1

Collaborative vs. Conflicting Learning, Evolution and Argumentation

Luís Moniz Pereira and Alexandre Miguel Pinto
(lmp|amp)@di.fct.unl.pt

Centro de Inteligência Artificial - CENTRIA
Universidade Nova de Lisboa

Summary. We discuss the adoption of a three-valued setting for inductive concept learning. Distinguishing between what is true, what is false and what is unknown can be useful in situations where decisions have to be taken on the basis of scarce, ambiguous, or downright contradictory information. In a three-valued setting, we learn a definition for both the target concept and its opposite, considering positive and negative examples as instances of two disjoint classes. Explicit negation is used to represent the opposite concept, while default negation is used to ensure consistency and to handle exceptions to general rules. Exceptions are represented by examples covered by the definition for a concept that belong to the training set for the opposite concept.

After obtaining the knowledge resulting from this learning process, an agent can then interact with the environment by perceiving it and acting upon it. However, in order to know what is the best course of action to take the agent must know the causes or explanations of the observed phenomena.

Abduction, or abductive reasoning, is the process of reasoning to the best explanations. It is the reasoning process that starts from a set of observations or conclusions and derives their most likely explanations. The term abduction is sometimes used to mean just the generation of hypotheses to explain observations or conclusions, given a theory. Upon observing changes in the environment or in some artifact of which we have a theory, several possible explanations (abductive ones) might come to mind. We say we have several alternative arguments to explain the observations.

One single agent exploring an environment may gather only so much information about it and that may not suffice to find the right explanations. In such case, a collaborative multi-agent strategy, where each agent explores a part of the environment and shares with the others its findings, might provide better results. We describe one such framework based on a distributed genetic algorithm enhanced by a Lamarckian operator for belief revision. The agents communicate their candidate explanations — coded as chromosomes of beliefs — by sharing them in a common pool. Another way of interpreting this communication is in the context of argumentation.

We often encounter situations in which someone is trying to persuade us of a point of view by presenting reasons for it. This is called “arguing a case” or “presenting an argument”. We can also argue with ourselves. Sometimes it is easy

to see what the issues and conclusions are, and the reasons presented, but sometimes not. In the process of taking all the arguments and trying to find a common ground or consensus we might have to change, or review, some of assumptions of each argument. Belief revision is the process of changing beliefs to take into account a new piece of information. The logical formalization of belief revision is researched in philosophy, in databases, and in artificial intelligence for the design of rational agents.

The resulting framework we present is a collaborative perspective of argumentation where arguments are put together at work in order to find the possible 2-valued consensus of opposing positions of learnt concepts.

1.1 Introduction

The scientific approach is the most skeptical one towards finding the explanations to natural phenomena. Such skeptical stance leads to a pre-disposition to continuous knowledge revision and refinement based on observations — the solid foundations for any reasonable theory. The endless scientific cycle of theory building and refinement consists of 1) the environment; 2) producing candidate theories that best cover the observations; 3) create and exploit the experiences that will best test and stress the theories; and 4) going back to step 1) by collecting new observations from the environment resulting from the experiences. This cycle is depicted in figure 1.1.

After some iterations along this theory building/refinement cycle the theory built is "good enough" in the sense that the predictions it makes are accurate "enough" concerning the environment observations resulting from experiences. At this point the theory can be used both to provide explanations to observations as well as to produce new predictions.

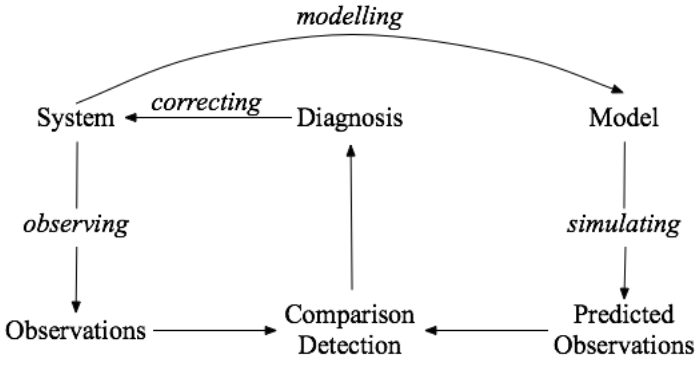

Fig. 1.1. Knowledge Model refinement cycle through Diagnosis

Throughout the years, scientists from every area of knowledge have relied upon logic to develop and refine theories and to argue about them. Logic has been an indispensable tool to build theories from observations, to use the theories to make predictions, and to revise theories when observational data contradicts the predicted results.

Expressing theories as Logic Programs has become more natural and common as the field of Computational Logic has grown mature and other fields started to use its tools and results. Theories are usually expressed as a set of 'if-then' rules and facts which allow for the derivation, through the use of logic inference, of non-obvious results. When writing such rules and facts, explicit negation, just like explicit affirmation, can be used to formalize sure knowledge which provides inevitable results. Not only has formal argumentation been characterized in terms of Logic Programs, but the various semantics of Logic Programs themselves have been characterized as the result of argumentation between competing program interpretations.

Theories can be further refined by adding special rules taking the form of Integrity Constraints (ICs). These impose that, whatever the assumptions might be, some conditions must be met. One implicit constraint on every reasonable theory is overall consistency, i.e, it must not be possible to derive one conclusion and its opposition.

Since in the real world the most common situation is one where there is incomplete and updatable information, any system making a serious attempt at dealing with real situations must cope with such complexities. To deal with this issue, the field of Computational Logic has also formalized another form of negation, Default Negation, used to express uncertain knowledge and exceptions, and used to derive results in the absence of complete information. When new information updates the theory, old conclusions might no longer be available (because they were relying on assumptions that become false with the new information), and further new conclusions might now be derived (for an analogous reasons).

The principle we use is thus the *Unknown World Assumption* (UWA) where everything is unknown or undefined until we have some solid evidence of its truthfulness or falseness. This principle differs from the more usual *Closed World Assumption* (CWA) where everything is assumed false until there is solid evidence of its truthfulness. We believe the UWA stance is more skeptical, cautious, and even more realistic than the CWA. We do not choose a fuzzy logic approach due to its necessity of specific threshold values. For such an approach we would need to compute those values *a priori*, possibly recurring to a probabilistic frequency-based calculation. Accordingly, we use a 3-valued logic (with the *undefined* truth value besides the *true* and *false* ones) instead of a more classical 2-valued logic.

We start by presenting the method for theory building from observations we use — a 3-valued logic rule learning method, and in the following section we focus on a method to analyze observations and to provide explanations for them given the learned theory. We show how the possible alternative explana-

tions can be viewed as arguments for and against some hypotheses, and how we can use these arguments in a collaborative way to find better consensual explanations. Conclusions and outlined future work close this chapter.

1.2 Theory building and refinement

In real-world problems, complete information about the world is impossible to achieve and it is necessary to reason and act on the basis of the available partial information. In situations of incomplete knowledge, it is important to distinguish between what is true, what is false, and what is unknown or undefined.

Such a situation occurs, for example, when an agent incrementally gathers information from the surrounding world and has to select its own actions on the basis of acquired knowledge. If the agent learns in a two-valued setting, it can encounter the problems that have been highlighted in [22]. When learning in a specific to general way, it will learn a cautious definition for the target concept and it will not be able to distinguish what is false from what is not yet known (see Figure 1.2a) . Supposing the target predicate represents the allowed actions, then the agent will not distinguish forbidden actions from actions with an outcome and this can restrict the agent's acting power. If the agent learns in a general to specific way (i.e., the agent starts with a most general concept and progressively restricts it by adding exceptions as he learns), instead, it will not know the difference between what is true and what is unknown (Figure 1.2b) and, therefore, it can try actions with an unknown outcome. Rather, by learning in a three-valued setting, it will be able to distinguish between allowed actions, forbidden actions, and actions with an unknown outcome (Figure 1.2c) . In this way, the agent will know which part of the domain needs to be further explored and will not try actions with an unknown outcome unless it is trying to expand its knowledge.

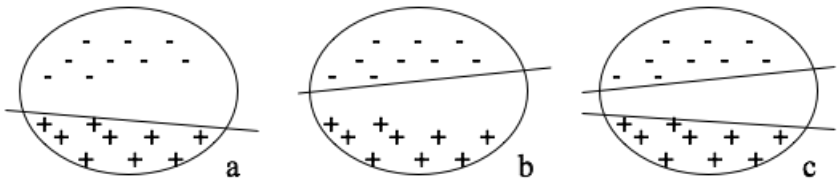

Fig. 1.2. (a,b) two-valued setting, (c) three-valued setting (Taken from [22])

In [47] the authors showed that various approaches and strategies can be adopted in Inductive Logic Programming (ILP, henceforth) for learning with Extended Logic Programs (ELP) — including explicit negation — under an

extension of well-founded semantics. As in [37, 38], where answer-sets semantics is used, the learning process starts from a set of positive and negative examples plus some background knowledge in the form of an extended logic program. Positive and negative information in the training set are treated equally, by learning a definition for both a positive concept p and its (explicitly) negated concept $\neg p$. Coverage of examples is tested by adopting the SLX [3] interpreter for ELP under the Well-Founded Semantics with explicit negation ($WFSX$) defined in [5, 25], and valid for its paraconsistent version [17].

Example 1. Explicit negation

Consider a person who just moved to another city. He has just arrived and so he does not know yet if the neighborhood he is going to live in is dangerous or not.

$$\begin{aligned} dangerous\_neighborhood &\leftarrow not\ \neg dangerous\_neighborhood \\ \neg dangerous\_neighborhood &\leftarrow not\ dangerous\_neighborhood \end{aligned}$$

Suppose now that he learns for sure that the neighborhood is not dangerous at all. In such case the program has another rule (which is actually a fact):

$$\neg dangerous\_neighborhood$$

Default negation is used in the learning process to handle *exceptions* to general rules. Exceptions are examples covered by the definition for the positive concept that belong to the training set for the negative concept or examples covered by the definition for the negative concept that belong to the training set for the positive concept.

