Chapter 10

INSPECTING AND PREFERRING ABDUCTIVE MODELS

Luís Moniz Pereira*, Pierangelo Dell'Acqua†, Alexandre Miguel Pinto‡ and Gonçalo Lopes§

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

†ITN — Department of Science and Technology
Linköping University, 60174 Norrköping, Sweden
pier@itn.liu.se

‡CISU — Centro de Informática e Sistemas
Universidade de Coimbra, Pólo II, Pinhal de Marrocos
3030-290 Coimbra, Portugal
ampinto@dei.uc.pt

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

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.

10.1. 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 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,[4] 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.

10.2. Abductive Framework

10.2.1. *Basic Abductive Language*

[5]Let $\mathcal{L}$ be a first order propositional language defined as follows. 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 $not\,A$, the latter expressing that the atom is false by default (CWA).

Definition 10.1. 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 10.2. A goal or query in $\mathcal{L}$ takes 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.

Notation 10.1. 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 10.1. Let $P = \{p \leftarrow abduced(a), a; q \leftarrow abduced(b)\}$ with set of abducibles $\mathcal{A}_P = \{a, b, abduced(a), abduced(b)\}$. Then, P has four intended models: $M = \{\}$, $M_2 = \{p, a, abduced(a)\}$, $M_3 = \{q, b, abduced(b)\}$, and $M_4 = \{p, q, a, b, abduced(a), abduced(b)\}$. The set $\{q, abduced(b)\}$ is not an intended model since the assumption of $abduced(b)$ 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 $abduced(.)$, that is:

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

10.2.1.1. *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,[6] and triggering one of the preference rules will break the cycle in favor 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 rules, for any given abducible $a \in \mathcal{A}^*$:

$$expect(a) \leftarrow L_1, \ldots, L_t$$
$$expect\_not(a) \leftarrow L_1, \ldots, L_t$$

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

Example 10.2. Let $P = \{p \leftarrow a; q \leftarrow b; expect(a); expect(b); expect\_not(a) \leftarrow q\}$ with set of abducibles $\mathcal{A}_P = \{a, b, abduced(a), abduced(b)\}$. Then, P has three intended models: $M = \{expect(a), expect(b)\}$, $M_2 = \{p, a, abduced(a), expect(a), expect(b)\}$ and $M_3 = \{q, b, abduced(b), expect(a), expect(b)\}$. It is not possible to assume both a and b because the assumption of b makes q true which in turn makes $expect\_not(a)$ true preventing a to be assumed.

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.

10.2.1.2. *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 $abduced(.)$ since they belong to $\mathcal{A}^*$. Removing this requirement would not add to the expressive power of $\mathcal{L}$.

Example 10.3. Let $P = \{p \leftarrow a; a \prec b; b \prec a; expect(a); expect(b)\}$ with set of abducibles $\mathcal{A}_P = \{a, b, abduced(a), abduced(b)\}$. Then, P has two intended models: $M = \{a \prec b, b \prec a, expect(a), expect(b)\}$ and $M_2 = \{a \prec b, b \prec a, a, b, p, expect(a), expect(b), abduced(a), abduced(b)\}$. This is due to the active abduction behavior of $a \prec b$ and $b \prec a$ that prevents intended models containing either a or b.

10.2.1.3. *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 $abduced(.)$. Removing this requirement would not add to the expressive power of $\mathcal{L}$.

Example 10.4. Let $P = \{p \leftarrow a; q \leftarrow b; a \sqsubset b; expect(a)\}$ with set of abducibles $\mathcal{A}_P = \{a, b, abduced(a), abduced(b)\}$. Then, there exists only one intended model of P: $M = \{a \sqsubset b, expect(a)\}$. Note that $M_2 = \{a \sqsubset b, b, q, expect(a), abduced(b)\}$ and $M_3 = \{a \sqsubset b, a, p, expect(a), abduced(a)\}$ 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.

Example 10.5. Let $P = \{p \leftarrow a; q \leftarrow b; a \sqsubset b; expect(a); expect(b)\}$ with abducibles $\mathcal{A}_P = \{a, b, abduced(a), abduced(b)\}$. Then, P has three intended models: $M = \{a \sqsubset b, expect(a), expect(b)\}$, $M_2 = \{a \sqsubset b, b, q, expect(a), expect(b), abduced(b)\}$ and $M_3 = \{a \sqsubset b, a, b, p, q, expect(a), expect(b), abduced(a), abduced(b)\}$.

Mark that the global $a \sqsubset b \leftarrow cond$ can be avoided altogether by replacing every occurrence of the abducible a with $cond, a, abduced(b)$. In fact, when one desires some occurrences of a to be subjected to conditional abduction and others not, then only some occurrences need be replaced as above.

10.2.1.4. *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 $abduced(.)$.

10.2.2. *Declarative Semantics*

The declarative semantics of programs over $\mathcal{L}$ is given in terms of abductive stable models. Before introducing them, we need few definitions. In the following we 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 $not\ A$ 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. We also need to introduce the notion of default assumptions of P with respect to an interpretation M, where a default literal $not\ A$ is considered an atom, and $not\ not\ A \equiv A$.

$Default(P, M)$

$= \{not\ A : \text{there exists no rule } A \leftarrow L_1, \ldots, L_t \text{ in } P \text{ such that } M \vDash L_1, \ldots, L_t\}$

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) $M \nvDash \bot$
(2) $M = least(P \cup Default(P, M))$, where $least$ 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 Δ, we write Δ^* to indicate:

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

Definition 10.3. Let $\Delta \subseteq \mathcal{A}_P$ be a set of abducibles. M is an abductive stable model with hypotheses Δ 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 10.6. 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. 10.3 is not fulfilled.

Definition 10.4. Let G be a goal. Then, Δ is an abductive explanation for G in P iff:

(1) M is an abductive stable model with hypotheses Δ of P, and
(2) $M \vDash G$

Definition 10.5. Let G be a goal. Then, Δ is a strict abductive explanation for G in P iff

(1) Δ 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. 10.5 condition (2) is not subject to minimization. The reason for this is clarified by the next example.

Example 10.7. Reconsider the program P of Example 10.6. 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 Δ is not a minimal set satisfying condition (1)

of Def. 10.5. Indeed, the minimal set is $\Delta_2 = \{abduced(a)\}$. Hence, there exists no strict abductive explanation for G in P.

The following result relates abductive explanations and strict abductive explanations.

