Strategies for Learning with Extended Logic Programs

Evelina Lamma, Fabrizio Riguzzi
DEIS, Università di Bologna, Viale Risorgimento 2
I-40136 Bologna, Italy,
Tel. +39 51 6443033, Fax. +39 51 6443073
{elamma,friguzzi}@deis.unibo.it

Luís Moniz Pereira
Centro de Inteligência Artificial (CENTRIA),
Tel. +351 1 294 8533, Fax. +351 1 294 8541
Departamento de Informática, Universidade Nova de Lisboa,
P-2825 Monte da Caparica, Portugal
lmp@di.fct.unl.pt

Abstract

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 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. For this purpose, we adopt extended logic programs under a well-founded semantics as the representation formalism for learning.

Explicit negation is used in order to represent the opposite concept, while default negation is used to ensure consistency and to handle exceptions to general rules. Exceptions to a concept are identified from examples of the opposite training set. Exceptions can be generalized, in their turn, by learning within a hierarchy of defaults.

Standard ILP techniques are used to learn the concept and its opposite. Depending on the technique adopted, we can learn the most general or the least general definition. Thus, four epistemological varieties occur, resulting from the combination of most general and least general solutions for the positive and negative concept. We discuss the factors that should be taken into account when choosing and strategically combining the level of generality.

1 Introduction

Most work on inductive concept learning considers a two-valued setting. In such a setting, what is not entailed by the learned theory is considered false, on the basis of the Closed World Assumption (CWA) [25]. However in practice, it is more often the case that we know with certainty the truth or falsity of only a limited number of facts, and are not able to draw any conclusion about the remaining ones, because the available information is too scarce. As it has been pointed out in [6], this is typically the case of an autonomous agent that gathers information incrementally from its surrounding world. Such an agent needs to distinguish between what is true, what is false and what is unknown, and therefore it needs to learn within a three-valued setting.

For this purpose, we adopt the class of *Extended Logic Programs* (ELP, for short, in the sequel) as the representation language for learning. ELP are logic programs with default negation, plus a second form of negation, called *explicit*, which have been recognized as a very expressive means of representation, in particular when used in conjunction. The adoption of ELP as a representation formalism allows one to deal directly with incomplete knowledge, by expressing *unknown* information through default negation as well as *negative* information through the second form of negation [1]. For instance, in [22, 1] it is shown how ELP are applicable to such diverse domains of knowledge representation as concept hierarchies, reasoning about actions, counterfactuals, diagnosis, and debugging.

In this work, we discuss various approaches or strategies that can be adopted in Inductive Logic Programming (ILP, henceforth) for learning ELP. As in [13, 12], the learning process starts from a set of positive and negative examples plus some background knowledge in the form of a logic program extended with explicit negation. 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 interpreter for extended logic programs under well founded semantics, defined in [2, 1]. Note that the learning of a concept and its explicit negation version can be carried out in parallel. This is not possible with the CWA rendering of negative concepts.

Default negation is used in the learning process to handle *exceptions* to general rules. Exceptions to a positive concept are identified from negative examples, whereas exceptions to a negative concept are identified from positive examples.

Furthermore, in order to handle *contradiction*, and avoid it whenever possible, learned rules are then made non-deterministic (see [21, 5]) by making the learned definition for a positive concept p depend on the default negation of the negative concept $\neg p$, and vice-versa.

Standard ILP techniques are used to learn some concept and its opposite. Depending on the technique adopted, one can learn the most general or the least general definition, or both. Thus, four epistemological varieties occur, resulting from the mutual combination of most general and least general solutions for the positive and negative concept. We discuss some of the factors that should be taken into account when choosing the level of generality of each, and their combination, to define a specific learning strategy.

The paper is organized as follows: in Section 2 we further motivate the need for learning ELP under well-founded semantics, and formally introduce the ILP framework. Section 3 discusses how to avoid contradiction and deal with exceptions. Section 4 relates the strategies for learning positive and negative concepts to the risk of damage in case of an inadequate level of learning generality. Related work is discussed in Section 5. Section 6 summarizes some conclusions and mentions directions for future work.