In this work, we consider standard ILP techniques to learn a concept and its opposite. Indeed, separately learned positive and negative concepts may conflict and, in order to handle possible *contradiction*, contradictory learned rules are made defeatable by making the learned definition for a positive concept p depend on the default negation of the negative concept $\neg p$, and vice-versa, i.e., each definition is introduced as an exception to the other. This way of coping with contradiction can be even generalized for learning n disjoint classes, or modified in order to take into account preferences among multiple learning agents or information sources (see [45]).
In the learning problem we consider we want to learn an ELP from a background knowledge that is itself an ELP and from a set of positive and a set of negative examples in the form of ground facts for the target predicates.

A learning problem for ELP's was first introduced in [38] where the notion of coverage was defined by means of truth in the answer-set semantics. Here the problem definition is modified to consider coverage as truth in the preferred *WFSX*.

Definition 1 (Learning Extended Logic Programs).
Given:

- *a set $\mathcal{P}$ of possible (extended logic) programs*
- *a set E^+ of positive examples (ground facts)*
- *a set E^- of negative examples (ground facts)*
- *a non-contradictory extended logic program B (*background knowledge [1]*)*

Find:

- *an extended logic program $P \in \mathcal{P}$ such that*
  - $\forall e \in E^+ \cup \neg E^-$, $B \cup P \models_{WFSX} e$ *(*completeness*)*
  - $\forall e \in \neg E^+ \cup E^-$, $B \cup P \not\models_{WFSX} e$ *(*consistency*)*

  where $\neg E = \{\neg e | e \in E\}$.

We suppose that the training sets E^+ and E^- are disjoint. However, the system is also able to work with overlapping training sets.

The learned theory will contain rules of the form:

$$p(\mathbf{X}) \leftarrow Body^+(\mathbf{X})$$
$$\neg p(\mathbf{X}) \leftarrow Body^-(\mathbf{X})$$

for every target predicate p, where $\mathbf{X}$ stands for a tuple of arguments. In order to satisfy the completeness requirement, the rules for p will entail all positive examples while the rules for $\neg p$ will entail all (explicitly negated) negative examples. The consistency requirement is satisfied by ensuring that both sets of rules do not entail instances of the opposite element in either of the training sets.

Note that, in the case of extended logic programs, the consistency with respect to the training set is equivalent to the requirement that the program is non-contradictory on the examples. This requirement is enlarged to require that the program be non-contradictory also for unseen atoms, i.e., $B \cup P \not\models L \wedge \neg L$ for every atom L of the target predicates.

We say that an example e is *covered* by program P if $P \models_{WFSX} e$. Since the *SLX* procedure is correct with respect to *WFSX*, even for contradictory programs, coverage of examples is tested by verifying whether $P \vdash_{SLX} e$.

The approach to learning with extended logic programs considered consists in initially applying conventional ILP techniques to learn a positive definition from E^+ and E^- and a negative definition from E^- and E^+. In these techniques, the *SLX* procedure substitutes the standard Logic Programming proof procedure to test the coverage of examples.

The ILP techniques to be used depend on the level of generality that we want to have for the two definitions: we can look for the Least General Solution (LGS) or the Most General Solution (MGS) of the problem of learning each

[1] By non-contradictory program we mean a program which admits at least one *WFSX* model.

concept and its complement. In practice, LGS and MGS are not unique and real systems usually learn theories that are not the least nor most general, but closely approximate one of the two. In the following, these concepts will be used to signify approximations to the theoretical concepts.

LGSs can be found by adopting one of the bottom-up methods such as relative least general generalization (*rlgg*) [66] and the GOLEM system [57] [2] , inverse resolution [56] or inverse entailment [48]. Conversely, MGSs can be found by adopting a top-down refining method (cf. [49]) and a system such as FOIL [68] or Progol [55].

1.2.1 Strategies for Combining Different Generalizations

The generality of concepts to be learned is an important issue when learning in a three-valued setting. In a two-valued setting, once the generality of the definition is chosen, the extension (i.e., the generality) of the set of false atoms is automatically decided, because it is the complement of the true atoms set. In a three-valued setting, rather, the extension of the set of false atoms depends on the generality of the definition learned for the negative concept. Therefore, the corresponding level of generality may be chosen independently for the two definitions, thus affording four epistemological cases. The adoption of ELP allows case combination to be expressed in a declarative and smooth way.

Furthermore, the generality of the solutions learned for the positive and negative concepts clearly influences the interaction between the definitions. If we learn the MGS for both a concept and its opposite, the probability that their intersection is non-empty is higher than if we learn the LGS for both. Intuitively, this happens because, as explained above, when learning the MGS for a concept we begin with a most permissive definition for that concept and progressively refine it by adding exceptions. It is easy to see that at the very beginning of the learning process, if the MGS is used for both a concept and its opposite, these coincide. As the process of refinement goes on, the intersection of the MGS of the concept and the MGS of its opposite diminishes. Accordingly, the decision as to which type of solution to learn should take into account the possibility of interaction as well: if we want to reduce this possibility, we have to learn two LGS, if we do not care about interaction, we can learn two MGS. In general, we may learn different generalizations and combine them in distinct ways for different strategic purposes within the same application problem.

The choice of the level of generality should be made on the basis of available knowledge about the domain. Two of the criteria that can be taken into account are the damage or risk that may arise from an erroneous classification of an unseen object, and the confidence we have in the training set as to its correctness and representativeness.

[2] For a recent implementation see http://www.doc.ic.ac.uk/∼shm/Software/golem/

When classifying an as yet unseen object as belonging to a concept, we may later discover that the object belongs to the opposite concept. The more we generalize a concept, the higher is the number of unseen atoms covered by the definition and the higher is the risk of an erroneous classification. Depending on the damage that may derive from such a mistake, we may decide to take a more cautious or a more confident approach. If the possible damage from an over extensive concept is high, then one should learn the LGS for that concept, if the possible damage is low then one can generalize the most and learn the MGS. The overall risk will depend also on the use of the learned concepts within other rules: we need to take into account the damage that may derive from mistakes made on concepts depending on the target one.

The problem of selecting a solution of an inductive problem according to the cost of misclassifying examples has been studied in a number of works. PREDICTOR [34] is able to select the cautiousness of its learning operators by means of meta-heuristics. These meta-heuristics make the selection based on a user-input penalty for prediction error. In [67] Provost provides a method to select classifiers given the cost of misclassifications and the prior distribution of positive and negative instances. The method is based on the Receiver Operating Characteristic (ROC) [35] graph from signal theory that depicts classifiers as points in a graph with the number of false positives on the X axis and the number of true positive on the Y axis. In [59] it is discussed how the different costs of misclassifying examples can be taken into account into a number of algorithms: decision tree learners, Bayesian classifiers and decision list learners.

As regards the confidence in the training set, we can prefer to learn the MGS for a concept if we are confident that examples for the opposite concept are correct and representative of the concept. In fact, in top-down methods, negative examples are used in order to delimit the generality of the solution. Otherwise, if we think that examples for the opposite concept are not reliable, then we should learn the LGS.

In the following, we present a realistic example of the kind of reasoning that can be used to choose and specify the preferred level of generality, and discuss how to strategically combine the different levels by employing ELP tools to learning.

Example 2. Consider now a person living in a bad neighborhood in Los Angeles. He is an honest man and to survive he needs two concepts, one about who is likely to attack him, on the basis of appearance, gang membership, age, past dealings, etc. Since he wants to take a cautious approach, he maximizes $attacker$ and minimizes $\neg attacker$, so that his $attacker1$ concept allows him to avoid dangerous situations.

$$\begin{aligned} attacker1(X) &\leftarrow attacker_{MGS}(X) \\ \neg attacker1(X) &\leftarrow \neg attacker_{LGS}(X) \end{aligned}$$

Another concept he needs is the type of beggars he should give money to (he is a good man) that actually seem to deserve it, on the basis of appearance,

health, age, etc. Since he is not rich and does not like to be tricked, he learns a $beggar1$ concept by minimizing $beggar$ and maximizing $\neg beggar$, so that his beggar concept allows him to give money strictly to those appearing to need it without faking.

$$\begin{array}{lcl} beggar1(X) & \leftarrow & beggar_{LGS}(X) \\ \neg beggar1(X) & \leftarrow & \neg beggar_{MGS}(X) \end{array}$$

However, rejected beggars, especially malicious ones, may turn into attackers, in this very bad neighborhood. Consequently, if he thinks a beggar might attack him, he had better be more permissive about who is a beggar and placate him with money. In other words, he should maximize $beggar$ and minimize $\neg beggar$ in a $beggar2$ concept.

$$\begin{array}{lcl} beggar2(X) & \leftarrow & beggar_{MGS}(X) \\ \neg beggar2(X) & \leftarrow & \neg beggar_{LGS}(X) \end{array}$$

These concepts can be used in order to minimize his risk taking when he carries, by his standards, a lot of money and meets someone who is likely to be an attacker, with the following kind of reasoning:

$$\begin{array}{lcl} run(X) & \leftarrow & lot\_of\_money(X), meets(X,Y), attacker1(Y), \\ & & not\ beggar2(Y) \\ \neg run(X) & \leftarrow & lot\_of\_money(X), give\_money(X,Y) \\ give\_money(X,Y) & \leftarrow & meets(X,Y), beggar1(Y) \\ give\_money(X,Y) & \leftarrow & meets(X,Y), attacker1(Y), beggar2(Y) \end{array}$$

If he does not have a lot of money on him, he may prefer not to run as he risks being beaten up. In this case he has to relax his attacker concept into $attacker2$, but not relax it so much that he would use $\neg attacker_{MGS}$.

$$\begin{array}{lcl} \neg run(X) & \leftarrow & little\_money(X), meets(X,Y), attacker2(Y) \\ attacker2(X) & \leftarrow & attacker_{LGS}(X) \\ \neg attacker2(X) & \leftarrow & \neg attacker_{LGS}(X) \end{array}$$

The various notions of *attacker* and *beggar* are then learned on the basis of previous experience the man has had (see [47]).