Proposition 10.1. *Let G be a goal and Δ a strict abductive explanation for G in P. Then, Δ is an abductive explanation for G in P.*

Proof. It follows immediately from Definitions 10.4 and 10.5. □

It is easy to see that the converse of Prop 10.1 does not hold since abductive explanations are not subject to the minimality requirement.

10.3. Pragmatics

10.3.1. *Constraining Abduction*

Quite often in domain problems it happens that the assumption of abducibles is subject to the fulfillment of certain conditions, including other assumptions, which must be satisfied. This requirement can be expressed in our framework by exploiting constrained abduction, $a \sqsubset b$. It states that the assumption of the abducible a is subject to the assumption of b with the property that b cannot be assumed in order to assume a.

Example 10.8. 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. In case it is late night, one can assume that the pub is open if there are people inside. The pub being located in an entertainment district, there is noise around if there is people in the pub or a party nearby. This scenario can be described by the following program P with $\mathcal{A}_P = \{open, cleaning, party, people, abduced(open), abduced(cleaning), abduced(party), abduced(people)\}$.

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

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: $\{open\}$ or $\{cleaning\}$. Otherwise (it is late night), then there is only one explanation for the lights being turned on: $\{cleaning\}$. If instead it is late night and one hears noise also (that is, the query is $?-light, noise$), then one will now have three abductive explanations: $\{open, people\}$, $\{cleaning, party\}$ and $\{open, party, people\}$. The last explanation reflects the fact that the pub may be open with late customers simultaneously with a party nearby, both events producing noise.

10.3.2. *Preferring Abducibles*

In this section we illustrate how to express preferences between considered abducibles. To do so, we employ the construct $L\,\langle l_1,\ldots,l_n\rangle\,U$ to constrain the number of abducibles assumed (without having to assume it). Such a construct has a passive abduction behavior and is defined as:

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

for any abducible $l_1,\ldots,l_n$ in $\mathcal{A}^*$. The following two examples illustrate the difference between $L\,\langle l_1,\ldots,l_n\rangle\,U$ and $L\,\{l_1,\ldots,l_n\}\,U$.

Example 10.9. Let $P = \{p \leftarrow abduced(a); 1\,\{a,b\}\,1; expect(a); expect(b)\}$ with set of abducibles $\mathcal{A}_P = \{a, b, abduced(a), abduced(b)\}$. Then, P has two abductive stable models: $M = \{p, a, 1\,\{a,b\}\,1, abduced(a), expect(a), expect(b)\}$ and $M_2 = \{b, 1\,\{a,b\}\,1, abduced(b), expect(a), expect(b)\}$. It holds that $\Delta = \{a, abduced(a)\}$ is a strict abductive explanation for $?{-}p$ in P.

Example 10.10. Let $Q = \{p \leftarrow abduced(a); 1\,\langle a,b\rangle\,1; expect(a); expect(b)\}$ with set of abducibles $\mathcal{A}_Q = \{a, b, abduced(a), abduced(b)\}$. Then, Q has two abductive stable models: $M = \{p, a, 1\,\langle a,b\rangle\,1, abduced(a), expect(a), expect(b)\}$ and $M_2 = \{b, 1\,\langle a,b\rangle\,1, abduced(b), expect(a), expect(b)\}$. In contrast to the program P of Example 10.9, there exists no strict abductive explanation for $?{-}p$ in Q.

Consider for example a situation where there are three alternative abducibles a, b and c can explain an observation p. Suppose that a and c are preferred to b. This situation can be axiomatized by the following program P with abducibles $\mathcal{A}_P = \{a, b, c, abduced(a), abduced(b), abduced(c)\}$:[a]

$$P = \{p \leftarrow a; p \leftarrow b; p \leftarrow c; a \prec b; c \prec b; 0\,\langle a,b,c\rangle\,1; expect(a); expect(b); expect(c)\}$$

P has three abductive stable models:

$$M = \{a \prec b, c \prec b, 0\,\langle a,b,c\rangle\,1, expect(a), expect(b), expect(c)\}$$
$$\Delta = \{\}$$

$$M_2 = \{a \prec b, c \prec b, 0\,\langle a,b,c\rangle\,1, a, p, expect(a), expect(b), expect(c), abduced(a)\}$$
$$\Delta_2 = \{a, abduced(a)\}$$

$$M_3 = \{a \prec b, c \prec b, 0\,\langle a,b,c\rangle\,1, c, p, expect(a), expect(b), expect(c), abduced(c)\}$$
$$\Delta_3 = \{c, abduced(c)\}$$

The strict abductive explanations for $?{-}p$ in P are Δ_2 and Δ_3. It is worth noting that the abducible b cannot be part of any abductive explanation. In fact, if it were the case, the rules $a \prec b$ and $c \prec b$ in P would enforce the abduction of a and c (respectively) violating therefore the cardinality constraint $0\,\langle a,b,c\rangle\,1$.

[a]Note that if the preference relation is a strict partial order, then it must be axiomatized explicitly.

Example 10.11. Consider a situation[b] 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 can be represented by a program P over $\mathcal{L}$ with set of abducibles $\mathcal{A}_P = \{tea, coffee, abduced(tea), abduced(coffee)\}$:

$$drink \leftarrow tea$$
$$drink \leftarrow coffee$$
$$expect(tea)$$
$$expect(coffee)$$
$$expect\_not(coffee) \leftarrow blood\_pressure\_high$$
$$0\,\langle tea, coffee\rangle\,1$$
$$coffee \prec tea \leftarrow sleepy$$

Following the abductive stable model semantics, the program above has two models, one with tea and the other with coffee. Also, by adding the literal *sleepy*, the enforced abduction comes into play, defeating the abductive stable model where only *tea* is present (due to the impossibility of simultaneously abducing coffee). 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*.

10.3.3. *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 power-set 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,[8] and below we present how one can nicely import these results to work with abduction of sets, and also hierarchies of sets.

Example 10.12. 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

[b]The following example is taken from.[7]

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 P over $\mathcal{L}$ with set of abducibles $\mathcal{A}_P = \{bread, salad, cheese, fish, meat, veggie, wine, juice, water, abduced(bread), abduced(salad), abduced(cheese), abduced(fish), abduced(meat), abduced(veggie), abduced(wine), abduced(juice), abduced(water)\}$:

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

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.

