literals which are effectively used to toggle such sets, and hence, allow us to enact preferences among sets.

Example 3. Let Q be the program of Example 1, extended with the following rules and with set of abducibles $\mathcal{A}_\mathcal{Q} = \{tea, coffee, mousse\}$. The new term $afters$ denotes a possible subgoal:

$$afters \leftarrow mousse$$
$$afters \leftarrow coffee$$

$$expect(mousse)$$
$$mousse \lhd coffee$$

1 {tea,coffee,mousse} 1

Q has two alternative explanations $\Delta_1 = \{coffee\}$ and $\Delta_2 = \{tea\}$ for the query *drink*. In fact, Q has two preferred abductive stable models:

$M_1 = \{expect(tea), expect(coffee), coffee, drink\}$ with hypotheses Δ_1

$M_2 = \{expect(tea), expect(coffee), tea, drink\}$ with hypotheses Δ_2

for which $M_1 \models drink$ and $M_2 \models drink$. Note that the abducible *mousse* is irrelevant for the query *drink* and, as such, is not even considered under the relevant part of the program, even if it is a more preferred abducible than *coffee*. The number of models reduces to one if we add *sleepy* to Q. In this case, *coffee* being a more preferred abducible than *tea*, and attending to the added cardinality constraint, the only model of Q is $M_1 \cup \{sleepy\}$.

2.6 Program Transformation

We now present an adapted program transformation from the one set forth in [6], translating a program encoded over $\mathcal{L}^+$ with abducibles into a program encoded as a NLP extended with cardinality constraints, and no abducible literals. The rules for the abducible literals encode the semantics presented in the previous section.

Definition 7. Let Q be a program over $\mathcal{L}^+$, with set of abducibles $\mathcal{A}_Q = \{a_1, \ldots, a_m\}$. The program $P = \Sigma(Q)$ with abducibles $\mathcal{A}_P = \{\}$, relevant under query G, is obtained as follows:

1. P contains all the domain rules in $Rel_G(Q)$
2. P contains all the cardinality constraint rules $r \in Q$ having $H(r)$ of the form $L\ \{l_1, \ldots, l_k\}\ U$, where every $l_i \notin \mathcal{A}_Q$
3. for every cardinality constraint rule r having $H(r)$ of the form $L\ \{b_1, \ldots, b_k\}\ U$, where every $b_i \in \mathcal{A}_Q$, P contains the cardinality constraint rule: $L\ \{l_1 : consider(l_1), \ldots, l_m : consider(l_m)\}\ U \leftarrow B(r)$ where every $l_i \in lit(r)$ and $l_i \in Rel_G(Q)$
4. for every $a_i \in \mathcal{A}_Q$ and $a_i \in Rel_G(Q)$, P contains the domain rule: $consider(a_i) \leftarrow expect(a_i), not\ expect\_not(a_i)$
5. for every preference rule $r \in Rel_G(Q)$, where $H(r) = x \lhd y$, P contains one of the following integrity constraints:
   - $\bot \leftarrow B(r), consider(x), consider(y), y, not\ x$ if $x, y \in \mathcal{A}_Q$
   - $\bot \leftarrow B(r), consider(y), y, not\ x$ if $x \notin \mathcal{A}_Q$ and $y \in \mathcal{A}_Q$
   - $\bot \leftarrow B(r), consider(x), y, not\ x$ if $x \in \mathcal{A}_Q$ and $y \notin \mathcal{A}_Q$
   - $\bot \leftarrow B(r), y, not\ x$ if $x, y \notin \mathcal{A}_Q$

Example 4. Let Q be the program of Example 3. The result of applying the transformation from Definition 7 to the relevant part of Q given the query *drink* is the program $\Sigma(Q)$:

$drink \leftarrow tea$
$drink \leftarrow coffee$

$expect(tea)$
$expect(coffee)$
$expect\_not(coffee) \leftarrow blood\_pressure\_high$