1.2.2 Strategies for Eliminating Learned Contradictions

The learned definitions of the positive and negative concepts may overlap. In this case, we have a contradictory classification for the objective literals[3] in the intersection. In order to resolve the conflict, we must distinguish two types

[3] An 'objective literal' in a Logic Program is just an atom, possibly explicitly negated. E.g., '$attacker2(X)$' and '$\neg attacker2(X)$' in example 2 are objective literals.

of literals in the intersection: those that belong to the training set and those that do not, also dubbed *unseen* atoms (see Figure 1.3).

In the following we discuss how to resolve the conflict in the case of unseen literals and of literals in the training set. We first consider the case in which the training sets are disjoint, and we later extend the scope to the case where there is a non-empty intersection of the training sets, when they are less than perfect. From now onwards, $\mathbf{X}$ stands for a tuple of arguments.

For unseen literals, the conflict is resolved by classifying them as undefined, since the arguments supporting the two classifications are equally strong. Instead, for literals in the training set, the conflict is resolved by giving priority to the classification stipulated by the training set. In other words, literals in a training set that are covered by the opposite definition are considered as *exceptions* to that definition.

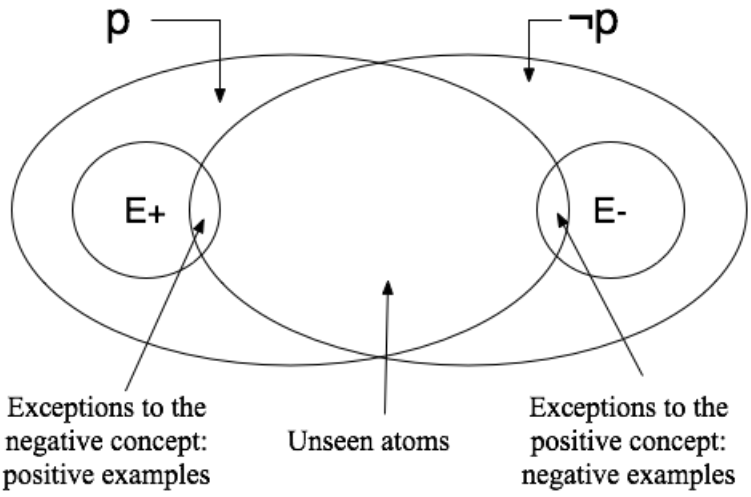

Fig. 1.3. Interaction of the positive and negative definitions on exceptions

Contradiction on Unseen Literals

For unseen literals in the intersection, the undefined classification is obtained by making opposite rules mutually defeasible, or "non-deterministic" (see [10, 5]). The target theory is consequently expressed in the following way:

$$p(\mathbf{X}) \leftarrow p^{+}(\mathbf{X}), not\ \neg p(\mathbf{X})$$
$$\neg p(\mathbf{X}) \leftarrow p^{-}(\mathbf{X}), not\ p(\mathbf{X})$$

where $p^{+}(\mathbf{X})$ and $p^{-}(\mathbf{X})$ are, respectively, the definitions learned for the positive and the negative concept, obtained by renaming the positive predicate

by p^+ and its explicit negation by p^-. From now onwards, we will indicate with these superscripts the definitions learned separately for the positive and negative concepts.

We want both $p(\mathbf{X})$ and $\neg p(\mathbf{X})$ to act as an exception to the other. In case of contradiction, this will introduce mutual circularity, and hence undefinedness according to *WFSX*. For each literal in the intersection of p^+ and p^-, there are two stable models, one containing the literal, the other containing the opposite literal. According to *WFSX*, there is a third (partial) stable model where both literals are undefined, i.e., no literal $p(\mathbf{X})$, $\neg p(\mathbf{X})$, *not* $p(\mathbf{X})$ or *not* $\neg p(\mathbf{X})$ belongs to the well-founded (or least partial stable) model. The resulting program contains a recursion through negation (i.e., it is non-stratified) but the top-down *SLX* procedure does not go into a loop because it comprises mechanisms for loop detection and treatment, which are implemented by *XSB* Prolog through tabling.

Example 3. Let us consider the Example of Section 1.2.1. In order to avoid contradictions on unseen atoms, the learned definitions must be:

$$
\begin{array}{ll}
attacker1(X) & \leftarrow attacker^+_{MGS}(X), not\ \neg attacker1(X) \\
\neg attacker1(X) & \leftarrow attacker^-_{LGS}(X), not\ attacker1(X) \\
beggar1(X) & \leftarrow beggar^+_{LGS}(X), not\ \neg beggar1(X) \\
\neg beggar1(X) & \leftarrow beggar^-_{MGS}(X), not\ beggar1(X) \\
beggar2(X) & \leftarrow beggar^+_{MGS}(X), not\ \neg beggar2(X) \\
\neg beggar2(X) & \leftarrow beggar^-_{LGS}(X), not\ beggar2(X) \\
attacker2(X) & \leftarrow attacker^+_{LGS}(X), not\ \neg attacker2(X) \\
\neg attacker2(X) & \leftarrow attacker^-_{LGS}(X), not\ attacker2(X)
\end{array}
$$

Note that $p^+(\mathbf{X})$ and $p^-(\mathbf{X})$ can display as well the undefined truth value, either because the original background is non-stratified or because they rely on some definition learned for another target predicate, which is of the form above and therefore non-stratified. In this case, three-valued semantics can produce literals with the value "undefined", and one or both of $p^+(\mathbf{X})$ and $p^-(\mathbf{X})$ may be undefined. If one is undefined and the other is true, then the rules above make both p and $\neg p$ undefined, since the negation by default of an undefined literal is still undefined. However, this is counter-intuitive: a defined value should prevail over an undefined one.

In order to handle this case, we suppose that a system predicate $undefined(X)$ is available[4], that succeeds if and only if the literal X is undefined. So we add the following two rules to the definitions for p and $\neg p$:

$$
\begin{array}{c}
p(\mathbf{X}) \leftarrow p^+(\mathbf{X}), undefined(p^-(\mathbf{X})) \\
\neg p(\mathbf{X}) \leftarrow p^-(\mathbf{X}), undefined(p^+(\mathbf{X}))
\end{array}
$$

[4] The *undefined* predicate can be implemented through negation NOT under CWA ($NOT\ P$ means that P is false whereas *not* means that P is false or undefined), i.e., $undefined(P) \leftarrow NOT\ P, NOT(not\ P)$.

According to these clauses, $p(\mathbf{X})$ is true when $p^+(\mathbf{X})$ is true and $p^-(\mathbf{X})$ is undefined, and conversely.

Contradiction on Examples

Theories are tested for consistency on all the literals of the training set, so we should not have a conflict on them. However, in some cases, it is useful to relax the consistency requirement and learn clauses that cover a small amount of counterexamples. This is advantageous when it would be otherwise impossible to learn a definition for the concept, because no clause is contained in the language bias that is consistent, or when an overspecific definition would be learned, composed of many specific clauses instead of a few general ones. In such cases, the definitions of the positive and negative concepts may cover examples of the opposite training set. These must then be considered exceptions, which are then due to abnormalities in the opposite concept.

Let us start with the case where some literals covered by a definition belong to the opposite training set. We want of course to classify these according to the classification given by the training set, by making such literals *exceptions*. To handle exceptions to classification rules, we add a negative default literal of the form $not\ abnorm_p(\mathbf{X})$ (resp. $not\ abnorm_{\neg p}(\mathbf{X})$) to the rule for $p(\mathbf{X})$ (resp. $\neg p(\mathbf{X})$), to express possible abnormalities arising from exceptions. Then, for every exception $p(\mathbf{t})$, an individual fact of the form $abnorm_p(\mathbf{t})$ (resp. $abnorm_{\neg p}(\mathbf{t})$) is asserted so that the rule for $p(\mathbf{X})$ (resp. $\neg p(\mathbf{X})$) does not cover the exception, while the opposite definition still covers it. In this way, exceptions will figure in the model of the theory with the correct truth value. The learned theory thus takes the form:

$$p(\mathbf{X}) \leftarrow p^+(\mathbf{X}), not\ abnorm_p(\mathbf{X}), not\ \neg p(\mathbf{X}) \tag{1.1}$$

$$\neg p(\mathbf{X}) \leftarrow p^-(\mathbf{X}), not\ abnorm_{\neg p}(\mathbf{X}), not\ p(\mathbf{X}) \tag{1.2}$$

$$p(\mathbf{X}) \leftarrow p^+(\mathbf{X}), undefined(p^-(\mathbf{X})) \tag{1.3}$$

$$\neg p(\mathbf{X}) \leftarrow p^-(\mathbf{X}), undefined(p^+(\mathbf{X})) \tag{1.4}$$

Abnormality literals have not been added to the rules for the undefined case because a literal which is an exception is also an example, and so must be covered by its respective definition; therefore it cannot be undefined.

Notice that if E^+ and E^- overlap for some example $p(\mathbf{t})$, then $p(\mathbf{t})$ is classified $false$ by the learned theory. A different behavior would be obtained by slightly changing the form of learned rules, in order to adopt, for atoms of the training set, one classification as default and thus give preference to false (negative training set) or true (positive training set).

Individual facts of the form $abnorm_p(\mathbf{X})$ might be then used as examples for learning a definition for $abnorm_p$ and $abnorm_{\neg p}$, as in [30, 38]. In turn, exceptions to the definitions of $abnorm_p$ and $abnorm_{\neg p}$ might be found and so on, thus leading to a hierarchy of exceptions (for our hierarchical learning of exceptions, see [44, 74]).