10.3.4. *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, but without performing additional abductions, and without having to produce a complete model to do so. Pereira and Pinto[9] argue that this type of reasoning requires a new mechanism. To achieve it, they introduce the concept of inspection point, and show how one can employ it to investigate side-effects of interest. Procedurally, inspection points can be construed as utilizing a form of meta-abduction, by "meta-abducing" the specific abduction of actually checking (i.e. passively verify) that a certain and 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 general, one may want to find one possible set of conditions (literals of the program assumed true) sufficient to entail the query. However, sometimes one may also want to know which are (some of) the consequences (side-effects) of such conditions. That is, one wants to know the truth value of some other literals, not part of the query, whose truth-value may be determined by the abductive conditions found as a solution of the query. In some cases, the focus can be in some specific side-effects of abductions performed. In their approach the side-effects of interest are explicitly indicated by the user by wrapping the corresponding goals within a reserved construct *inspect*/1. Procedurally, *inspect* goals must be solved without abducing regular abducibles, only "meta-abducibles" of the form *abduced*/1. An example will make this concept clear.

Example 10.13. Consider the following program taken from,[9] where *tear_gas*, *fire*, and *water_cannon* are abducibles.

$$\bot \leftarrow police, riot, not\, contain$$

$contain \leftarrow tear\_gas$	$contain \leftarrow water\_cannon$
$smoke \leftarrow fire$	$smoke \leftarrow inspect(tear\_gas)$
$police$	$riot$

Notice the two rules for *smoke*. The first states that 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*. Thus, *tear_gas* is an inspection point.

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. *smoke* is plausibly 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 which lends credibility to hypotheses.

Example 10.14. A gourmet's proper meal goes along with red wine if the main dish is meat, and with white wine if the main dish is fish. Specific wine abduction can be done before main dish abduction, without committing to the kind of main dish. If incompatible, an alternative abduction for the wine can be produced. Consider the following program, with abducibles $\mathcal{A}_P = \{meat, fish, merlot, chardonnay\}$:

$\bot \leftarrow not\, proper\_meal$	$main\_dish \leftarrow meat$
$proper\_meal \leftarrow wine, main\_dish$	$main\_dish \leftarrow fish$
$wine \leftarrow red\_wine, inspect(meat)$	$red\_wine \leftarrow merlot$
$wine \leftarrow white\_wine, inspect(fish)$	$white\_wine \leftarrow chardonnay$

$0\ \langle meat, fish\rangle\ 1$
$0\ \langle merlot, chardonnay\rangle\ 1$

The alternative combinations for the gourmet's proper meal are thus: {*fish*, *chardonnay*} and {*meat*, *merlot*}. The main dish is not determined by the wine, even

if the latter is pre-selected first. This example also illustrates the use of inspection points to express a preference whereby one can only have wine with a main dish.

Inspection points can occur within the proof tree of another inspection point, as the next example shows.

Example 10.15. Consider the following program, where the abducibles are a, b, c, d:

$$\begin{array}{ll} x \leftarrow a, \mathit{inspect}(y), b, c, \mathit{not}\, d & y \leftarrow \mathit{inspect}(\mathit{not}\, a) \\ z \leftarrow d & y \leftarrow b, \mathit{inspect}(\mathit{not}\, z), c \end{array}$$

Let the query be $?{-}x$. According to the intended semantics of inspection points, the set $\Delta = \{a, b, c\}$ is an abductive explanation for the query.

The following simple transformation maps programs with inspection points into programs over $\mathcal{L}$. Mark that the abductive stable models of the transformed program clearly correspond to the intended procedural meanings ascribed to the inspection points of the original program.

Definition 10.6. (Transforming Inspection Points) Let P a program containing rules whose body possibly contains inspection points. The program $\Pi(P)$ over $\mathcal{L}$ consists of:

(1) all the rules obtained by the rules in P by replacing:

$$\mathit{inspect}(a) \text{ or } \mathit{inspect}(\mathit{abduced}(a)) \text{ with } \mathit{abduced}(a)$$

if a is an abducible, and keeping $\mathit{inspect}(L)$ otherwise

(2) $\mathit{inspect}(\mathit{not}\, L)$ with $\mathit{not}\, \mathit{inspect}(L)$

(3) for every abducible a, the rule: $\mathit{expect}(a)$

(4) for every rule $A \leftarrow L_1, \ldots, L_t$ in P, the additional rule: $\mathit{inspect}(A) \leftarrow L_1^{'}, \ldots, L_t^{'}$ where for every $1 \leq i \leq t$

$$L_i^{'} = \begin{cases} \mathit{abduced}(L_i) & \text{if } L_i \text{ is an abducible} \\ \mathit{inspect}(X) & \text{if } L_i \text{ is } \mathit{inspect}(X) \\ \mathit{inspect}(L_i) & \text{otherwise} \end{cases}$$

Example 10.16. Let P be the program of Example 10.15. Then, $\Pi(P)$ is:

$x \leftarrow a, \mathit{inspect}(y), b, c, \mathit{not}\, d$
$\mathit{inspect}(x) \leftarrow \mathit{abduced}(a), \mathit{inspect}(y), \mathit{abduced}(b), \mathit{abduced}(c), \mathit{abduced}(\mathit{not}\, d)$

$y \leftarrow \mathit{abduced}(\mathit{not}\, a)$
$\mathit{inspect}(y) \leftarrow \mathit{abduced}(\mathit{not}\, a)$

$y \leftarrow b, \mathit{inspect}(\mathit{not}\, z), c$
$\mathit{inspect}(y) \leftarrow \mathit{abduced}(b), \mathit{inspect}(\mathit{not}\, z), \mathit{abduced}(c)$

$z \leftarrow d$
$inspect(z) \leftarrow abduced(d)$

$expect(a).$
$expect(b).$
$expect(c).$
$expect(d).$

The abductive stable model of $\Pi(P)$ respecting the inspection points is:

$$\{x, a, b, c, abduced(a), abduced(b), abduced(c), expect(a), expect(b), expect(c), expect(d)\}.$$

Note that for each $abduced(a)$ the corresponding A is in the model.

10.4. Procedural Semantics

10.4.1. *Framework*

In this section, we consider a first-order propositional language $\mathcal{L}^{\#}$ containing rules as defined in Def. 10.1. Programs over $\mathcal{L}^{\#}$ are constrained to satisfy certain given properties.

Definition 10.7. Let Γ and Σ be sets of rules over $\mathcal{L}^{\#}$. Then (Γ, Σ) is a restricted program.