$\bot \leftarrow sleepy, consider(tea), consider(coffee), tea, not\ coffee$

$1\ \{tea : consider(tea), coffee : consider(coffee)\}\ 1$

$consider(tea) \leftarrow expect(tea), not\ expect\_not(tea)$
$consider(coffee) \leftarrow expect(coffee), not\ expect\_not(coffee)$

Following the stable model semantics extended with cardinality constraints, we obtain the two preferred abductive stable models from the previous example. Also, by adding the literal *sleepy*, the integrity constraint comes into play, defeating the abductive stable model where only *tea* is present (due to the impossibility of simultaneously abducing coffee, cf. cardinality constraint). However, if later on we add *blood_pressure_high* to the program, coffee is no longer expected, and as such, 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. A POSTERIORI PREFERENCES

While we can indeed place *a priori* constraints on which abducibles are relevant given contextual knowledge in the situation, more often than not we are only able to enact certain choices after looking at the consequences of adopting one or another abducible. The consequences of each abductive stable model can, and often are, unique to that model, and we cannot model preferences across these consequences during model generation itself. Only after the relevant models are computed can we reason about which consequences, or other features of the models, are determinant for the final choice, i.e. the quality of the model.

One possibility is to consider a quantitative classification of the obtained models, for instance by associating some measure of utility to each choice scenario, like in decision theory. This allows us to consider and integrate many techniques and results from more quantitative decision making frameworks into our own theories, accounting for more elaborate choice models.

When considering abductive logic programs as specifications of choice models of agents, other possibilities come up derived from the capability of the agent to act upon and sense the environment. Namely, if it is the case that the currently available knowledge of the situation is insufficient to commit to any single preferred abductive model, it may be possible for the agent to gather additional information by performing experiments, or consulting an oracle in order to confirm or disconfirm some of the remaining hypotheses. This process may even be nested, so that the available experiments are themselves considered as hypotheses with associated qualitative and quantitative preferences, allowing to express arbitrarily complex context-dependent choices.

We explore these novel approaches in the sequel, along

with some telling examples which intend to show the need for, and illustrate, the proposed reasoning schemes.

Other possibilities, currently under exploration, concern a more qualitative appraisal of models, say in term of general moral or ethical rules that look at the consequences of abduced actions, and which may be tuned with experience.

3.1 The consequences of abduction

A desirable result of encoding abduction semantics over models of a program (where each abducible literal may be assumed or not) is that we immediately obtain the consequences of commiting 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.

Sometimes these computed consequences are relevant to the process of preference handling itself, as we prefer certain consequences to others. However, more often than not it is not possible to simply condition preferences between abducibles based on these consequences, as it may lead to unexpected circular inconsistencies. Also it may be difficult to express more general preferences over what are the preferred literals in a more complete model.

Example 5. Consider the simple abductive logic program presented below, with $\mathcal{A} = \{a, b\}$:

$$\begin{array}{ll} c \leftarrow a & 1\ \{a, b\}\ 1 \\ expect(a) & \\ expect(b) & \end{array}$$

This program has two abductive stable models:

$$M_1 = \{expect(a), expect(b), a, c\}$$

$$M_2 = \{expect(a), expect(b), b\}.$$

Now let us suppose that we want to prefer models where the literal c is not present. We could model this, for this particular program, by stating $b \lhd a \leftarrow c$. However, if we introduced another abducible that directly or indirectly resulted in the derivation of c then we would have to shortcut another preference rule for this abducible. Also, if the derivation of c resulted from the combination of more than one abduction, one would have to extensively encode preferences over these sets to all other possible combinations of literals which didn't derive c.

In the long run, the complexity of writing program rules to account for all possible combinations for c would quickly become unsurmountable. Also, if other interesting literals also suggested other types of preferences over the models, and particularly if these other preferences contradicted the previous ones, inconsistencies could easily arise which would destroy entire models of the program. Also, how could we model more general meta-preferences like the one: 'prefer models which have a greater number of abduced literals'.

In these cases, *a posteriori* reasoning is much more general and powerful to express these kinds of constraints and preference rules which operate on the consequences of the models themselves.