Example 4. Consider a domain containing entities a, b, c, d, e, f and suppose the target concept is $flies$. Let the background knowledge be:

$bird(a)$	$has\_wings(a)$	
$jet(b)$	$has\_wings(b)$	
$angel(c)$	$has\_wings(c)$	$has\_limbs(c)$
$penguin(d)$	$has\_wings(d)$	$has\_limbs(d)$
$dog(e)$		$has\_limbs(e)$
$cat(f)$		$has\_limbs(f)$

and let the training set be:

$E^+ = \{flies(a)\}$ $\quad\quad$ $E^- = \{flies(d), flies(e)\}$

A possible learned theory is:

$$\begin{aligned} flies(X) &\leftarrow flies^+(X), not\ abnormal_{flies}(X), not\ \neg flies(X) \\ \neg flies(X) &\leftarrow flies^-(X), not\ flies(X) \\ flies(X) &\leftarrow flies^+(X), undefined(flies^-(X)) \\ \neg flies(X) &\leftarrow flies^-(X), undefined(flies^+(X)) \\ abnormal_{flies}(d) &\leftarrow true \end{aligned}$$

where $flies^+(X) \leftarrow has\_wings(X)$ and $flies(X)^- \leftarrow has\_limbs(X)$.

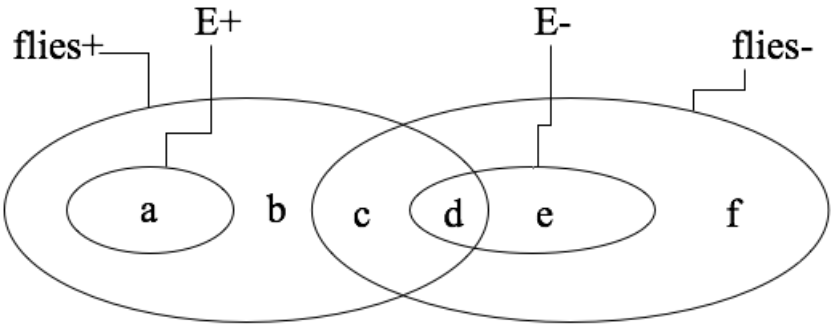

Fig. 1.4. Coverage of definitions for opposite concepts

The example above and Figure 1.4 show all the possible cases for a literal when learning in a three-valued setting. a and e are examples that are consistently covered by the definitions. b and f are unseen literals on which there is no contradiction. c and d are literals where there is contradiction, but c is classified as undefined whereas d is considered as an exception to the positive definition and is classified as negative.

Identifying contradictions on unseen literals is useful in interactive theory revision, where the system can ask an oracle to classify the literal(s) leading to contradiction, and accordingly revise the least or most general solutions for p and for $\neg p$ using a theory revision system such as REVISE [15] or CLINT [21, 23]. Detecting uncovered literals points to theory extension.

Extended logic programs can be used as well to represent n disjoint classes $p_1, \ldots, p_n$. When one has to learn n disjoint classes, the training set contains a number of facts for a number of predicates $p_1, \ldots, p_n$. Let p_i^+ be a definition learned by using, as positive examples, the literals in the training set classified as belonging to p_i and, as negative examples, all the literals for the other classes. Then the following rules ensure consistency on unseen literals and on exceptions regardless of the algorithm used for learning the p_i^+.

$$
\begin{array}{lcl}
p_1(\mathbf{X}) & \leftarrow & p_1^+(\mathbf{X}), not\ abnormal_{p_1}(\mathbf{X}), not\ p_1(\mathbf{X}), not\ p_2(\mathbf{X}), \ldots, not\ p_n(\mathbf{X}) \\
p_1(\mathbf{X}) & \leftarrow & p_1^+(\mathbf{X}), not\ abnormal_{p_1}(\mathbf{X}), not\ p_2(\mathbf{X}), \ldots, not\ p_n(\mathbf{X}) \\
p_2(\mathbf{X}) & \leftarrow & p_2^+(\mathbf{X}), not\ abnormal_{p_2}(\mathbf{X}), not\ p_1(\mathbf{X}), not\ p_3(\mathbf{X}), \ldots, not\ p_n(\mathbf{X}) \\
\ldots & \leftarrow & \ldots \\
p_n(\mathbf{X}) & \leftarrow & p_n^+(\mathbf{X}), not\ abnormal_{p_n}(\mathbf{X}), not\ p_1(\mathbf{X}), \ldots, not\ p_{n-1}(\mathbf{X})
\end{array}
$$

$$
\begin{array}{lcl}
p_1(\mathbf{X}) & \leftarrow & p_1^+(\mathbf{X}), undefined(p_2^+(\mathbf{X})), \ldots, undefined(p_n^+(\mathbf{X})) \\
p_2(\mathbf{X}) & \leftarrow & p_2^+(\mathbf{X}), undefined(p_1^+(\mathbf{X})), undefined(p_3^+(\mathbf{X})), \ldots, \\
 & & undefined(p_n^+(\mathbf{X})) \\
\ldots & \leftarrow & \ldots \\
p_n(\mathbf{X}) & \leftarrow & p_n^+(\mathbf{X}), undefined(p_1^+(\mathbf{X})), \ldots, undefined(p_{n-1}^+(\mathbf{X}))
\end{array}
$$

1.3 Observation Analysis and Explanation

After a theory is built it can now be used to analyze observations and to provide explanations for them. Such explanations are sets of abductive hypotheses which, when assumed true under the theory at hand, yield the observations as conclusions. We can also understand each such set of hypotheses as an argument explaining why the observations hold. There can be, of course, many different possible explanations, or arguments. In the end, most of the times, we want to find the single "best" explanation for the observations, and hence we must have some mechanism to identify the "best" solution among the several alternative ones.

1.3.1 Abduction

Deduction and abduction differ in the direction in which a rule like "a entails b" is used for inference. Deduction allows deriving b as a consequence of a ; i.e., deduction is the process of deriving the consequences of what is known. Abduction allows deriving a as a hypothetical explanation of b. Abduction works in reverse to deduction, by allowing the precondition a of "a entails b" to

be derived from the consequence b, i.e., abduction is the process of explaining what is known. Charles Saunders Peirce [60] introduced abduction into logic, to mean the use of a rule or hypothetical fact to explain an observation, e.g. "if it rains the grass is wet" is used to explain why the grass is wet, given that it has rained, or vice-versa. In logic, abduction is done from a logical theory T representing a domain and a set of observations O. Abduction is the process of deriving a set of explanations of O according to T. For E to be an explanation of O according to T, it should satisfy two conditions:

- O follows from E and T;
- E is consistent with T.

In formal logic, O and E are assumed to be sets of literals. The two conditions for E being an explanation of O according to theory T are:

- $T \cup E \models O$;
- $T \cup E$ is consistent.

Among the possible explanations E satisfying these two conditions, a condition of minimality is usually imposed to avoid irrelevant facts (i.e. not contributing to the entailment of O) to be included in the explanations. An application of abduction is that of detecting faults in systems: given a theory relating faults with their effects and a set of observed effects, abduction can be used to derive sets of faults that are likely to be the cause of the problem. Belief revision, the process of adapting beliefs in view of new information, is another field in which abduction has been applied. The main problem of belief revision is that the new information may be inconsistent with the corpus of beliefs, while the result of the incorporation must not be inconsistent.

1.3.2 An Argumentation Perspective

When different alternative explanations arise, people argue for and against their theories and others'. For a long time, because of its origins in rhetoric and the law, argumentation has been thought of as a kind of a battle where two (or more) opposing opinions are formalized into arguments, and logic is used for the framing rules for the battle and to decide the outcome. At the end of the battle one of the arguers will be 'right' and the other(s) 'wrong'. The 'winner' is the one whose argument, attacking others' arguments, cannot be counter-attacked. The problem of argumentation becomes more complex when all arguments successfully attack each other's corresponding to different possible opinions, not inevitable conclusions. In this case, argumentation could take a new flavor, one of Collaboration besides Conflict.

1.3.3 Finding alternative explanations for observations

Trying to find explanations for observations can be implemented by simply finding the alternative abductive models that satisfy both the theory's rules

and the observations. The latter can be coded as Integrity Constraints (ICs) which are added to the theory thereby imposing the truthfulness of the observations they describe.

Example 5. Running example

We will use this running example throughout the rest of the chapter.

Consider the following Logic Program consisting of four rules. According to this program a 'professional' is someone who is a regular employee or someone who is a boss in some company. Also, a non-employee is assumed to be a student as well as all those who are junior (all children should go to school!).

$$professional(X) \leftarrow employee(X) \qquad student(X) \leftarrow not\ employee(X)$$
$$professional(X) \leftarrow boss(X) \qquad student(X) \leftarrow junior(X)$$

For now keep this example in mind as we will use it to illustrate the concepts and methods we are about to describe. Assume that '$employee/1$', '$boss/1$', and '$junior/1$' are abducible hypotheses.

Adding one single IC to the theory might yield several alternative 2-valued models (sets of abductive hypotheses) satisfying it, let alone adding several ICs.

In the example above, adding just the Integrity Constraint '$\bot \leftarrow not\ professional(john)$' — coding the fact that John is a professional — would yield two alternative abductive solutions: $\{employee(john)\}$ and $\{boss(john)\}$.

When the information from several observations comes in at one single time, several ICs must be added to the theory in order to be possible to obtain the right explanations for the corresponding observations.

In a fairly complex knowledge domain coded in a complex and lengthy theory, finding each one alternative explanation for a given observation can be quite hard and time consuming, let alone finding the "best" explanation. In general, following Occam's principle, the "best" explanation for any given observation is usually the simplest one, i.e., the one recurring to the fewest number of hypotheses — the minimal set of hypotheses.

In [46] the authors presented a method for finding the minimal belief revision solution for a set of observations. Therein, each belief corresponds to an abducible hypothesis and the belief revision is the process of finding the set of hypotheses that conforms to the observations by revising their truth value from *true* to *false* or vice-versa. Minimality is required for compliance with the Occam's principle.