2 Learning in a Three-valued Setting

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. For example, consider an autonomous agent that gathers information incrementally from its surrounding world. The agent has to choose its actions on the basis of the knowledge it possesses. Therefore it needs to distinguish between actions that certainly lead to a failure from actions that have an unknown outcome. The agent will never try an action when it is sure of its negative effect, but it may try an action with an unknown outcome in situations where no other action can be taken, or in order to expand its knowledge.

Therefore, such an agent needs to be able to distinguish between what is certainly true, what is certainly false and what is unknown. For this purpose, it should adopt a three-valued truth-value setting, and learn a definitions for both a concept and its complement, using positive examples for the concept as negative examples for its complement and viceversa. The so learned theory will then classify instances in three ways: instances covered by the positive definition are positive, instances covered by the negative definition are negative and instances not covered by any definition, or covered by both, are unknown.

This requires the adoption of a more expressible class of programs to be learned, where some atoms are true, some false and some others are unknown or undefined. This class can be represented by means of Extended Logic Programs, either under a stable semantics [11] or under a well-founded one [20]. An extended logic program is a set of rules of the form

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_{m+n}$$

where $m, n \geq 0$ and each l_i $(i = 1, \ldots, m+n)$ is an *objective* literal of the form a or $\neg a$, where a is an atom. Thus, from now on we will distinguish between negation by default (or *implicit*) negation, denoted by *not*, and a second form of (*explicit*) negation, denoted by $\neg$. $\neg A$ is said the *opposite* literal of A (and viceversa) and *not* A the *complementary* literal of A (and viceversa). The Coherence principle [20] relates the two negations by stipulating that $\neg L$ entails $notL$ for any objective L.

In the following, we will adopt the well-founded semantics with explicit negation *WFSX* [20] because none of the other semantics for ELP, answer-sets and 3-valued strong negation, enjoy the property of relevance [1, 3], i.e., they cannot have top-down querying procedures. Instead, for *WFSX* there exists the top-down proof procedure *SLX* [2, 1], which is correct with respect to the semantics as per [1].

Starting from a set of positive and negative examples for a target predicate p, and a background knowledge which is itself an extended logic program under *WFSX*, we apply standard ILP techniques in order to learn a definition for both a positive concept p and its opposite $\neg p$. The ILP learning problem for the case of ELP was first introduced in [13] and is given as follows.

Given:

- a set $\mathcal{P}$ of possible (extended logic) programs
- a set E^+ of positive examples
- a set E^- of negative examples
- an extended logic program B (*background knowledge*)

Find:

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

  where $\neg E = \{\neg e | e \in E\}$, and $E^+, \neg E^-$ (resp. $E^-, \neg E^+$) stands for the conjunction of each atom e in E^+ and in $\neg E^-$ (resp. in E^- and $\neg E^+$).

Note that, in the case of ELP, 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 is non-contradictory in a more general sense, i.e. $B \cup P \not\models L, \neg L$ for no L.

Our approach to learning ELP consists in applying conventional ILP techniques to learn the positive definition and the negative one too. The ILP technique to be used depends on the level of generality that we want to have for the two definitions: we can look for the Least General Solution (LGS for short) or the Most General Solution (MGS for short) of the problem of learning each concept. The generality of the solutions should be chosen independently for the two definitions, thus leading to four epistemological cases, that will be discussed in section 4.

LGSs can be found, for example, by adopting a bottom-up method such as Relative Least General Generalization (RLGG) [23], Inverse Resolution [18] or Inverse Entailment [16]. Instead, MGSs can be found by adopting a top-down system such as FOIL [24] or Progol [17].

In order to illustrate the difference between the various generalizations, consider the following example.