3.2 Utility Theory

Economic decision theory has been well recognized as a comprehensive and well-founded model for describing ideal rational agents, and much work in the field of AI has been undertaken in order to synthesize models of bounded rationality in computational systems. The logic programming field is no exception, with a number of results being imported into logic models of belief, choice and decision making [12]. A particularly interesting field of study regards modelling agents capable of planning ahead their future choices and actions.

Abduction can also 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 in this case is enacting preferences over the imagined future of the agent. In this particular domain, it is unavoidable to deal with uncertainty, a problem that decision theory is ready to address using probability theory coupled with utility functions.

We intend to show that combining the qualitative reasoning addressed in previous sections with the quantitative decision making mechanisms of decision theory is a natural extension to both mechanisms for handling preferences, and neatly accounts for some interesting properties presented by agents in the real world.

In fact, it is possible to associate a quantitative measure of utility to each abductive scenario, by conditioning utility literals on consequences of abducibles. By combining utilities with information regarding the probability of the occurrence of uncertain literals, we end up with an interesting mixture of qualitative and quantitative reasoning, where possible abducibles, constrained by the expectation and preference rules, generate all possible relevant future scenarios which are then associated with a degree of belief to be coupled with the importance that the model be adopted.

Example 6. Suppose that agent Claire is spending a day at the beach and she is deciding what means of transportation to adopt. She knows that usually it is faster and more comfortable to go by car, but she also knows that, because it is hot, there is a possibility that there will be a traffic jam. There is also the possibility of using public transportation (by train), but it will take a lot of time, although it meets her wishes of being more environmentally friendly.

This situation can be modelled by the following abductive logic program:

$$\begin{array}{l} go\_to(beach) \leftarrow car \\ go\_to(beach) \leftarrow train \\ \\ expect(car) \\ expect(train) \\ 1\ \{car, train\}\ 1 \\ \\ probability(traffic\_jam, 0.7) \leftarrow hot \\ probability(\neg traffic\_jam, 0.3) \leftarrow hot \\ \\ utility(stuck\_in\_traffic, -8) \\ utility(wasting\_time, -4) \\ utility(comfort, 10) \\ utility(environment\_friendly, 3) \\ \\ hot \end{array}$$

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 is important to clarify that it wouldn't be possible to condition any kind of comparison or preference between abducibles based on the value of the computed utilities *during* model computation itself. This results from the fact that the final utilities depend on literals particular to each model, and are not available *a priori*. It should be clear that enacting preferential reasoning over the utilities computed for each model has thus to be performed after the scenarios are available, with an *a posteriori* meta-reasoning over the models and their respective utilities.

3.3 Oracles

Performing an experiment can be a critical element in the process of making informed choices. For medical practitioners, for instance, it is a natural extension of abductive reasoning. Preliminary diagnosis (or hypotheses generation) points to expected symptoms and consequences of assuming a certain diagnosis. From these expected symptoms, experiments can be extracted to confirm or disconfirm these consequences. Experiments themselves are abducible choices, and preferences are often applied to specify which experiment should be performed first given the context of a particular patient. Some of them may be more expensive, or more stressing to the patient, or less reliable under certain situations.

New information obtained from performed experiments is incorporated into the original knowledge base in order to refine the preliminary diagnosis. In addition to triming down on some of the available hypotheses, information from the experiments can actually bring about additional hypotheses for diagnosis that the practitioner was not able to conjecture prematurely. This cycle of refining a diagnosis with additional inquiry is an example of iterated abduction where the dynamics of sensing and acting upon the environment are critical to the very process of opting for the best possible set of hypotheses, and the following diagnosis example illustrates well how it relates to the non-static nature of human preferences.