In [46] the authors also code observations as ICs added to the theory, but they recur to finding the support sets for *falsum* ($\bot$)— the special reserved atom for the heads of the rules coding ICs, in order to find the belief revisions necessary to comply to the ICs. After finding such support sets they can identify the minimal sets of hypotheses that need to be revised in order to prevent *falsum* from being derived.

Here we are also concerned with finding explanations to observations, but the method used is quite different. In a nutshell, we split the set of observations into several smaller subsets; then we create several agents and give each agent the same base theory and a subset of the observations coded as ICs. We then allow each agent to come up with several alternative explanations to its ICs; the explanations need not be minimal sets of hypotheses.

Going back again to our running example, if we also know that John is a student, besides adding the '$\bot \leftarrow$ *not* $professional(john)$' IC we must also add the '$\bot \leftarrow$ *not* $student(john)$' IC.

Finding possible alternative explanations is one problem; finding which one(s) is(are) the "best" is another issue. In the next section we assume "best" means minimal set of hypotheses and we describe the method we use to find such best. Another interpretation of "best" could be "most probable" and in this case the theory inside the agents must contain the adequate probabilistic information. One such possibility would be the one described in [9]. We do not pursue this approach yet, but we consider it for future work, namely following the principles in [9].

1.3.4 Choosing the best explanation

Ex contradictione quodlibet. This well-known Latin saying means "Anything follows from contradiction". But contradictory, oppositional ideas and arguments can be combined together in different ways to produce new ideas. Since "anything follows from contradiction" one of the things that might follow from it is a solution to a problem to which several alternative positions contribute.

One well known method for solving complex problems widely used by creative teams is that of 'brainstorming'. In a nutshell, every agent participating in the 'brainstorm' contributes by adding one of his/her ideas to the common idea-pool shared by all the agents. All the ideas, sometimes clashing and oppositional among each other, are then mixed, crossed and mutated. The solution to the problem arises from the pool after a few iterations of this evolutionary process.

The evolution of alternative ideas and arguments in order to find a collaborative solution to a group problem is the underlying inspiration of this work.

Evolutionary Inspiration

Darwin's theory is based on the concept of natural selection: only those individuals that are most fit for their environment survive, and are thus able to generate new individuals by means of reproduction. Moreover, during their lifetime, individuals may be subject to random mutations of their genes that they can transmit to offspring. Lamarck's [42] theory, instead, states that evolution is due to the process of adaptation to the environment that an

individual performs in his/her life. The results of this process are then automatically transmitted to his/her offspring, via its genes. In other words, the abilities learned during the life of an individual can modify his/her genes.

Experimental evidence in the biological kingdom has shown Darwin's theory to be correct and Lamarck's to be wrong. However, this does not mean that the process of adaptation (or learning) does not influence evolution. Baldwin [8] showed how learning could influence evolution: if the learned adaptations improve the organism's chance of survival then the chances for reproduction are also improved. Therefore there is selective advantage for genetically determined traits that predisposes the learning of specific behaviors. Baldwin moreover suggests that selective pressure could result in new individuals to be born with the learned behavior already encoded in their genes. This is known as the Baldwin effect. Even if there is still debate about it, it is accepted by most evolutionary biologists.

Lamarckian evolution [43] has recently received a renewed attention because it can model cultural evolution. In this context, the concept of "meme" has been developed. A meme is the cognitive equivalent of a gene and it stores abilities learned by an individual during his lifetime, so that they can be transmitted to his offspring.

In the field of genetic programming [41], Lamarckian evolution has proven to be a powerful concept and various authors have investigated the combination of Darwinian and Lamarckian evolution.

In [46] the authors propose a genetic algorithm for belief revision that includes, besides Darwin's operators of selection, mutation and crossover, a logic based Lamarckian operator as well. This operator differs from Darwinian ones precisely because it modifies a chromosome coding beliefs so that its fitness is improved by experience rather than in a random way. There, the authors showed that the combination of Darwinian and Lamarckian operators are useful not only for standard belief revision problems, but especially for problems where different chromosomes may be exposed to different constraints, as in the case of a multi-agent system. In these cases, the Lamarckian and Darwinian operators play different roles: the Lamarckian one is employed to bring a given chromosome closer to a solution (or even find an exact one) to the current belief revision problem, whereas the Darwinian ones exert the role of randomly producing alternative belief chromosomes so as to deal with unencountered situations, by means of exchanging genes amongst them.

Evolving Beliefs

Belief revision is an important functionality that agents must exhibit: agents should be able to modify their beliefs in order to model the outside world. What's more, as the world may be changing, a pool of separately and jointly evolved chromosomes may code for a variety of distinct belief evolution potentials that can respond to world changes as they occur. This dimension has been explored in [46] with specific experiments to that effect. Mark that it is

not our purpose to propose here a competitor to extant classical belief revision methods, in particular as they apply to diagnosis. More ambitiously, we do propose a new and complementary methodology, which can empower belief revision — any assumption based belief revision — to deal with time/space distributed, and possibly intermittent or noisy laws about an albeit varying artifact or environment, possibly by a multiplicity of agents which exchange diversified genetically encoded experience. We consider a definition of the belief revision problem that consists in removing a contradiction from an extended logic program by modifying the truth value of a selected set of literals corresponding to the abducible hypotheses. The program contains as well clauses with *falsum* ($\bot$) in the head, representing ICs. Any model of the program must ensure the body of ICs false for the program to be non-contradictory. Contradiction may also arise in an extended logic program when both a literal L and its opposite $\neg L$ are obtainable in the model of the program. Such a problem has been widely studied in the literature, and various solutions have been proposed that are based on abductive logic proof procedures. The problem can be modeled by means of a genetic algorithm, by assigning to each abducible of a logic program a gene in a chromosome. In the simplest case of a two valued revision, the gene will have the value 1 if the corresponding abducible is *true* and the value 0 if the abducible is *false* . The fitness functions that can be used in this case are based on the percentage of ICs that are satisfied by a chromosome. This is, however, an over-simplistic approach since it assumes every abducible is a predicate with arity 0, otherwise a chromosome would have as many genes as the number of all possible combinations of ground values for variables in all abducibles.

Specific Belief Evolution Method

In multi-agent joint belief revision problems, agents usually take advantage of each other's knowledge and experience by explicitly communicating messages to that effect. In our approach, however, we introduce a new and complementary method (and some variations of it), in which we allow knowledge and experience to be coded as genes in an agent. These genes are exchanged with those of other agents, not by explicit message passing but through the crossover genetic operator. Crucial to this endeavor, a logic-based technique for modifying cultural genes, i.e. memes, on the basis of individual agent experience is used.

The technique amounts to a form of belief revision, where a meme codes for an agent's belief or assumptions about a piece of knowledge, and which is then diversely modified on the basis of how the present beliefs may be contradicted by laws (expressed as ICs). These mutations have the effect of attempting to reduce the number of unsatisfied constraints. Each agent possesses a pool of chromosomes containing such diversely modified memes, or alternative assumptions, which cross-fertilize Darwinianly amongst themselves. Such an experience in genetic evolution mechanism is aptly called Lamarckian.

Since we will subject the sets of beliefs to an evolutionary process (both Darwinian and Lamarckian) we will henceforth refer to this method as "Belief Evolution" (BE) instead of the classical "Belief Revision" (BR).

General Description of the Belief Evolution Method

Each agent keeps a population of chromosomes and finds a solution to the BE problem by means of a genetic algorithm. We consider a formulation of the distributed BE problem where each agent has the same set of abducibles and the same program expressed theory, but is exposed to possibly different constraints. Constraints may vary over time, and can differ because agents may explore different regions of the world. The genetic algorithm we employ allows each agent to cross over its chromosomes with chromosomes from other agents. In this way, each agent can be prepared in advance for situations that it will encounter when moving from one place to another.

The algorithm proposed for BE extends the standard genetic algorithm in two ways:

- crossover is performed among chromosomes belonging to different agents[5],
- a Lamarckian operator called Learn is added in order to bring a chromosome closer to a correct revision by changing the value of abducibles

The Structure of a Chromosome

In BR and BE, each individual hypothesis is described by the truth value of all the abducibles. Several possibilities of increasing complexity and expressive power arise now. Concerning truth values we can have 2-valued and 3-valued revisions — if we are not considering multi-valued logics. Orthogonally to this criterion we can eventualy encode more information in each gene of a chromosome. In particular, some possibilities are:

- each gene encodes a ground literal — all its variables are bound to fixed values
- each gene encodes a literal with non-ground variables plus constraints restricting the possible values for the free variables

Surely there are many other possibilities for the information each gene can encode, but in this work we restrict ourselves to the first one above. In such case, we represent a chromosome as a list of genes and memes, and different chromosomes may contain information about different genes. This implies a major difference to traditional genetic algorithms where every chromosome refers exactly to the same genes and the crossover and mutation operations are somewhat straightforward.

The memes in a chromosome will be just like genes — representing abducibles — but they will have extra information. Each meme has associated

[5] Similarly to what happens with island models [75].

with it a counter keeping record of how many times the meme has been confirmed or refuted. Each time a meme is confirmed this value is increased, and each time it is refuted the value decreases. This value provides thus a measure of confidence in the corresponding meme.

Example 6. Running example (cont.)

Continuing with our running example, let us assume that both $professional(john)$ and $student(john)$ have been observed.

We can create two agents, each with the same rule-set theory, and split the observations among them. We would have thus

Agent 1:

$$\begin{array}{lll} & \leftarrow & not\ professional(john) \\ professional(X) & \leftarrow & employee(X) \\ professional(X) & \leftarrow & boss(X) \\ student(X) & \leftarrow & not\ employee(X) \\ student(X) & \leftarrow & junior(X) \end{array}$$

Agent 2:

$$\begin{array}{lll} & \leftarrow & not\ student(john) \\ professional(X) & \leftarrow & employee(X) \\ professional(X) & \leftarrow & boss(X) \\ student(X) & \leftarrow & not\ employee(X) \\ student(X) & \leftarrow & junior(X) \end{array}$$

In the simplest case where a gene encodes an abductive ground literal Agent 1 would come up with two alternative abductive solutions for its IC '$\bot \leftarrow\ not\ professional(john)$': $\{employee(john)\}$ and $\{boss(john)\}$. Moreover, Agent 2 would come up with two other alternative abductive solutions for its IC '$\bot \leftarrow\ not\ student(john)$': $\{not\ employee(john)\}$ and $\{junior(john)\}$.