The basic idea is that Γ contains the rules formalizing the application domain while Σ formalizes the properties that Γ must satisfy. Every restricted program is associated with a set of abducibles $\mathcal{A}_{(\Gamma,\Sigma)}$.

Remark 10.1. Note that the definition of restricted program can be generalized to have Σ be any set of wffs.

In the following let $\Delta \subseteq \mathcal{A}_{(\Gamma,\Sigma)}$ be a set of abducibles.

Definition 10.8. M is a valid stable model with hypotheses Δ of (Γ, Σ) iff:

(1) $M \nvDash \bot$
(2) $M = least(\Gamma^+ \cup Default(\Gamma^+, M))$, where $\Gamma^+ = \Gamma \cup \Delta$
(3) $M \vDash \Sigma$

A valid stable model is an abductive stable model (conditions (1) and (2)) satisfying the wffs in Σ (condition (3)). (explain why not to have strict valid stable models)

Example 10.17. Let (Γ, Σ) be a restricted program where: $\Gamma = \{p \leftarrow a; expect(a)\}$ and $\Sigma = \{q \leftarrow a\}$ with $\mathcal{A}_{(\Gamma,\Sigma)} = \{a, abduced(a)\}$. Then, its unique valid stable model is $M = \{expect(a)\}$. In fact, the only other model could be $M_2 = \{p, a, abduced(a), expect(a), expect(b)\}$ but it is not valid being Σ not satisfied.

Example 10.18. Let (Γ, Σ) be $(\{p \leftarrow a; \mathit{expect}(a); \mathit{expect}(b)\}, \{\bot \leftarrow a, \mathit{not}\, b\})$ with $\mathcal{A}_{(\Gamma,\Sigma)} = \{a, b, \mathit{abduced}(a), \mathit{abduced}(b)\}$. This program has two valid stable models: $M = \{\mathit{expect}(a), \mathit{expect}(b)\}$ and $M_2 = \{p, a, b, \mathit{abduced}(a), \mathit{abduced}(b), \mathit{expect}(a), \mathit{expect}(b)\}$.

Definition 10.9. Let G be a goal. Then, Δ is a valid explanation for G in (Γ, Σ) iff:

(1) M is a valid stable model with hypotheses Δ of (Γ, Σ), and
(2) $M \models G$

Definition 10.10. Let G be a goal. Then, Δ is a strict valid explanation for G in (Γ, Σ) iff

(1) Δ is a minimal set for which:
- $M \not\models \bot$
- $M = \mathit{least}(\Gamma^+ \cup \mathit{Default}(\Gamma^+, M))$, where $\Gamma^+ = \Gamma \cup \Delta$
- $M \models G$

(2) $M \models \Sigma$

Proposition 10.2. *Let G be a goal and Δ a strict valid explanation for G in (Γ, Σ). Then, Δ is an valid explanation for G in (Γ, Σ).*

Proof. It follows immediately from Definitions 10.9 and 10.10. □

It is easy to see that the converse of Prop 10.2 does not hold. Consider the following example.

Example 10.19. Let $\Gamma = \{p \leftarrow a\}$ and $\Sigma = \{\bot \leftarrow \mathit{not}\, b\}$, where $\mathcal{A}_{(\Gamma,\Sigma)} = \{a, b\}$. Then, $M = \{p, a, b\}$ is a valid stable model of (Γ, Σ). Suppose the goal G is $?{-}p$. Then, $\Delta = \{a, b\}$ is a valid explanation for G in P. However, there exists no strict valid explanation for G.

10.4.2. *Program Transformation*

We define a transformation γ mapping programs over $\mathcal{L}$ into restricted programs over $\mathcal{L}^{\#}$.

Definition 10.11. 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.
Γ consists of:

(1) all the rules in P
(2) $\bot \leftarrow a, \mathit{not}\ \mathit{expect}(a)$
$\bot \leftarrow a, \mathit{expect\_not}(a)$
for every abducible $a \in \mathcal{A}_P^*$

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

Σ consists of:

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

Remark 10.2. We assume given the atom $\mathit{count}([l_1, \ldots, l_n], m)$ 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 $M \vDash \mathit{count}([l_1, \ldots, l_n], m)$ iff $W(C, M) = m$, for any interpretation M.

Remark 10.3. If P contains inspection points, then apply the transformation γ to $\Pi(P)$ (cf. Definition 10.6) instead of P directly.

Example 10.20. Consider again the program P of Example 10.11. The transformed program $\gamma(P) = (\Gamma, \Sigma)$ over $\mathcal{L}^{\#}$ with abducibles $\mathcal{A}_{(\Gamma,\Sigma)} = \{\mathit{tea}, \mathit{coffee}, \mathit{abduced}(\mathit{tea}), \mathit{abduced}(\mathit{coffee})\}$ is:
Γ consists of:

$\mathit{drink} \leftarrow \mathit{tea}$
$\mathit{drink} \leftarrow \mathit{coffee}$

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

$0\,\{\mathit{abduce}(\mathit{tea}), \mathit{abduce}(\mathit{coffee})\}\,1$
$\mathit{coffee} \prec \mathit{tea} \leftarrow \mathit{sleepy}$

$\bot \leftarrow \mathit{tea}, \mathit{not}\ \mathit{expect}(\mathit{tea})$
$\bot \leftarrow \mathit{coffee}, \mathit{not}\ \mathit{expect}(\mathit{coffee})$
$\bot \leftarrow \mathit{tea}, \mathit{not}\ \mathit{expect\_not}(\mathit{tea})$
$\bot \leftarrow \mathit{coffee}, \mathit{not}\ \mathit{expect\_not}(\mathit{coffee})$

$\bot \leftarrow \mathit{tea}, \mathit{not}\ \mathit{abduced}(\mathit{tea})$
$\bot \leftarrow \mathit{coffee}, \mathit{not}\ \mathit{abduced}(\mathit{coffee})$

$\bot \leftarrow \mathit{coffee} \prec \mathit{tea} \leftarrow \mathit{expect}(\mathit{coffee}), \mathit{not}\ \mathit{expect\_not}(\mathit{coffee}), \mathit{tea}, \mathit{not}\ \mathit{coffee}$

$\bot \leftarrow 0\,\{\mathit{abduce}(\mathit{tea}), \mathit{abduce}(\mathit{coffee})\}\,1, \mathit{count}([\mathit{abduce}(\mathit{tea}), \mathit{abduce}(\mathit{coffee})], N), N \leq 0$

$\bot \leftarrow 0\,\{abduce(tea), abduce(coffee)\}\,1, count([abduce(tea), abduce(coffee)], N), N \geq 1$

Σ consists of:

$\bot \leftarrow abduced(tea), not\ tea$
$\bot \leftarrow abduced(coffee), not\ coffee$

10.4.3. *Properties*

The correctness of the transformation γ is guaranteed by the following two properties.