Example 2.1 *The domain contains four entities a, b, c, d and the target concept is fly.*

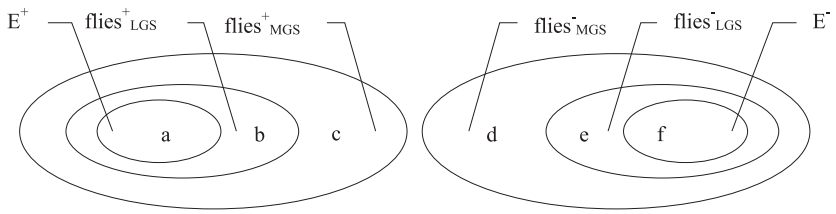

Figure 1: Coverage of most general and least general definitions for the positive and negative concepts

$bird(a).$ $\quad has_wings(a).$
$bird(b).$ $\quad has_wings(b).$
$jet(c).$ $\quad has_wings(c).$
$dog(d).$ $\quad has_limbs(d).$
$cat(e).$ $\quad has_limbs(e).$
$cat(f).$ $\quad has_limbs(f).$

The training set is:

$E^+ = \{flies(a)\}$
$E^- = \{flies(f)\}$

Below are reported the most general and least general definitions, for both the positive and negative concepts, that constitute a solution to the learning problem:

$flies^+_{MGS}(X) \leftarrow has_wings(X).$
$flies^+_{LGS}(X) \leftarrow has_wings(X), bird(X).$
$flies^-_{MGS}(X) \leftarrow has_limbs(X).$
$flies^-_{LGS}(X) \leftarrow has_limbs(X), cat(X).$

The example is depicted in figure 1.

3 Eliminating Learned Contradictions

The definitions of the positive and negative concepts may overlap. In this case, we have a contradictory classification for the atoms in the intersection. In order to resolve the conflict, we must distinguish two types of atoms in the intersection: those that also belong to the training set and those that don't, the latter of which we dub *unseen* atoms (see figure 2).

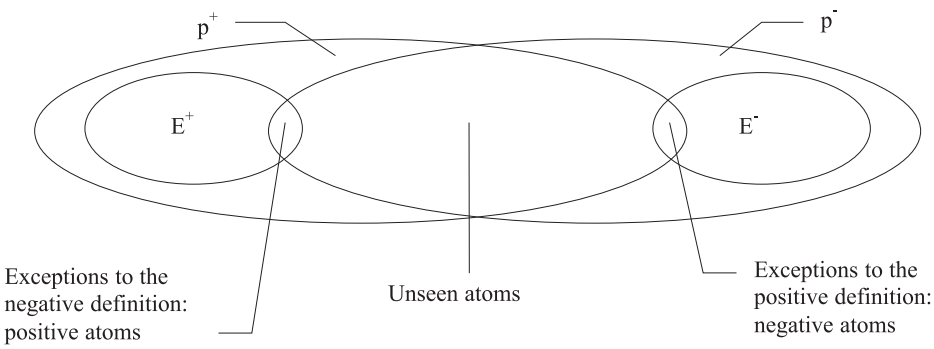

Figure 2: Interaction of the positive and negative definitions on exceptions.

For unseen atoms, the conflict should be resolved by classifying the atoms as unknown, since the arguments supporting the two classifications are equally strong. Instead, for atoms in the training set, the conflict should be resolved by giving priority to the classification stipulated by the training set, supposing this information is reliable, i.e., under the hypothesis of the absence of noise. In other words, atoms in a training set that are covered by the opposite definition should be considered as *exceptions* to that definition.

For unseen atoms in the intersection, the unknown classification is obtained by making the rules non deterministic (see [21, 5, 1]). The target theory is thus expressed in the following way:

$$\begin{aligned} p(\vec{X}) &\leftarrow p^+(\vec{X}), not\ \neg p(\vec{X}) \\ \neg p(\vec{X}) &\leftarrow p^-(\vec{X}), not\ p(\vec{X}) \end{aligned}$$

where $p^+(\vec{X})$ and $p^-(\vec{X})$ are, respectively, the definitions learned for the positive and the negative concept, and $\vec{X}$ stands for a tuple of arguments. For each atom in the intersection, there are two partial stable models, one containing the atom in its positive version, the other containing the opposite literal. According to the well-founded semantics for explicit negation [20] (i.e., in the least partial stable model) the atom is unknown[1].