Crossover

When a chromosome is a list of abducible hypotheses (with or without constraints over variables), as it is in the case we present here, the crossover and mutation operations cannot fallback into the well-known versions of standard genetic algorithms. If two different chromosomes, each encoding information about different abducibles, are to be crossed over there are more possibilities other than simply selecting cut points and switching genes between cut points.

As described above, each agent produces several chromosomes which are lists of abducible hypotheses needed to respect the ICs the agent knows. Since each agent knows only some ICs the abductive answer the algorithm seeks should be a combination of the partial answers each agent comes up with. In principle, the overlap on abducibles among two chromosomes coming from different agents should be less than total — after all, each agent is taking care

of its own ICs which, in principle, do not refer to the exact same abducibles. Therefore, crossing over such chromosomes can simply turn out to be the merging of the chromosomes, i.e., the concatenation of the lists of abducibles.

If several ICs refer to the exact same abducibles the chromosomes from different agents will contain either the same gene — in which case we can see this as an 'agreement' between the agents as far as the corresponding abducible is concerned — or genes stating contradictory information about the same abducible. In this last case if the resulting concatenated chromosome turns out to be inconsistent in itself the fitness function will filter it out by assigning it a very low value.

Example 7. Running example (cont.)

Continuing with our running example, recall that Agent 1 would come up with two alternative abductive solutions for its IC '$\bot \leftarrow$ *not professional*$(john)$': $\{employee(john)\}$ and $\{boss(john)\}$. Moreover, Agent 2 would come up with two other alternative abductive solutions for its IC '$\bot \leftarrow$ *not student*$(john)$': $\{$*not employee*$(john)\}$ and $\{junior(john)\}$.

The crossing over of these chromosomes will yield the four combinations $\{employee(john), $*not employee*$(john)\}$, $\{employee(john), junior(john)\}$, $\{boss(john), $*not employee*$(john)\}$, and $\{boss(john), junior(john)\}$.

The first resulting chromosome is contradictory so it will be filtered out by the fitness function. The second chromosome correspond to the situation where John is a junior employee who is still studying — a quite common situation, actually. The third chromosome corresponds to the situation where John is a senior member of a company — a 'boss' — who is taking some course (probably a post-graduation study). The last chromosome could correspond to the situation of a young entrepreneur who, besides owning his/hers company, is also a student — this is probably an uncommon situation and, if necessary, the fitness function can reflect that "unprobability".

Mutation

When considering a list of abducible literals the mutation operation resembles the standard mutation of genetic algorithms by changing one gene to its opposite; in this case negating the truth value of the abducted literal.

Example 8. Running example (cont.)

In the example we have been using this could correspond to mutating the chromosome $\{$*not employee*$(john)\}$ to $\{employee(john)\}$, or to mutating the chromosome $\{junior(john)\}$ to $\{$*not junior*$(john)\}$.

The Lamarckian Learn operator

The Lamarckian operator Learn can change the values of variables of an abducible in a chromosome c_i so that a bigger number of constraints is satisfied, thus bringing c_i closer to a solution. Learn differs from a normal belief revision operator because it does not assume that all abducibles are false by

CWA before the revision but it starts from the truth values that are given by the chromosome c_i. Therefore, it has to revise the values of variables of some abducibles and, in the particular case of an abducible without variables, from *true* to *false* or from *false* to *true* .

In the running example this could correspond, for example, to changing the chromosome $\{junior(john)\}$ to $\{junior(mary)\}$, where 'mary' is another value in the domain range of the variable for abducible $junior/1$.

This Lamarckian *Learn* operator will introduce an extra degree of flexibility allowing for changes to a chromosome to induce the whole belief evolution algorithm to search a solution considering new values for variables.

The Fitness Functions

Various fitness functions can be used in belief revision. The simplest fitness function is the following

$$Fitness(c_i) = \frac{\frac{n_i}{n}}{1 + NC} \tag{1.5}$$

where n_i is the number of integrity constraints satisfied by chromosome c_i, n is the total number of integrity constraints, and NC is the number of contradictions in chromosome c_i. We will call it an accuracy fitness function.

1.4 Argumentation

In [28], the author shows that preferred maximal scenarios (with maximum default negated literals — the hypotheses) are always guaranteed to exist for NLPs; and that when these yield 2-valued complete (total), consistent, admissible scenarios, they coincide with the Stable Models of the program. However, preferred maximal scenarios are, in general, 3-valued. The problem we address now is how to define 2-valued complete models based on preferred maximal scenarios. In [64] the authors took a step further from what was achieved in [28], extending its results. They did so by completing a preferred set of hypotheses rendering it approvable, ensuring whole model consistency and 2-valued completeness.

The resulting semantics thus defined, dubbed Approved Models [64], is a conservative extension to the widely known Stable Models semantics [32] in the sense that every Stable Model is also an Approved Model. The Approved Models are guaranteed to exist for every Normal Logic Program, whereas Stable Models are not. Concrete examples in [64] show how NLPs with no Stable Models can usefully model knowledge, as well as produce additional models. Moreover, this guarantee is crucial in program composition (say, from knowledge originating in divers sources) so that the result has a semantics. It is important too to warrant the existence of semantics after external updating, or in Stable Models based self-updating [1].

For the formal presentation and details of the Approved Models semantics see [64].

1.4.1 Intuition

Most of the ideas and notions of argumentation we are using here come from the Argumentation field — mainly from the foundational work of Phan Minh Dung in [28]. In [64] the *Reductio ad Absurdum* reasoning principle is also considered. This has been studied before in [62], [63], and [65].

Definition 2. ***Argument.*** *In [28] the author presents an argument as*

> *"an abstract entity whose role is solely determined by its relations to other arguments. No special attention is paid to the internal structure of the arguments."*

In this paper we will pay attention to the internal structure of an argument by considering an *argument* (or set of hypotheses) as a set S of abducible literals of a NLP P.

We have seen before examples of Extended Logic Programs — with explicit negation. In [18] the authors show that a simple syntactical program transformation applied to an ELP produces a Normal Logic Program with Integrity Constraints which has the exact same semantics as the original ELP.

Example 9. Transforming an ELP into a NLP with ICs
Taking the program

$$\begin{array}{l} dangerous\_neighborhood \leftarrow not\ \neg dangerous\_neighborhood \\ \neg dangerous\_neighborhood \leftarrow not\ dangerous\_neighborhood \end{array}$$

we just transform the explicitly negated literal $\neg dangerous\_neighborhood$ into the positive literal $dangerous\_neighborhood^n$, and the original $dangerous\_neighborhood$ literal is converted into $dangerous\_neighborhood^p$

Now, in order to ensure consistency, we just need to add the IC $\bot \leftarrow dangerous\_neighborhood^p, dangerous\_neighborhood^n$. The resulting transformed program is

$$\begin{array}{l} dangerous\_neighborhood^p \leftarrow not\ dangerous\_neighborhood^n \\ dangerous\_neighborhood^n \leftarrow not\ dangerous\_neighborhood^p \\ \bot \leftarrow dangerous\_neighborhood^p, dangerous\_neighborhood^n \end{array}$$

Now know that we can just consider NLPs with ICs without loss of generality and so, henceforth, we will assume just that case. NLPs are in fact the kind of programs most Inductive Logic Programming learning systems produce.

1.4.2 Assumptions and Argumentation

Previously, we have seen that assumptions can be coded as abducible literals in Logic Programs and that those abducibles can be packed together in chromosomes. The evolutionary operators of genetic and memetic crossover, mutation and fitness function applied to the chromosomes provide a means to search for a consensus of the initial assumptions since it will be a consistent mixture of these.

Moreover, the 2-valued contradiction removal method presented in subsection 1.2.2 is a very superficial one. That method removes the contradiction between $p(\mathbf{X})$ and $\neg p(\mathbf{X})$ by forcing a 2-valued semantics for the ELP to choose either $p(\mathbf{X})$ or $\neg p(\mathbf{X})$ since they now are exceptions to one another. It is a superficial removal of the contradiction because the method does not look into the reasons why both $p(\mathbf{X})$ and $\neg p(\mathbf{X})$ hold simultaneously. The method does not go back to find the underlying assumptions supporting both $p(\mathbf{X})$ and $\neg p(\mathbf{X})$ to find out which assumptions should be revised in order to restore overall consistency. Any one such method must fall back into the principles of argumentation: to find the arguments supporting one conclusion in order to prevent it if it leads to contradiction.

One such 'deeper' method for contradiction removal is presented in [46]. In this chapter we have presented another alternative method inspired by evolution.

1.4.3 Collaborative Opposition

In [28] the author shows that the Stable Models of a NLP coincide with the 2-valued complete Preferred Extensions which are self-corroborating arguments. However, as it is well known, not all NLPs have Stable Models. In fact, [31] showed that the NLPs with Odd Loops Over Negation (OLONs) [6] are the one who might have no Stable Models. It is always possible to argue that when an ILP system is building the NLP it can detect if there are such OLONs and do something about them. However, in a distributed knowledge environment, e.g. a Semantic Web, several NLPs can be produced by several ILP systems and the NLPs may refer to the same literals. There is a reasonable risk that when merging the NLPs together OLONs might appear which were not present in each NLP separately. The works in [62], [63], [64], and [65] show how to solve OLONs.