Theorem 10.1. *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 Δ of $\gamma(P)$.*

Proof.

($\Rightarrow$)
Let M be an abductive stable model with hypotheses $\Delta \subseteq \mathcal{A}_P$ of P. Then, we need to show that M is a valid stable model with hypotheses Δ of $\gamma(P) = (\Gamma, \Sigma)$. This is established by proving the three conditions below.

(1) $M \nvDash \bot$
Immediate by condition (1) of Definition 10.3.

(2) $M = least(\Gamma^+ \cup Default(\Gamma^+, M))$, where $\Gamma^+ = \Gamma \cup \Delta$
Note first that by Definition 10.11 it holds that $\Delta \subseteq \mathcal{A}_P = \mathcal{A}_{(\Gamma,\Sigma)}$. By condition (2) of Definition 10.3 we have that:

$$M = least(Q \cup Default(Q, M)), \text{ where } Q = P \cup \Delta.$$

Let Q_2 be the program obtained by extending Q with the rules:

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

for every abducible $a \in \mathcal{A}_P^*$. Then, M is an abductive stable model of Q_2 since if $a \in \Delta$ then the claim follows by condition (3) of Definition 10.3, otherwise ($a \notin \Delta$) the claim is immediate by noting that the body of the rule is false.

Let Q_3 be the program obtained by extending Q_2 with the rules:

$$\bot \leftarrow a, not\ abduced(a)$$

for every abducible $a \in \mathcal{A}_P^*$. By condition (4) of Definition 10.3, it follows immediately that M is an abductive stable model of Q_3.

Let Q_4 be the program obtained by extending Q_3 with the rules:

$$\bot \leftarrow a \prec b, expect(a), not\ expect\_not(a), b, not\ a$$

for every rule $a \prec b \leftarrow L_1, \ldots, L_t$ in P. To establish that M is an abductive stable model of Q_4 consider the following two cases. If it holds that $M \not\models (a \prec b, \mathit{expect}(a), \mathit{not\ expect\_not}(a), b)$ then the claim is immediate. Otherwise, the claim follows by condition (5) of Definition 10.3 stating that $M \models a$.

Let Q_5 be the program obtained by extending Q_4 with the rules:

$$\bot \leftarrow L\,\{l_1, \ldots, l_n\}\,U, \mathit{count}([l_1, \ldots, l_n], N), N \leq L$$
$$\bot \leftarrow L\,\{l_1, \ldots, l_n\}\,U, \mathit{count}([l_1, \ldots, l_n], N), N \geq U$$

for every rule $L\,\{l_1, \ldots, l_n\}\,U \leftarrow L_1, \ldots L_t$ in P. M is an abductive stable model of Q_5 by condition (6) of Definition 10.3 under the assumption that $M \models \mathit{count}([l_1, \ldots, l_n], m)$ iff $W(C, M) = m$ with $C = L\,\{l_1, \ldots, l_n\}\,U$.

By noting that $Q_6 = \Gamma^+$, claim (2) follows.

(3) $M \models \Sigma$

Note first that by Definition 10.11 the program Σ consists of two kinds of rules:

$$\bot \leftarrow \mathit{abduced}(a), \mathit{not}\ a$$

for every abducible $a \in \mathcal{A}_P^*$, and

$$\bot \leftarrow a \sqsubset b, a, \mathit{not}\ b$$

for every rule $a \sqsubset b \leftarrow L_1, \ldots, L_t$ in P. Conditions (7) and (8) of Definition 10.3 ensure that the bodies of the rules above is false. Hence, the claim follows.

($\Leftarrow$)

Let M be a valid stable model with hypotheses Δ of $\gamma(P) = (\Gamma, \Sigma)$. We need to show that M is an abductive stable model with hypotheses $\Delta \subseteq \mathcal{A}_P$ of P. The proof of this claim is omitted being quite similar to the proof of the previous case ($\Rightarrow$). □

Lemma 10.1. *Let P be a program over $\mathcal{L}$ and G a goal. Let $\Delta_2 \subseteq \mathcal{A}_P$ and M_2 be a set of abducibles and an interpretation that satisfy condition (1) of Definition 10.10 for G in $\gamma(P)$. Then, Δ_2 and M_2 satisfy condition (1) of Definition 10.5 for G in P.*

Proof. To prove that Δ_2 and M_2 satisfy condition (1) of Definition 10.5 we have to prove the following conditions:

- $M_2 \models G$ and $M_2 \not\models \bot$ since M_2 satisfies condition (1) of Definition 10.10
- $M_2 = \mathit{least}(Q \cup \mathit{Default}(Q, M_2))$, where $Q = P \cup \Delta_2$ by construction of γ, Γ exactly contains all the rules in P plus rules of the form $\bot \leftarrow \mathit{body}$. This claim follows from the facts that:
  - $\mathit{Default}(Q, M_2) = \mathit{Default}(\Gamma^+, M_2)$, and
  - $M_2 = \mathit{least}(\Gamma^+ \cup \mathit{Default}(\Gamma^+, M_2))$

  where $\Gamma^+ = \Gamma \cup \Delta_2$

- $M \models \mathit{expect}(a)$ and $M \not\models \mathit{expect\_not}(a)$, for every $a \in \Delta_2^*$ immediate from case (2) of Definition 10.11 and $M_2 \not\models \bot$
- for every $a \in \Delta_2^*$, if $M_2 \models a$ then $M_2 \models \mathit{abduced}(a)$ immediate from case (3) of Definition 10.11 and $M_2 \not\models \bot$
- for every atom $a \prec b$, if $M_2 \models a \prec b$, $M_2 \models \mathit{expect}(a)$, $M_2 \not\models \mathit{expect\_not}(a)$ and $M_2 \models b$, then $M_2 \models a$ immediate from case (4) of Definition 10.11 and $M_2 \not\models \bot$
- for every atom $C = L\,\{l_1, \ldots, l_n\}\,U$, if $M_2 \models C$ then $L \leq W(C, M_2) \leq U$ immediate from case (5) of Definition 10.11 and $M_2 \not\models \bot$

□

Theorem 10.2. *Let P be a program over $\mathcal{L}$ and G a goal. $\Delta \subseteq \mathcal{A}_P$ is a strict abductive explanation for G in P iff Δ is a strict valid explanation for G in $\gamma(P)$.*

Proof.
Let $\gamma(P) = (\Sigma, \Gamma)$.
($\Rightarrow$)
Let $\Delta \subseteq \mathcal{A}_P$ be a strict abductive explanation for G in P. Then, we need to show that Δ is a strict valid explanation for G in $\gamma(P)$.