[1]Mark that one could now relax the condition that the training sets be disjoint, because any contradiction is blocked by these rules

Let us consider now the case in which some atoms in the intersection belong to the opposite training set. We want to classify these atoms according to the classification given by the training set pertaining to the concept in question, such atoms representing *exceptions*. To handle exceptions to classification rules, we add a negative literal (using negation as default) of the form $not\ abnorm_{p^+}(\vec{X})$ (resp. $not\ abnorm_{p^-}(\vec{X})$) to the rule for p (resp. $\neg p$), expressing the default condition. Then, for every exception $p(\vec{t})$, an individual fact of the form $abnorm_{p^+}(\vec{t})$ (resp. $abnorm_{p^-}(\vec{t})$) is asserted so that the rule for p (resp. $\neg p$) will not cover the exception. In this way, exceptions will figure in the model of the theory with the correct truth value. The learned theory thus takes the following form:

$$\begin{aligned} p(\vec{X}) &\leftarrow p^+(\vec{X}), not\ abnorm_{p^+}(\vec{X}), not\ \neg p(\vec{X}) \\ \neg p(\vec{X}) &\leftarrow p^-(\vec{X}), not\ abnorm_{p^-}(\vec{X}), not\ p(\vec{X}) \end{aligned}$$

The individual facts of the form $abnorm_{p^+}(\vec{t})$ can then be used as examples for learning a definition for $abnorm_{p^+}$, as in [13, 15].

In this way, we are able to deal both with the case in which the two definitions are contradictory and with the case in which exceptions to rules exist. Consider again example 2.1 and suppose the learned theory is

$$\begin{aligned} &flies(X) \leftarrow flies^+_{MGS}(X), not\ abnorm_{flies^+}(X), not\ \neg flies(X). \\ &\neg flies(X) \leftarrow flies^-_{MGS}(X), not\ abnorm_{flies^-}(X), not\ flies(X). \end{aligned}$$

where $flies^+_{MGS}(X)$ and $flies^-_{MGS}(X)$ are:

$$\begin{aligned} flies^+_{MGS}(X) &\leftarrow has_wings(X). \\ flies^-_{MGS}(X) &\leftarrow has_limbs(X). \end{aligned}$$

Now, consider two new instances g and h. For the instance g, the following facts are true

$$angel(g). \qquad has_wings(g). \qquad has_limbs(g).$$

$flies(g)$ is an unseen atom which, however, is covered by both $flies^{+}_{MGS}(X)$ and $flies^{-}_{MGS}(X)$. According to the learned theory, $flies(g)$ is unknown.

As regards the instance h, suppose that the training set E^{-} contains also the atom $flies(h)$ and that the background contains

$$penguin(h). \qquad has_wings(h). \qquad has_limbs(h).$$

$flies(h)$ is covered by $flies^{+}_{MGS}(X)$ and therefore it is an exception. During the learning process, the atom $abnorm_{flies^{+}}(h)$ is asserted into the theory, and therefore the literal $not\ abnorm_{flies^{+}}(h)$ fails and $flies(h)$ is not derivable. On the contrary a proof for $\neg flies(h)$ succeeds.

Up to now we have simply admitted the possibility that a definition covers atoms of the opposite training set, but we have not yet explained how this could happen. In fact, both the top-down and bottom-up algorithms that we have described produce only consistent clauses, or clauses that do not cover any atom of the opposite training set. Therefore, exceptions are normally not present. However, in some cases, it is useful to relax the consistency requirement and to learn clauses that cover a small amount of negative examples. This happens when it would be otherwise impossible to learn a definition for the concept, because no clause in the language bias is consistent, or when an overspecific definition would be learned, composed of very many specific clauses instead of a few general ones. In these cases, the consistency requirement can be relaxed by allowing clauses that cover a small fraction of negative training examples.

Exceptions may be due to noise in the collection of data or to abnormalities in the opposite concept. In the second case, if exceptions form a class, it may be possible to learn a definition for it, provided that we have data on their common properties and the language bias allows such a definition.