The more challenging environment of a Semantic Web is one possible 'place' where the future intelligent systems will live in. Learning in 2-values or in 3-values are open possibilities, but what is most important is that knowledge and reasoning will be shared and distributed. Different opposing concepts and arguments will come from different agents. It is necessary to know how

[6] An OLON is just a loop or cycle in the program's dependency graph for some literal where the number of default negations along the loop is odd.

to conciliate those opposing arguments, and how to find 2-valued consensus as much as possible instead of just keeping to the least-commitment 3-valued consensus. In [64] the authors describe another method for finding such 2-valued consensus in an incremental way. In a nutshell, we start by merging together all the opposing arguments into a single one. The conclusions from the theory plus the unique merged argument are drawn and, if there are contradictions against the argument or contradictions inside the argument we non-deterministically choose one contradicted assumption of the argument and revise its truth value. The iterative repetition of this step eventually ends up in a non-contradictory argument (and all possibilities are explored because there is a non-deterministic choice).

In a way, the evolutionary method we presented in subsection 1.3.4 implements a similar mechanism to find the consensus non-contradictory arguments.

1.5 Conclusions

The two-valued setting that has been adopted in most work on ILP and Inductive Concept Learning in general is not sufficient in many cases where we need to represent real world data. This is for example the case of an agent that has to learn the effect of the actions it can perform on the domain by performing experiments. Such an agent needs to learn a definition for allowed actions, forbidden actions and actions with an unknown outcome, and therefore it needs to learn in a richer three-valued setting.

The programs that are learnt will contain a definition for the concept and its opposite, where the opposite concept is expressed by means of explicit negation. Standard ILP techniques can be adopted to separately learn the definitions for the concept and its opposite. Depending on the adopted technique, one can learn the most general or the least general definition.

The two definitions learned may overlap and the inconsistency is resolved in a different way for atoms in the training set and for unseen atoms: atoms in the training set are considered exceptions, while unseen atoms are considered unknown. The different behavior is obtained by employing negation by default in the definitions: default abnormality literals are used in order to consider exceptions to rules, while non-deterministic rules are used in order to obtain an unknown value for unseen atoms.

We have also presented an evolution-inspired algorithm for performing belief revision in a multi-agent environment. The standard genetic algorithm is extended in two ways: first the algorithm combines two different evolution strategies, one based on Darwin's and the other on Lamarck's evolutionary theory and, second, chromosomes from different agents can be crossed over with each other. The Lamarckian evolution strategy is obtained be means of an operator that changes the genes (or, better, the memes) of an agent in order to improve their fitness.

Lamarckian and Darwinian operators have complimentary functions: Lamarckian operators are used to get closer to a solution of a given belief revision problem, while Darwinian operators are used in order to distribute the acquired knowledge amongst agents. The contradictions that may arise between chromosomes of memes from different agents are of a distinct nature from the contradictions arising from the learning process. The former correspond to different alternative explanations to the observations, whereas the latter correspond to uncertainties in the learned concepts.

Moreover, we can also bring the same evolutionary algorithm to a single agent's mind. In such case, we can think of the agent's mind as a pool of several sub-agents, each considering one aspect of the environment, each acting as a specialist in some sub-domain.

We have presented too a new and productive way to deal with oppositional concepts in a collaboration perspective, in different degrees. We use the contradictions arising from opposing arguments as hints for the possible collaborations. In so doing, we extend the classical conflictual argumentation giving a new treatment and new semantics to deal with the contradictions.

The direction of our future work includes three main axis: continuing theory development, implementing prototypes and exploring possible applications.

Most recently, we have been exploring the constructive negation [51] reasoning mechanism. In a nutshell, constructive negation concerns getting answers to queries by imposing inequality constraints on the values of variables (e.g., getting an answer of the form "all birds fly, except for the penguins"). Such mechanism is particularly well suited for the integration of answers coming from different agents, e.g., one agent can learn the general rule that "all birds fly", and another might learn only the exceptional case that "penguins do not fly", and that "penguins are birds". Constructive negation can play a synergistic rôle in this matter by gracefully merging the knowledge of the different agents into a single consistent, integrated, more specialized and refined one.

Our future efforts will therefore engage in bringing together the three branches we described — learning, belief evolution, and argumentation — under the scope of constructive negation. Besides the theoretical research we have been doing, and will continue to do in this area, we have already under way a practical implementation of this constructive negation reasoning mechanism [58] on top of XSB Prolog [71].

Also, one application field for all this work is Ambient Intelligence [14]. In a nutshell, Ambient Intelligence concerns intelligent software agents embedded in real-world environments and that are sensitive and responsive to the presence of people. Such environments are constantly changing as different people come in and out of play, each of which may influence the rest of the environment.

We envisage a framework for Ambient Intelligence where agents interact with users

(i) with the aim of monitoring them for ensuring some degree of consistence and coherence in user behavior and, possibly,

(ii) with the objective of training them in some particular task.

In our view, a system which is a realization of the envisaged framework will bring to a user the following potential advantages: the user is relieved of some of the responsibilities related to her behavior, as directions about the "right thing" to do are constantly and punctually provided. She is assisted in situations where she perceived herself as inadequate, in some respect, to perform her activities or tasks. She is possibly told how to cope with unknown, unwanted or challenging circumstances. She interacts with a "Personal Assistant" that improves in time, both in its "comprehension" of the user needs, cultural level, preferred kinds of explanations, etc. and in its ability to cope with the environment.

References

1. Alferes, J. J., Brogi, A., Leite, J. A., and Pereira, L. M. Evolving logic programs. In S. Flesca et al., editor, *JELIA*, volume 2424 of *LNCS*, pages 50–61. Springer, 2002.
2. Alferes, J. J., Damásio, C. V., and Pereira, L. M. (1994). SLX - A top-down derivation procedure for programs with explicit negation. In Bruynooghe, M., editor, *Proc. Int. Symp. on Logic Programming*. The MIT Press.
3. Alferes, J.J., and Pereira, L. M. http://xsb.sourceforge.net/manual2/node179.html
4. Alferes, J. J., and Pereira, L. M. An argumentation theoretic semantics based on non-refutable falsity. In J. Dix et al., editor, *NMELP*, pages 3–22. Springer, 1994.
5. Alferes, J. J. and Pereira, L. M. (1996). *Reasoning with Logic Programming*, volume 1111 of *LNAI*. Springer-Verlag.
6. Alferes, J. J., Pereira, L. M., and Przymusinski, T. C. (1998). "Classical" negation in non-monotonic reasoning and logic programming. *Journal of Automated Reasoning*, 20:107–142.
7. Bain, M. and Muggleton, S. (1992). Non-monotonic learning. In Muggleton, S., editor, *Inductive Logic Programming*, pages 145–161. Academic Press.
8. http://www.psych.utoronto.ca/museum/baldwin.htm
9. Baral, C., Gelfond, M., and Rushton, J. Nelson. Probabilistic reasoning with answer sets. In Vladimir Lifschitz and Ilkka Niemelä, editors, *LPNMR*, volume 2923 of *Lecture Notes in Computer Science*, pages 21–33. Springer, 2004.
10. Baral, C. and Gelfond, M. (1994). Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148.
11. Baral, C., and Subrahmanian, V. S. Dualities between alternative semantics for logic programming and nonmonotonic reasoning. *J. Autom. Reasoning*, 10(3):399–420, 1993.
12. Bondarenko, A., Dung, P. M., Kowalski, R. A., and Toni, F. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.*, 93:63–101, 1997.

13. Chan, P. and Stolfo, S. (1993). Meta-learning for multistrategy and parallel learning. In *Proceedings of the 2nd International Workshop on Multistrategy Learning*, pages 150–165.
14. Costantini S., Dell'Acqua, P., Pereira, L. M., and Toni, F., (2007) Towards a Model of Evolving Agents for Ambient Intelligence. In Sadri, F., and Stathis, K., editors, *Procs. Symposium on Artificial Societies for Ambient Intelligence (ASAmI'07)*. Extended version submited to a Journal.
15. Damásio, C. V., Nejdl, W., and Pereira, L. M. (1994). REVISE: An extended logic programming system for revising knowledge bases. In Doyle, J., Sandewall, E., and Torasso, P., editors, *Knowledge Representation and Reasoning*, pages 607–618. Morgan Kaufmann.
16. Damásio, C. V. and Pereira, L. M. (1997). Abduction on 3-valued extended logic programs. In Marek, V. W., Nerode, A., and Trusczynski, M., editors, *Logic Programming and Non-Monotonic Reasoning - Proc. of 3rd International Conference LPNMR'95*, volume 925 of *LNAI*, pages 29–42, Germany. Springer-Verlag.
17. Damásio, C. V. and Pereira, L. M. (1998). A survey on paraconsistent semantics for extended logic programs. In Gabbay, D. and Smets, P., editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 2, pages 241–320. Kluwer Academic Publishers.
18. Damásio, C. V. and Pereira, L. M. Default Negated Conclusions: Why Not?. In *ELP'96*, pages 103–117. Springer, 1996
19. De Raedt, L. (1992). *Interactive Theory Revision: An Inductive Logic Programming Approach*. Academic Press.
20. De Raedt, L., Bleken, E., Coget, V., Ghil, C., Swennen, B., and Bruynooghe, M. (1993). Learning to survive. In *Proceedings of the 2nd International Workshop on Multistrategy Learning*, pages 92–106.
21. De Raedt, L. and Bruynooghe, M. (1989). Towards friendly concept-learners. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 849–856. Morgan Kaufmann.
22. De Raedt, L. and Bruynooghe, M. (1990). On negation and three-valued logic in interactive concept learning. In *Proceedings of the 9th European Conference on Artificial Intelligence*.
23. De Raedt, L. and Bruynooghe, M. (1992). Interactive concept learning and constructive induction by analogy. *Machine Learning*, 8(2):107–150.
24. Dix, J. A Classification-Theory of Semantics of Normal Logic Programs: I, II. *Fundamenta Informaticae*, XXII(3):227–255, 257–288, 1995.
25. Dix, J., Pereira, L. M., and Przymusinski, T. (1997). Prolegomena to logic programming and non-monotonic reasoning. In Dix, J., Pereira, L. M., and Przymusinski, T., editors, *Non-Monotonic Extensions of Logic Programming - Selected papers from NMELP'96*, number 1216 in LNAI, pages 1–36, Germany. Springer-Verlag.
26. Drobnic, M. and Gams, M. (1993). Multistrategy learning: An analytical approach. In *Proceedings of the 2nd International Workshop on Multistrategy Learning*, pages 31–41.
27. Džeroski, S. (1991). Handling noise in inductive logic programming. Master's thesis, Faculty of Electrical Engineering and Computer Science, University of Ljubljana.
28. Dung, P. M. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.*, 77(2):321–358, 1995.