The proof is by contradiction. Suppose that Δ is not a strict valid explanation for G in $\gamma(P)$. That is, there is no interpretation M for which conditions (1) and (2) of Definition 10.10 hold. Let $\overline{M}$ be an abductive stable model with hypotheses Δ of P. By Theorem 10.1, $\overline{M}$ is a valid stable model with hypotheses Δ of (Γ, Σ), and therefore Δ is a valid explanation for G in (Γ, Σ).

Since Δ is not a strict valid explanation, the two conditions (1) and (2) of Definition 10.10 do not hold for $\overline{M}$. Consider condition (2). Since $\overline{M}$ is a valid stable model of (Γ, Σ), then we have that $\overline{M} \models \Sigma$. Thus, it is condition (1) that makes Δ not strict. Hence, there must exist a set of abducibles $\Delta_2 \subset \Delta$ and an interpretation M_2 that satisfy condition (1) of Definition 10.10:[c]

(1) Δ_2 is a minimal set for which:

- $M_2 \not\models \bot$
- $M_2 = \mathit{least}(\Gamma^+ \cup \mathit{Default}(\Gamma^+, M_2))$, where $\Gamma^+ = \Gamma \cup \Delta_2$
- $M_2 \models G$

The hypotheses Δ_2 satisfy condition (1) of Definition 10.5 by Lemma 10.1, and therefore Δ cannot be a strict abductive explanation for G in P (because Δ would not be a minimal set to satisfy condition (1)).

($\Leftarrow$)
Let $\Delta \subseteq \mathcal{A}_P$ be a strict valid explanation for G in $\gamma(P)$. We need to show that Δ is a strict abductive explanation for G in P.

The proof of this claim is omitted being quite similar to the proof of the previous case. □

[c]Note that M_2 might not satisfy Σ.

10.5. 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.

10.5.1. *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 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.

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 10.21. 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 $b \prec a \leftarrow c$. 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.

10.5.2. *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.[10] A particularly interesting field of study regards modeling 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 10.22. 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 modeled by the following abductive logic program:

$$
\begin{array}{l}
go\_to(beach) \leftarrow car \\
go\_to(beach) \leftarrow train \\
\\
expect(car) \\
expect(train) \\
1\ \{car, train\}\ 1 \\
\\
probability(traffic\_jam, 0.7) \leftarrow hot \\
probability(not\ traffic\_jam, 0.3) \leftarrow hot \\
\\
utility(stuck\_in\_traffic, -8) \\
utility(wasting\_time, -4) \\
utility(comfort, 10) \\
utility(environment\_friendly, 3) \\
\\
hot
\end{array}
$$

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.

10.5.3. *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 trimming 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 10.23. 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:

$$
\begin{array}{l}
percussion\_pain \leftarrow periapical\_lesion \\
percussion\_pain \leftarrow fracture \\
\\
radiolucency \leftarrow periapical\_lesion \\
\\
fracture \leftarrow horizontal\_fracture \\
elliptic\_fracture\_trace \leftarrow horizontal\_fracture \\
tooth\_mobility \leftarrow horizontal\_fracture
\end{array}
$$

$$fracture \leftarrow vertical\_fracture$$
$$decompression\_pain \leftarrow vertical\_fracture$$

$$0\ \{periapical\_lesion, horizontal\_fracture, vertical\_fracture\}\ 1$$

$$expect(periapical\_lesion)$$
$$expect(horizontal\_fracture)$$
$$expect(vertical\_fracture)$$

$$\bot \leftarrow not\ percussion\_pain$$

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:

$$1\ \{$$
$$\quad xray, percussion\_test,$$
$$\quad mobility\_test, decompression\_test$$
$$\}\ 1$$

$$expect(percussion\_test) \leftarrow possible(percussion\_pain)$$
$$expect(xray) \leftarrow possible(radiolucency)$$
$$expect(xray) \leftarrow possible(elliptic\_fracture\_trace)$$
$$expect\_not(xray) \leftarrow radiotherapy\_patient$$

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

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

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.[11] Its implementation is, nevertheless, outside the scope of this work.

10.6. Sophie's Choice

One of the most discussed cases where the same moral precept gives rise to conflicting obligations is taken from William Styron's Sophie's Choice.[12] Sophie and her two children are at a Nazi concentration camp. A guard confronts Sophie and tells her that one of her children will be allowed to live and one will be killed. But it is Sophie who must decide which child will be killed. Sophie can prevent the death of either of her children, but only by condemning the other to be killed. The guard makes the situation even more excruciating by informing Sophie that if she chooses neither, then both will be killed. With this added factor, Sophie has a morally compelling reason to choose one of her children. But for each child, Sophie has an apparently equally strong reason to save him or her. Thus the same moral precept gives rise to conflicting obligations.

The described scenario can be formalized with the following program P over $\mathcal{L}$ with set of abducibles:

$$\mathcal{A}_P = \{letting\_both\_die, kill(child\_1), kill(child\_2), flip\_a\_coin\}$$