4 Strategically Combining Different Generalizations

The generality of the solutions 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, since it is the complement of the true atoms set. Instead,

in a three-valued setting, the extension of the set of false atoms depends on the generality of the definition for the negative concept. Therefore, the level of generality must be chosen independently for both definitions, thus obtaining four possible combinations.

Moreover, the generality of the solutions learned for the positive and negative concept clearly influences the interaction between the definitions. For example, if we learn the MGS for both the concept and its opposite, the probability that their intersection is non-empty is higher than if we learn the LGS for both definitions. Therefore, the decision on which type of solution to learn should take into account as well the possibility of interaction: if we want to reduce this possibility, we have to learn two LGSs, if we do not care about interaction, we can learn two MGSs. In general, we may learn diverse generalizations and combine them in distinct ways for different strategic purposes within one same application problem.

In order to chose the desired generality of the positive and negative definitions one of the factors that should be taken into account is the risk of damage that can arise from an erroneous classification of an unseen object. Depending on how high is this risk, one may choose to take a cautious or confident approach to learning. In the following we present an example of the kind of reasoning that can be used to chose the level of generality, and we discuss how to strategically combine levels of generalization by employing the extended logic programming approach to learning set forth in this paper.

Consider a person living in a bad neighbourhood 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, band membership, age, past dealings, etc. Because he wants to take a cautious approach, he maximizes $attacker^+$ and minimizes $attacker^-$, so that his $attacker_1$ concept allows him to avoid dangerous situations.

$attacker_1(X) \leftarrow attacker^+_{MGS}(X), not\ \neg attacker_1(X).$

$\neg attacker_1(X) \leftarrow attacker^-_{LGS}(X), not\ attacker_1(X).$

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, children, etc. Because he is not rich and also he doesn't like to be tricked, he learns a $beggar_1$ concept by minimizing $beggar^+$ and maximizing

$beggar^-$, so that his beggar concept allows him to give money strictly to those appearing to need it without faking.

$beggar_1(X) \leftarrow beggar^+_{LGS}(X), not\ \neg beggar_1(X).$

$\neg beggar_1(X) \leftarrow beggar^-_{MGS}(X), not\ beggar_1(X).$

However rejected beggars, especially malicious ones, may turn into attackers, in this very bad neighbourhood. Consequently, if he thinks a beggar might attack him he'd better be more permissive about what is a beggar and placate him with money. In other words, he should maximize $beggar^+$ and minimize $beggar^-$ in a $beggar_2$ concept.

$beggar_2(X) \leftarrow beggar^+_{MGS}(X), not\ \neg beggar_2(X).$

$\neg beggar_2(X) \leftarrow beggar^-_{LGS}(X), not\ beggar_2(X).$

These concepts can be used in order to minimize his risk taking when he carries a lot of money, with the following kind of reasoning:

$run \leftarrow a_lot_of_money, attacker_1(X), not\ beggar_2(X).$

$\neg run \leftarrow give_money.$

$give_money \leftarrow beggar_1(X).$

$give_money \leftarrow attacker_1(X), beggar_2(X).$

If he doesn't 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 $attacker_2$, but not relaxed so much that he would use $attacker^-_{MGS}$.

$\neg run \leftarrow little_money, attacker_2(X).$

$attacker_2(X) \leftarrow attacker^+_{LGS}(X), not\ \neg attacker_2(X).$

$\neg attacker_2(X) \leftarrow attacker^-_{LGS}(X), not\ attacker_2(X).$

5 Related Work

The problems raised by negation and uncertainty in concept-learning, and Inductive Logic Programming in particular, were pointed out in some previous work (e.g., [4, 6]). De Raedt and Bruynooghe [6] proposed to use a three-valued logic and an explicit definition of the negated concept in concept learning. This technique has been integrated within the CLINT system, an interactive concept-learner. In the resulting system, both a positive and a negative definition are learned for a concept (predicate) p, stating, respectively, the conditions under which p is true and false. The definitions are learned so that they do not pro-

duce an inconsistency on the examples. Differently from this system, we take also care that the two definitions do not produce inconsistency on unseen atoms and we are able to learn definitions for exceptions to both concepts.