Example 7. A patient shows up at the dentist with signs of pain upon teeth percussion. The expected causes for the observed signs are:

- Periapical lesion
- Horizontal Fracture of the root and/or crown
- Vertical Fracture of the root and/or crown

This setting can be modelled by the following abductive logic program D, representing a partial medical knowledge base of the practitioner:

$$
\begin{array}{l}
percussion\_pain \leftarrow periapical\_lesion \\
radiolucency \leftarrow periapical\_lesion \\
\\
percussion\_pain \leftarrow fracture \\
\\
fracture \leftarrow horizontal\_fracture \\
elliptic\_fracture\_trace \leftarrow horizontal\_fracture \\
tooth\_mobility \leftarrow horizontal\_fracture \\
\\
fracture \leftarrow vertical\_fracture \\
decompression\_pain \leftarrow vertical\_fracture \\
\\
0\ \{ \\
\quad periapical\_lesion, \\
\quad horizontal\_fracture, \\
\quad vertical\_fracture \\
\}\ 1 \\
\\
expect(periapical\_lesion) \\
expect(horizontal\_fracture) \\
expect(vertical\_fracture) \\
\\
\bot \leftarrow not\ percussion\_pain
\end{array}
$$

The integrity constraint indicates that the practitioner must conclude *percussion_pain* since that is the symptom of the patient that requires explanation. There are three preferred abductive stable models for D corresponding to each of the available hypotheses $\Delta_1 = \{periapical\_lesion\}$, $\Delta_2 = \{horizontal\_fracture\}$ and $\Delta_3 = \{vertical\_fracture\}$. Excluding the expectation domain literals, which are contained in all models, we have:

$M_1 = \{percussion\_pain, periapical\_lesion, radiolucency\}$ with hypotheses Δ_1

$M_2 = \{percussion\_pain, horizontal\_fracture, fracture, tooth\_mobility, elliptic\_fracture\_trace\}$ with hypotheses Δ_2

$M_3 = \{percussion\_pain, vertical\_fracture, fracture, decompression\_pain\}$ with hypotheses Δ_3

Notice that in the collection of abductive stable models we have not only each of the possible diagnosis, but also *all* the expected symptoms of assuming each of the diagnosis.

Following the computation of possible diagnosis scenarios, it is necessary to generate and choose an experiment which can lead to confirmation or disconfirmation of the hypotheses. Let us suppose that the medical practitioner has available an additional knowledge base of possible experiments and rules stating when each experiment is indicated. Consider the following abductive logic program E:

$$
\begin{array}{l}
1\ \{ \\
\quad xray, percussion\_test, \\
\quad mobility\_test, decompression\_test \\
\}\ 1 \\
\\
expect(percussion\_test) \leftarrow possible(percussion\_pain) \\
expect(xray) \leftarrow possible(radiolucency) \\
expect(xray) \leftarrow possible(elliptic\_fracture\_trace) \\
expect\_not(xray) \leftarrow radiotherapy\_patient
\end{array}
$$

$expect(mobility\_test) \leftarrow possible(tooth\_mobility)$
$expect(decompression\_test) \leftarrow$
$\quad possible(decompression\_pain)$

$mobility\_test \triangleleft xray$
$decompression\_test \triangleleft xray$
$mobility\_test \triangleleft decompression\_test \leftarrow trauma$

The literal $possible/1$ indicates that a given symptom is an expected possibility that should be confirmed by an experiment, if one is available. The less invasive experiments are preferred to those which are more invasive (e.g. xray). We also impose the constraint that only one experiment may be executed at any one time.

Let us now take all of the consequences which were extracted from the models of D and were not original symptoms of the patient, and assert them in program E enclosed in distinct $possible/1$ literals. The following rules are then added to E, forming the new abductive logic program E_1:

$$possible(radiolucency)$$
$$possible(elliptic\_fracture\_trace)$$
$$possible(tooth\_mobility)$$
$$possible(decompression\_pain)$$

Computing the models of E_1 yields two preferred abductive hypotheses for conducting a test on the patient: $\Delta_1 = \{mobility\_test\}$ and $\Delta_2 = \{decompression\_test\}$. Both of these hypotheses stand on equal grounds, so it is possible to non-deterministically pick one for execution. Let us assume that Δ_1 is ultimately chosen. Conducting the mobility test on the patient's tooth shows that no significant mobility is present. The new information can now be asserted onto the original diagnosis knowledge base described by D, in the form of the new integrity constraint:

$$\bot \leftarrow tooth\_mobility$$

meaning that we have disproven tooth_mobility so it would be contradictory to conclude it as a consequence of diagnosis.