(1) $expect(kill(child\_1))$
$expect(kill(child\_2))$
$expect(flip\_a\_coin)$
$expect(letting\_both\_die)$
(2) $on\_observe(decide) \leftarrow sophie\_choice$
(3) $decide \leftarrow letting\_both\_die$
$decide \leftarrow kill(child\_1)$
$decide \leftarrow kill(child\_2)$
$decide \leftarrow flip\_a\_coin$
(4) $1\ \langle letting\_both\_die, kill(child\_1), kill(child\_2), flip\_a\_coin\rangle\ 1$
(5) $expect\_not(kill(C)) \leftarrow special\_reason(C)$
(6) $expect\_not(flip\_a\_coin) \leftarrow special\_reason(child\_1), not\ special\_reason(child\_2)$
$expect\_not(flip\_a\_coin) \leftarrow special\_reason(child\_2), not\ special\_reason(child\_1)$
(7) $die(2) \leftarrow abduced(letting\_both\_die)$
$die(1) \leftarrow abduced(kill(child\_1))$
$die(1) \leftarrow abduced(kill(child\_2))$
$die(1) \leftarrow abduced(flip\_a\_coin)$
(8) $pr(feel\_guity, 1) \leftarrow abduced(kill(child\_1))$
$pr(feel\_guity, 1) \leftarrow abduced(kill(child\_2))$
$pr(feel\_guity, 0.5) \leftarrow abduced(flip\_a\_coin)$
(9) $Ai \ll Aj \leftarrow member(die(N), Ai), member(die(K), Aj), N < K$
(10) $Ai \ll Aj \leftarrow member(pr(feel\_guity, Pi), Ai), member(pr(feel\_guity, Pj), Aj), Pi < Pj$

Line 1 says that there is always expectation for every abducible declared in $\mathcal{A}_P$. In addition, if Sophie has no special reason for any child, there is no expectation to the

contrary of those abducibles (lines 5–6). Thus, the strict abductive explanations for the goal? — *decide* are:

$$\Delta_1 = \{letting\_both\_die, not\ kill(child\_1), not\ kill(child\_2), not\ flip\_a\_coin\}$$
$$\Delta_2 = \{kill(child\_1), not\ kill(child\_2), not\ flip\_a\_coin\}$$
$$\Delta_3 = \{kill(child\_2), not\ kill(child\_1), not\ flip\_a\_coin\}$$
$$\Delta_4 = \{flip\_a\_coin\}$$

e.g. it is possible for Sophie to decide to let both of her children die, choose one on her own decision, or flip a coin to decide. Then, in the next stage, a posteriori preferences are taken into account to filter out the less preferred abductive solutions. Considering the "a posteriori" preference encoded in line 9, solution that includes *letting_both_die* is ruled out since it leads to the consequence of two children dying, which is less preferred than any of the (equally) preferred remaining solutions (all with the consequence of just one child dying) (line 7). From the three remaining solutions, the ones that kill some child are ruled out since their consequences are the greater probability of Sophie to feel guilty than the one of $\{flip\_a\_coin\}$ (line 8), having taken into account "a posteriori preference" in line 10 where $member(G, A)$ is the standard system predicate meaning in this context to check whether G belongs to the set of literals representing the abductive stable model A. In short, Sophie's final decision is to flip a coin since, according to this, only one child will die and she will feel less guilty about her decision. Next consider the case when some special reason for a single child, e.g. *child_1*, is entered. Then the expectation to the contrary of killing child 1 (line 5) and of flipping a coin (line 6) are held. Therefore, only two strict abductive explanations, one including *letting_both_die* and one including $kill(child\_2)$ are available for Sophie to choose. Then, as above, by applying the "a posteriori" preference in line 9, the first one is ruled out. In other words, Sophie's final decision is to kill the *child_2*.

10.7. Implementation

10.7.1. *XSB-XASP Interface*

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[d] is one of the most sophisticated, powerful, efficient and versatile among these implementations, with a focus on execution

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

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[13,16] (standing for XSB Answer Set Programming), is included in XSB Prolog as a practical programming interface to Smodels,[14] 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 10.5.1, is that we could end up having to compute the models of the whole program in order to 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.

10.7.2. *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, as mentioned previously. Computation of such models is performed in two stages.

In the first stage, XSB computes the Well-Founded Model (WFM) of the program w.r.t. the given query, supporting goal derivation on any expected abducibles that are considered by not otherwise being defeated, as detailed in Section 10.2.1.1. Satisfaction of integrity constraints is also enforced at this stage, by always querying for $not\ \bot$ in conjunction with any specified goal.

In this way, we aim to dynamically collect only those abducibles which are necessary to prove the query. However, we cannot assume them either true or false at this stage, so they must come up undefined in the derivation tree. This is achieved by codifying considered abducibles in the following manner:

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

The latter two clauses are defined for every abducible, encoding abductive literals as even-loops over default negation, which guarantees that any considered abducibles come up undefined in the WFM of the program, and hence are contained in the residual program computed by the XSB Prolog-based meta-interpreter as explained in Section 10.7.1.

10.7.3. *Computation of Abductive Stable Models*

In the second stage, Smodels will 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 argument to *consider/1* clauses. Relevant preference rules are pre-evaluated at this stage as well, by querying for any such rules envolving any pair of the relevant abducible set.

Currently we need to assume ground literals since we will be producing a program transformation over the residual program based on the set of confirmed abducibles, which will be sent directly to Smodels. Smodels needs all literals to be instantiated, or to have an associated domain for instantiation. This limitation is present in many other state-of-the-art logic programming systems and its solution is not the main point of this work. Nevertheless, it is worth considering that

XSB's unification algorithm can ground some of the variables during the top-down derivation.

The XASP package[16] allows the programmer to collect rules in an XSB clause store. When the programmer has determined that enough clauses have been added to the store to form a semantically complete sub-program, the program is then *committed*. This means that information in the clauses is copied to Smodels, codified using Smodels data structures so that stable models of those clauses can be computed and examined.

When both the relevant abducibles and preference rules are determined, a variation of transformation Γ 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 abducible choices (i.e. those *consider/1* literals which were collected beforehand in the residual program).

10.7.4. *Inspection Points*

Given the availability of a top-down proof procedure for abduction, implementing inspection points becomes simply a matter of adapting the evaluation of derivation subtrees falling under *inspect/1* literals, by following Definition 10.6 at a meta-interpreter level.

Basically, considered abducibles evaluated under *inspect/1* subtrees are codified in the following manner:

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

All the *abduced/1* predicates are thus collected during computation of the residual program and are later checked against the abductive stable models themselves. Every $abduced(a)$ predicate must have a corresponding abducible a for the model to be accepted, as defined by transformation Γ.

10.7.5. *A Posteriori Choice Mechanisms*

If only a single model emerges from computation of the abductive stable models, goal evaluation can successfully terminate. However, in situations where multiple models still remain, there is an opportunity to introduce a posteriori choice mechanisms, which can actually be domain-specific for any given program. We account for this specificity by providing an implementation hook which the user can adopt for introducing the specific code for this final selection process.