In a recent work, Inoue and Kudoh [13] present a method for learning ELP in Inductive Logic Programming, under the answer set semantics. In particular, their system (called LELP, Learning ELP) learns default rules with exceptions in the form of ELP given incomplete positive and negative examples and background knowledge. LELP generates a definition for the concept (positive or negative) whose example set is bigger. Both concepts are learned only when the number of positive examples is close to that of negative ones. Rules are learned from positive examples only (resp. negative examples only) and background knowledge in an ordinary ILP (bottom-up) framework. Then exceptions are identified (from covered examples of the opposite set) and rules specialized through negation as default and abnormality atoms, which are then assumed in order to prevent the coverage of exceptions. These assumptions can be, in their turn, generalized to generate hierarchical default rules. Finally, in order to generate consistent programs, learned rules are made non-deterministic in the context of answer set semantics. We rely, instead, on a well-founded semantics for ELP, and the *SLX* interpreter in particular, which is computationally less expensive than the answer set semantics, and allows for top-down query procedures. For non-stratified programs, which the authors require, answer sets semantics does not enjoy the structural property of relevance [8, 9], as *WFSX*, and so they cannot employ top-down querying methods. Also, we do not have to eventually compute all answer-sets to find out if an atom is unknown. Last but not least, answer set semantics is not cumulative, which implies that you cannot assert what you learn without the risk of changing the semantics of what you learned.

Several other authors have also addressed the task of learning rules with exceptions [7]. In these frameworks, nonmonotonicity and exceptions are dealt with by learning logic programs with negation. In [7] the authors rely on a language which uses a limited form of "classical" (or, better, syntactic) negation together with a priority relation among the sentences of the program [14] which can be easily mapped into negation as default.

It is worth mentioning that the treatment of exceptions by means of the

addition of a non-abnormality literal (as we and [13] do) to each rule is similar to the approach for declarative debugging followed in [19]. In order to debug a logic program, in [19] the authors first transform it by adding a different default literal to each rule. These literals are then used as assumptions of the correctness of the rule, to be possibly revised in the face of a wrong solution. The debugging algorithm determines the assumptions that led to the wrong solution, thus identifying the incorrect rules.

Non-abnormality literals can also be viewed as new abducible predicates, as done for instance in [15, 10, 12]. In particular, in [15, 10] the authors have considered the integration and cooperation of induction and abduction in order to learn Abductive Logic Programs (ALP) from (possibly) incomplete background knowledge expressed as ALP in its turn. In order to make a rule for a target predicate p consistent, the rule is specialized by adding a new abducible literal $not_abnorm_i(\vec{X})$. Exceptions are then ruled out by assuming $abnorm_i(\vec{t})$ for them. These assumptions are then used to learn a definition for $abnorm_i$ that describes the class of exceptions. In this way, they are able to learn hierarchies of exceptions.

6 Conclusions

We have shown that Extended Logic Programs are an appropriate representation formalism for learning in a three-valued setting. In this case, one has to learn a definition for both the target and the opposite concept, by considering positive and negative examples as instances of two different disjoint classes.

With ELP we are able to represent opposite concepts and to deal with their interaction. Inconsistencies are dealt with differently according to the type of atom causing them. Unseen atoms are classified as unknown through the adoption of non-deterministic rules, while exceptions are dealt with by non-abnormality defaults.

The level of generality of the learned definitions also impact on their interaction. In particular, we can learn the Least General Solution or the Most General Solution for the positive and negative concept. The generality of the solutions should be chosen independently for the two definitions, thus leading to four epistemological cases which can be strategically combined to achieve

diversely useful learned concepts. Future work will be devoted to identify the type of learning problems more appropriate for each case.

Acknowledgments

This research was partially funded by the PRAXIS XXI project MENTAL, and a NATO sabbatical scholarship to L. M. Pereira. We would like to thank the participants of the post-ILPS97 Workshop on Logic Programming and Knowledge Representation for useful comments and insights on this work.