By recomputing the models for the new abductive logic program D_1, we verify that the previous hypotheses set $\Delta_2 = \{horizontal\_fracture\}$ has been defeated, and now only diagnosis of $periapical\_lesion$ and $vertical\_fracture$ remain as possible candidates.

An additional iteration over the abductive logic program $E \cup Ps$, where Ps is the set of $possible/1$ literals derived from the consequences of D_1, reveals that the next experiment to be performed is the non-invasive $decompression\_test$. Depending on the result of this experiment, the practitioner will be able to conclude the final diagnosis on a last iteration of the augmented medical knowledge base. This particular example is oversimplified however, since medical knowledge is already structured so as to provide only *crucial literals*. A literal is crucial with respect to two theories if only one of the two theories supports the derivation of that literal. Computing such crucial literals is a non-trivial, but well-known problem, originally addressed in [14]. Its implementation is, nevertheless, outside the scope of this work.

An empirical implementation of such automated medical diagnostics has already been undertaken in the ACORDA system [9], a logic programming framework specifically designed to accomodate for abduction in evolving scenarios, using the perspectives previously outlined.

4. IMPLEMENTATION

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

The semantics of Stable Models has become the 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 [5, 4] (standing for XSB Answer Set Programming), is included in XSB Prolog as a practical programming interface to Smodels [11], one of the most successful and efficient implementations of the Stable Models semantics over generalized logic programs. The XASP system allows one not only to compute the models of a given NLP, but also to effectively combine 3-valued with 2-valued reasoning. The latter is achieved by using Smodels to compute the stable models of the so-called residual program, the one that results from a query evaluated in XSB using tabling [15]. This residual program is represented by delay lists, that is, the set of undefined literals for which the program could not find a complete proof, due to mutual dependencies or loops over default negation for that set of literals, detected by the XSB tabling mechanism. This method allows us to obtain any two-valued semantics in completion to the three-valued semantics the XSB system produces.

This kind of integration allows one to maintain the relevance property for queries over our programs, something that the Stable Model semantics does not originally enjoy. In Stable Models, by the very definition of the semantics, it is necessary to compute all the models for the whole program. In our implementation framework, we sidestep this issue, by using XASP to compute the query relevant residual program on demand, usually after some degree of transformation. Only the resulting program is then sent to Smodels for computation of abductive stable models.

This is one of the main problems which abduction over stable models has been facing, in that it always has to consider all the abducibles in a program and then progressively defeat all those that are irrelevant for the problem at hand. This is not so in our system framework, since we can usually begin evaluation by a top-down derivation of a query, which immediately constrains the set of abducibles that are relevant to the satisfaction and proof of that particular query.

An important consideration of computing consequences, like suggested in Section 3.1, is that we could end up having to compute the models of the whole program in order to

[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`

obtain just a particular relevant subset which will be used to enact a posteriori preferences. This can be easily avoided by performing preliminary computation of the relevant residual program, given the consequences that we expect to observe. This means that the consequences believed significant for model preference can be computed on the XSB side, and their additional residual program sent to Smodels as well. In this phase, we do not allow for additional abduction of literals, but merely enforce that rules for consequences are *consumers* of considered abducibles which have already been *produced*.

In this way, we combine a declarative methodology to describe the abductive process, with an efficient and viable implementation of reasoning by complementing a 3-valued well-founded derivation with the computation of the stable models of the residual program, in a natural way to obtain all the possible 2-valued models from the well-founded one.

5. CONCLUSIONS

We have shown that a priori preferences over abductive logic programs are an important tool for knowledge representation in modelling different kinds of choice situations where an agent needs to make a decision while considering its present and future context. Some limitations which had already been identified in [6] were addressed, namely how to express preferences between abducible sets. We have shown how previous results of extensions to the Stable Model semantics, using cardinality constraints, can be used to govern and constrain abduction of abductive literals in the context of our framework. The incorporation of these results has also led to a simpler transformation of abductive logic programs into normal logic programs extended with cardinality constraints.