We have currently implemented a fully working set of a posteriori choice mechanisms, more specifically those detailed in Section 10.5, but additional preference

mechanisms can be exploited in the future to choose between abductive stable models.

These are the bases of the ACORDA system,[17,18] a logic programming framework specifically designed to accomodate for abduction in evolving scenarios, using the perspectives previously outlined. In addition, we have also implemented our framework in Neg-Abdual[19] an implemented XSB-Prolog system combining constructive negation and the abduction over the well-founded semantics of Abdual.[20] All our examples have been tested with success in these systems.

10.8. Conclusions

We have shown that a priori preferences over abductive logic programs are an important tool for knowledge representation in modeling 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[4] 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 and inspection points, 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.

In,[5] the authors detect a problem with the IFF abductive proof procedure[21] of Kung and Kowalski, in what concerns the treatment of negated abducibles in integrity constraints (e.g. in 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. 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.

Our work on Inspection Points generalizes the scope of the problem they solved and solves the problems involved in this wider scope. For one we allow for passive checking not just of negated abducibles but also of positive ones, as well as passive checking of any literal, whether or not abducible, and allow also to single out which occurrences are passive or active. Thus, we can cater for both passive and active

ICs, depending on the use desired. Our solution uses abduction itself to solve the problem, making it general for use in other abductive frameworks and procedures. The declarative semantics of our approach is supplied by a straight forward, meaning preserving, program transformation whose correctness is apparent.

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 partial order for preferences, in[4] we have already shown that this need not necessarily be so, namely by specifying different conditions for revising contradictory preferences.[22] 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.

In,[23] 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.[24] provides a stimulating survey of opportunities and problems in the use of preferences, reliant on AI techniques.

References

1. J. J. Alferes and L. M. Pereira. Updates plus preferences. In eds. M. O. Aciego, I. P. de Guzmán, G. Brewka, and L. M. Pereira, *Procs. of JELIA'00*, LNAI 1919, pp. 345–360, Málaga, Spain, (2000). Springer.
2. G. Brewka. Logic programming with ordered disjunction. In ed. M. Kaufmann, *Proc. 18th National Conference on Artificial Intelligence, AAAI-02*, (2002). URL `citeseer.ist.psu.edu/brewka02logic.html`.
3. G. Brewka, I. Niemelä, and M. Truszczynski. Answer set optimization. In *Proc. IJCAI-2003*, pp. 867–872, (2003). URL `citeseer.ist.psu.edu/brewka03answer.html`.
4. P. Dell'Acqua and L. M. Pereira, Preferential theory revision, *Journal of Applied Logic*. **5**(4), 586–601, (2007).
5. F. Sadri and F. Toni. Abduction with Negation as Failure for Active and Reactive Rules. In eds. E. Lamma and P. Mello, *An Implementation for Abductive Logic Agents*, LNAI 1792, pp. 49–60. Springer-Verlag, (1999).
6. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In eds. R. Kowalski and K. A. Bowen, *5th Intl. Logic Programming Conf.*, pp. 1070–1080. MIT Press, (1988).
7. L. M. Pereira, G. Lopes, and P. Dell'Acqua. Pre and post preferences over abductive models. In eds. J. Delgrande and W. Kieling, *Multidisciplinary Workshop on Advances in Preference Handling (M-Pref'07) at 33rd Intl. Conf. on Very Large Data Bases (VLDB'07)*. VLDB, (2007). URL `http://www.mycosima.com/vldb2007-preferences/`.

8. I. Niemelä and P. Simons. Extending the Smodels system with cardinality and weight constraints. In ed. J. Minker, *Logic-Based Artificial Intelligence*, pp. 491–521. Kluwer Academic Publishers, (2000). URL `citeseer.ist.psu.edu/niemel00extending.html`.
9. L. M. Pereira and A. M. Pinto. Inspection points and meta-abduction in logic programs. Submitted, (2008).
10. D. Poole, The independent choice logic for modelling multiple agents under uncertainty, *Artificial Intelligence*. **94**(1-2), 7–56, (1997). URL `citeseer.ist.psu.edu/poole97independent.html`.
11. 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).
12. W. Styron, *Sophie's Choice*. (Bantam Books, New York, 1980).
13. L. F. Castro and D. S. Warren. An environment for the exploration of non monotonic logic programs. In ed. A. Kusalik, *Proc. of the 11th Intl. Workshop on Logic Programming Environments (WLPE'01)*, (2001).
14. I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In eds. J. Dix, U. Furbach, and A. Nerode, *4th Intl. Conf. on Logic Programming and Nonmonotonic Reasoning*, LNAI 1265, pp. 420–429, Berlin, (1997). Springer.
15. T. Swift, Tabling for non-monotonic programming, *Annals of Mathematics and Artificial Intelligence*. **25**(3-4), 201–240, (1999).
16. L. Castro, T. Swift, and D. S. Warren. *XASP: Answer Set Programming with XSB and Smodels*. `http://xsb.sourceforge.net/packages/xasp.pdf`.
17. G. Lopes and L. M. Pereira. Prospective logic programming with ACORDA. In eds. G. Sutcliffe, R. Schmidt, and S. Schulz, *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).
18. L. M. Pereira and G. Lopes. Prospective logic agents. In eds. J. M. Neves, M. F. Santos, and J. M. Machado, *Progress in Artificial Intelligence, Procs. 13th Portuguese Intl.Conf. on Artificial Intelligence (EPIA'07)*, LNAI 4874, pp. 73–86, Guimarães (December, 2007). Springer.
19. Neg-Abdual. Available at `http://centria.di.fct.unl.pt/ lmp/software.html`, (2008).
20. J. J. Alferes, L. M. Pereira, and T. Swift, Abduction in Well-Founded Semantics and Generalized Stable Models via Tabled Dual Programs, *Theory and Practice of Logic Programming*. **4**(4), 383–428, (2004).
21. T. H. Fung and R. Kowalski, The IFF Proof Procedure for Abductive Logic Programming, *The J. of Logic Programming*. **33**(2), 151–165, (1997).
22. P. Santana and L. M. Pereira. Emergence of cooperation through mutual preference revision. In eds. M. Ali and D. Richard, *Procs. of the 19th Intl. Conf. on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE'06)*, LNAI, Annecy, France, (2006). Springer.
23. H. Rott, *Change, Choice and Inference*. Oxford University Press, Oxford, (2001).
24. J. Doyle, Prospects for preferences, *Computational Intelligence*. **20**(3), 111–136, (2004).