References

[1] J. J. Alferes and L. M. Pereira. *Reasoning with Logic Programming*, volume 1111 of *LNAI*. SV, Heidelberg, 1996.

[2] J.J. Alferes, C.V. Damásio, and L.M. Pereira. Top-down query evaluation for well-founded semantics with explicit negation. In *Proceedings of the European Conference on Artificial Intelligenece ECAI94*, pages 140–144. Morgan Kaufmann, 1994.

[3] J.J. Alferes, T.C. Przymusinski, and L.M. Pereira. "Classical" negation in non-monotonic reasoning and logic programming. *Journal of Automated Reasoning*, 1, 1998.

[4] M. Bain and S. Muggleton. Non-monotonic learning. In S. Muggleton, editor, *Inductive Logic Programming*, pages 145–161. Academic Press, 1992.

[5] C. Baral and M. Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.

[6] L. De Raedt and M. Bruynooghe. On negation and three-valued logic in interactive concept learning. In *Proceedings of the 9th European Conference on Artificial Intelligence*, 1990.

[7] Y. Dimopoulos and A. Kakas. Learning Non-monotonic Logic Programs: Learning Exceptions. In *Proceedings of the 8th European Conference on Machine Learning*, 1995.

[8] J. Dix. A classification-theory of semantics of normal logic programs: I. strong properties. *Fundamenta Informaticae*, XXII(3):227–255, 1995.

[9] J. Dix. A classification-theory of semantics of normal logic programs: Ii. weak properties. *Fundamenta Informaticae*, XXII(3):257–288, 1995.

[10] F. Esposito, S. Ferilli, E. Lamma, P. Mello, M. Milano, F. Riguzzi, and G. Semeraro. Cooperation of abduction and induction in logic programming. In Peter Flach and Antonis Kakas, editors, *Abductive and Inductive Reasoning*, Pure and Applied Logic. Kluwer, 1998. Submitted for publication.

[11] M. Gelfond and V. Lifschitz. Classical negation in logic programming and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[12] Katsumi Inoue. Learning abductive and nonmonotonic logic programs. In Peter Flach and Antonis Kakas, editors, *Abductive and Inductive Reasoning*, Pure and Applied Logic. Kluwer, 1998. Submitted for publication.

[13] Katsumi Inoue and Yoshimitsu Kudoh. Learning extended logic programs. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 176–181. Morgan Kaufmann, 1997.

[14] A.C. Kakas, P. Mancarella, and P.M. Dung. The acceptability semantics for logic programs. In *Proceedings of the 11th International Conference on Logic Programming*, 1994.

[15] E. Lamma, P. Mello, M. Milano, and F. Riguzzi. Integrating Induction and Abduction in Logic Programming. In P. P. Wang, editor, *Proceedings of the Third Joint Conference on Information Sciences*, volume 2, pages 203–206, 1997.

[16] S. Lapointe and S. Matwin. Sub-unification: A tool for efficient induction of recursive programs. In D. Sleeman and P. Edwards, editors, *Proceedings of the 9th International Workshop on Machine Learning*, pages 273–281. Morgan Kaufmann, 1992.

[17] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[18] S. Muggleton and W. Buntine. Machine invention of first-order predicates by inverting resolution. In S. Muggleton, editor, *Inductive Logic Programming*, pages 261–280. Academic Press, 1992.

[19] L. M. Pereira, C. V. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *Proceedings of the 2nd International Workshop on Logic Programming and Non-monotonic Reasoning*, pages 316–330. MIT Press, 1993.

[20] L.M. Pereira and J.J. Alferes. 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, 1992.

[21] L.M. Pereira, Aparício, and J.J. Alferes. Nonmonotonic reasoning with well founded semantics. In *Proceedings of the International Conference on Logic Programming ICLP91*, pages 475–489. The MIT Press, 1991.

[22] L.M. Pereira, Aparício, and J.J. Alferes. Non-monotonic reasoning with logic programming. *Journal of Logic Programming*, 17:227–263, 1993.

[23] G.D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

[24] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[25] R. Reiter. On closed-word data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 55–76. Plenum Press, 1978.