The broadening of our research direction to incorporate a posteriori preferences has also been presented here as a major topic of interest for research into prospective agents, which can not only consider their immediate context, but can also present a modicum of lookahead and meta-reasoning over available scenarios, using both qualitative and quantitative dimensions for decision making. We have also shown the importance of using selected computed consequences to constrain and condition preference handling itself, and how model computation can be complemented by performing experiments with the purpose of acquiring new information with which to enact more informed choices.

Finally, we have shown the advantages of implementing our framework as a hybrid Prolog-Smodels system via the XASP package, in order to efficiently combine backwards- and forwards-chaining reasoning, respectively for constraining the abducibles to only those which are relevant for a given goal, and then to compute the consequences of assuming them in each possible scenario.

Although in this work we insisted on a strict partial order for preferences, in [6] we have already shown that this need not necessarily be so, namely by specifying different conditions for revising contradictory preferences [13]. The possible alternative revisions, required to satisfy the conditions, impart a non-monotonic or defeasible reading of the preferences given initially. Such a generalization permits us to go beyond a simply foundational view of preferences, and allows us to admit a coherent view as well, inasmuch several alternative consistent stable models may obtain for our preferences, as a result of each revision.

6. REFERENCES

[1] J. J. Alferes and L. M. Pereira. Updates plus preferences. In M. O. Aciego, I. P. de Guzmán, G. Brewka, and L. M. Pereira, editors, *Procs. of JELIA'00*, LNAI 1919, pages 345–360, Málaga, Spain, 2000. Springer.

[2] G. Brewka. Logic programming with ordered disjunction. In M. Kaufmann, editor, *Proc. 18th National Conference on Artificial Intelligence, AAAI-02*, 2002.

[3] G. Brewka, I. Niemelä, and M. Truszczynski. Answer set optimization. In *Proc. IJCAI-2003*, pages 867–872.

[4] L. Castro, T. Swift, and D. S. Warren. *XASP: Answer Set Programming with XSB and Smodels*. `http://xsb.sourceforge.net/packages/xasp.pdf`.

[5] 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.

[6] P. Dell'Acqua and L. M. Pereira. Preferential theory revision. *J. Applied Logic*, 2007. Forthcoming.

[7] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *5th Intl. Logic Programming Conf.*, pages 1070–1080. MIT Press, 1988.

[8] K. Kunen. Signed data dependencies in logic programs. *J. Log. Program.*, 7(3):231–245, 1989.

[9] G. Lopes and L. M. Pereira. Prospective logic programming with ACORDA. In G. Sutcliffe, R. Schmidt, and S. Schulz, editors, *Procs. of the FLoC'06 Ws. on Empirically Successful Computerized Reasoning, 3rd Intl. J. Conf. on Automated Reasoning*, number 192 in CEUR Workshop Procs., 2006.

[10] I. Niemelä and P. Simons. Extending the Smodels system with cardinality and weight constraints. In *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers.

[11] I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In J. Dix, U. Furbach, and A. Nerode, editors, *4th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning*, LNAI 1265, pages 420–429, Berlin, 1997. Springer.

[12] D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.

[13] P. Santana and L. M. Pereira. Emergence of cooperation through mutual preference revision. In M. Ali and D. Richard, editors, *Procs. of the 19th Intl. Conf. on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE'06)*, LNAI, Annecy, France, 2006. Springer.

[14] A. Seki and A. Takeuchi. An algorithm for finding a query which discriminates competing hypotheses. Technical Report TR 143, Institute for New Generation Computer Technology, Japan, 1985.

[15] T. Swift. Tabling for non-monotonic programming. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):201–240, 1999.

[16] A. van Gelder, K. Ross, and J. Schlipf. Unfounded sets and well-founded semantics for general logic programs. *JACM*, 38(3):620–650, 1991.