29. Dung, P. M., Kowalski, R. A., and Toni, F. Dialectic proof procedures for assumption-based, admissible argumentation. *Artif. Intell.*, 170(2):114–159, 2006.
30. Esposito, F., Ferilli, S., Lamma, E., Mello, P., Milano, M., Riguzzi, F., and Semeraro, G. (1998). Cooperation of abduction and induction in logic programming. In Flach, P. A. and Kakas, A. C., editors, *Abductive and Inductive Reasoning*, Pure and Applied Logic. Kluwer.
31. Fages, F. Consistency of Clark's completion and existence of stable models. *Methods of Logic in Computer Science*, 1:51–60, 1994.
32. Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. and Bowen, K. A., editors, *Proceedings of the 5th Int. Conf. on Logic Programming*, pages 1070–1080. MIT Press.
33. Gelfond, M. and Lifschitz, V. (1990). Logic programs with classical negation. In *Proceedings of the 7th International Conference on Logic Programming ICLP90*, pages 579–597. The MIT Press.
34. Gordon, D. and Perlis, D. (1989). Explicitly biased generalization. *Computational Intelligence*, 5(2):67–81.
35. Green, D.M., and Swets, J.M. (1966). Signal detection theory and psychophysics. New York: John Wiley and Sons Inc.. ISBN 0-471-32420-5.
36. Greiner, R., Grove, A. J., and Roth, D. (1996). Learning active classifiers. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML96)*.
37. Inoue, K. (1998). Learning abductive and nonmonotonic logic programs. In Flach, P. A. and Kakas, A. C., editors, *Abductive and Inductive Reasoning*, Pure and Applied Logic. Kluwer.
38. Inoue, K. and Kudoh, Y. (1997). Learning extended logic programs. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 176–181. Morgan Kaufmann.
39. Jenkins, W. (1993). Intelog: A framework for multistrategy learning. In *Proceedings of the 2nd International Workshop on Multistrategy Learning*, pages 58–65.
40. Kakas, A. C. and Mancarella, P. Negation as stable hypotheses. In *LPNMR*, pages 275–288. MIT Press, 1991.
41. Koza, J.R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection MIT Press
42. Jean Baptiste Lamarck: http://www.ucmp.berkeley.edu/history/lamarck.html
43. Grefenstette, J. J., (1991). Lamarckian learning in multi-agent environments
44. Lamma, E., Riguzzi, F., and Pereira, L. M. (1988). Learning in a three-valued setting. In *Proceedings of the Fourth International Workshop on Multistrategy Learning*.
45. Lamma, E., Riguzzi, F., and Pereira, L. M. (1999a). Agents learning in a three-valued setting. Technical report, DEIS - University of Bologna.
46. Lamma, E., Pereira, L. M., and Riguzzi, F. Belief revision via lamarckian evolution. *New Generation Computing*, 21(3):247–275, August 2003.
47. Lamma, E., Riguzzi, F., and Pereira, L. M. Strategies in combined learning via logic programs. *Machine Learning*, 38(1-2):63–87, January 2000.
48. Lapointe, S. and Matwin, S. (1992). Sub-unification: A tool for efficient induction of recursive programs. In Sleeman, D. and Edwards, P., editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 273–281. Morgan Kaufmann.
49. Lavrač, N. and Džeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

50. Leite, J. A. and Pereira, L. M. (1998). Generalizing updates: from models to programs. In Dix, J., Pereira, L. M., and Przymusinski, T. C., editors, *Collected Papers from Workshop on Logic Programming and Knowledge Representation LPKR'97*, number 1471 in LNAI. Springer-Verlag.
51. Liu, J. Y., Adams, L., and Chen, W. Constructive Negation Under the Well-Founded Semantics J. Log. Program., volume 38, 3: 295-330, 1999.
52. Malý, M. Complexity of revised stable models. Master's thesis, Comenius University Bratislava, 2006.
53. Michalski, R. (1973). Discovery classification rules using variable-valued logic system VL1. In *Proceedings of the Third International Conference on Artificial Intelligence*, pages 162–172. Stanford University.
54. Michalski, R. (1984). A theory and methodology of inductive learning. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning - An Artificial Intelligence Approach*, volume 1, pages 83–134. Springer-Verlag.
55. Muggleton, S. (1995). Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286.
56. Muggleton, S. and Buntine, W. (1992). Machine invention of first-order predicates by inverting resolution. In Muggleton, S., editor, *Inductive Logic Programming*, pages 261–280. Academic Press.
57. Muggleton, S. and Feng, C. (1990). Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan.
58. http://centria.di.fct.unl.pt/∼lmp/software/contrNeg.rar
59. Pazzani, M. J., Merz, C., Murphy, P., Ali, K., Hume, T., and Brunk, C. (1994). Reducing misclassification costs. In *Proceedings of the Eleventh International Conference on Machine Learning (ML94)*, pages 217–225.
60. http://www.peirce.org/
61. Pereira, L. M. and Alferes, J. J. (1992). Well founded semantics for logic programs with explicit negation. In *Proceedings of the European Conference on Artificial Intelligenece ECAI92*, pages 102–106. John Wiley and Sons.
62. Pereira, L. M. and Pinto, A. M. Revised stable models - a semantics for logic programs. In G. Dias et al., editor, *Progress in AI*, volume 3808 of *LNCS*, pages 29–42. Springer, 2005.
63. Pereira, L. M. and Pinto, A. M. Reductio ad absurdum argumentation in normal logic programs. In *Argumentation and Non-monotonic Reasoning (ArgNMR'07) workshop at LPNMR'07*, pages 96–113, 2007.
64. Pereira, L. M. and Pinto, A. M. Approved Models for Normal Logic Programs. In *LPAR*, pages 454–468. Springer, 2007.
65. Pinto, A. M. Explorations in revised stable models — a new semantics for logic programs. Master's thesis, Universidade Nova de Lisboa, February 2005.
66. Plotkin, G. (1970). A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press.
67. Provost, F. J. and Fawcett, T. (1997). Analysis and visualization of classifier performance: Comparison under imprecise class and cost distribution. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD97)*. AAAI Press.
68. Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5:239–266.
69. Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA.

70. Reiter, R. (1978). On closed-word data bases. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 55–76. Plenum Press.
71. Sagonas, K. F., Swift, T., Warren, D. S., Freire, J., and Rao, P. (1997). *The XSB Programmer's Manual Version 1.7.1.*
72. Soares, L. Revising undefinedness in the well-founded semantics of logic programs. Master's thesis, Universidade Nova de Lisboa, 2006.
73. Van Gelder, A., Ross, K. A., and Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650.
74. Vere, S. A. (1975). Induction of concepts in the predicate calculus. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI75)*, pages 281–287.
75. Whitley, D., Rana, S., and Heckendorn, R.B. (1998) The Island Model Genetic Algorithm: On Separability, Population Size and Convergence

A Appendix

The definition of WFSX that follows is taken from [2] and is based on the alternating fix points of Gelfond-Lifschitz Γ-like operators.

Definition 3. ***The Γ-operator.*** *Let P be an extended logic program and let I be an interpretation of P . $\Gamma_P(I)$ is the program obtained from P by performing in the sequence the following four operations:*

- *Remove from P all rules containing a default literal $L = not\ A$ such that $A \in I$.*
- *Remove from P all rules containing in the body an objective literal L such that $\neg L \in I$.*
- *Remove from all remaining rules of P their default literals $L = not\ A$ such that $not\ A \in I$.*
- *Replace all the remaining default literals by proposition u.*

In order to impose the coherence requirement, we need the following definition.

Definition 4. ***Seminormal Version of a Program.***

The seminormal version of a program P is the program P_s obtained from P by adding to the (possibly empty) Body of each rule $L \leftarrow$ Body the default literal $not\ \neg L$, where $\neg L$ is the complement of L with respect to explicit negation.

In the following, we will use the following abbreviations: $\Gamma(S)$ for $\Gamma_P(S)$ and $\Gamma_s(S)$ for $\Gamma_{P_s}(S)$.

Definition 5. ***Partial Stable Model.***

An interpretation $T \cup not\ F$ is called a partial stable model *of P iff $T = \Gamma\Gamma_s T$ and $F = H^E(P) - \Gamma_s T$.*

Partial stable models are an extension of stable models *[32] for extended logic programs and a three-valued semantics. Not all programs have a partial*

stable model (e.g., $P = \{a, \neg a\}$) and programs without a partial stable model are called contradictory*.*

Theorem 1. ***WFSX Semantics.***

Every non-contradictory program P has a least (with respect to $\subseteq$) partial stable model, the well-founded model of P denoted by $WFM(P)$.

Proof. To obtain an iterative "bottom-up" definition for $WFM(P)$ we define the following transfinite sequence $\{I_\alpha\}$:

$$I_0 = \{\}; \quad I_{\alpha+1} = \Gamma\Gamma_S I_\alpha \ ; \quad I_\delta = \bigcup\{I_\alpha | \alpha < \delta\}$$

where δ is a limit ordinal. There exists a smallest ordinal λ for the sequence above, such that I_λ is the smallest fix point of $\Gamma\Gamma_S$. Then, $WFM(P) = I_\lambda \cup not\ (H^E(P) - \Gamma_S I_\lambda)$. $\square$