Automatic Induction of Abduction and Abstraction Theories from Observations

Stefano Ferilli, Teresa M.A. Basile, Nicola Di Mauro, and Floriana Esposito

Department of Computer Science, University of Bari, Italy
{ferilli, basile, ndm, esposito}@di.uniba.it

Abstract. Traditional Machine Learning approaches are based on single inference mechanisms. A step forward concerned the integration of multiple inference strategies within a first-order logic learning framework, taking advantage of the benefits that each approach can bring. Specifically, abduction is exploited to complete the incoming information in order to handle cases of missing knowledge, and abstraction is exploited to eliminate superfluous details that can affect the performance of a learning system. However, these methods require some background information to exploit the specific inference strategy, that must be provided by a domain expert.

This work proposes algorithms to automatically discover such an information in order to make the learning task completely autonomous. The proposed methods have been tested on the system INTHELEX, and their effectiveness has been proven by experiments in a real-world domain.

1 Introduction

In real-life domains, learning systems often have to deal with various kinds of imperfections in data: presence of random errors in both training examples and background knowledge (*noise*); too sparse training examples from which it is difficult to reliably detect correlations (*incompleteness*); inappropriateness of the description language which does not contain/facilitate an exact representation of the target concept (*inexact data*). Another kind of imperfection, more difficult to be dealt with, is represented by missing values in the training examples. As a solution, various noise-handling mechanisms have been exploited.

In dealing with such situations, most traditional Machine Learning approaches that exploit simple or constrained knowledge representations for the sake of efficiency, and are based on single (often simple or simplified) inference mechanisms, have reached their limits [16]. In order to investigate how to broaden the applicability of machine learning schemes, it is necessary to make different inference strategies work together, taking advantage of the benefits that each approach can bring. Many studies presented in the literature aimed at enforcing the integration of multiple inference strategies within a logic programming framework for first-order logic learning.

S. Kramer and B. Pfahringer (Eds.): ILP 2005, LNAI 3625, pp. 103–120, 2005.

The general schema of the inductive concept-learning paradigm ($BK \cup T \models O$) involves four variables, namely: the language $\mathcal{L}$, for which in this work the *single representation trick* [1] will be assumed, the background knowledge BK and the theory T that contains concept definitions explaining the occurrence of some observations O. Observations O stand for the extensional representation of concepts, and the aim is building an intensional description T, expressed in the language $\mathcal{L}$, that explains such concepts, supposed that BK is insufficient to give such an explanation. Most approaches focus on inductive mechanisms to fine-tune T in order to achieve the learning goal.

Two problems of the traditional approach to concept-learning can be singled out: the partial relevance of the available evidence O and the insolvability of a learning problem when the language $\mathcal{L}$ is not enough powerful to express a proper predicate definition in T. Abduction and abstraction can be exploited, respectively, to overcome such limitations: the former could bridge the gap between the observations and the definitions in the theory. The latter could shift to a higher language bias when the current one does not allow to capture the target predicate definition. From an operational viewpoint, abduction should somehow complete the observations with unknown facts that are likely to take place in the given situation and that can help in solving the learning problem at hand; it can be carried out by an abductive proof procedure, that shares the falsity-preserving nature with the inductive refinement operators [14]. As regards abstraction, it should deal with cases when learning can be more effective if it can take place at multiple (different) levels of complexity, which can be compared to the language bias shift considered in [2]; a useful perspective for the integration of this inference operator in an inductive learning framework was given in [23].

According to such a perspective, the incremental ILP system INTHELEX was extended in previous works to exploit abduction and abstraction to support the learning process [5]. However, it assumes that the information needed to apply the additional inference strategies is provided by the user. The objective of this work is investigating solutions for the automatic inference of such information from the same observations that are input to the inductive process, assuming that they are sufficiently significant. Abstraction should simplify the description language by grouping or eliminating correspondences that hold often or seldom, respectively, among the given observations. Abduction should consider as integrity constraints combinations of properties and relations that do not hold in the available observations. In the former case, the method focuses on the discovery of sets of common features in the observations; in the latter, sets of mutually exclusive features have to be singled out.

2 The General Framework

2.1 Handling Incomplete Information: Abduction

The problem of abduction, defined as *inference to the best explanation* according to a given domain theory, can be formalized as follows [4]: **Given** a theory T[1],

[1] Here, the theory T is assumed to include also the background knowledge.

some observations O and some constraints I, **Find** an explanation H such that: $T \cup H$ is consistent and satisfies I, $T \cup H \models O$. Candidate explanations H should be described in terms of domain-specific predicates, referred to as *abducibles*, that are not (completely) defined in T, but contribute to the definition of other predicates. They carry all the incompleteness of theory T: if it was possible to complete these predicates then the theory would be correctly described. The integrity constraints I should provide indirect information about them [10].

Since abduction is able to capture *default reasoning* (a form of reasoning which deals with incomplete information [10]), it can be exploited to face the problem of relevance and incompleteness. Indeed, when partial relevance is assumed, it could be the case that not only the set of all observations is partially known, but also any single observation may turn out to be incomplete. The usual *Abductive Logic Programming* framework [14, 6] can be adapted to concept learning theory revision problem as follows:

Definition 1. *An* abductive logic theory *is a triple* $AT = (T, \mathcal{A}, \mathcal{I})$ *where* T *is a (hierarchical) normal logic program;* $\mathcal{A}$ *is the set of abducible predicates;* $\mathcal{I}$ *is a set of integrity constraints represented as program clauses.*

In the original ALP framework, the theories are full normal logic programs interpreted according to the Stable Model semantics [11]. We restrict to *hierarchical* theories in order to exploit the Least Herbrand Models semantics, where if $T \models P_1, \ \ldots, \ T \models P_n$ then it also holds that $T \models P_1 \wedge \cdots \wedge P_n$, which is fundamental in an incremental setting, where examples are provided over time, to check correctness of the refined hypotheses with respect to older examples by testing them separately. This cannot be done when stable models semantics is adopted (cf. [6] for an example). Additionally, Least Herbrand Models semantics allows to cope with negation by means of the Negation as Failure rule, without transforming the theory and goals into their *positive version*, as required by the original framework. The integrity constraints $\mathcal{I}$ can be represented in principle as any first order formulæ. Some restrictions are to be applied: in the integrated framework described in [3], they are represented as range-restricted Horn clauses.

An abductive proof procedure can find explanations that make hypotheses (abductive assumptions) on the state of the world, possibly involving new abducible concepts, and is generally goal-driven by the observations that it tries to explain. The abductive proof procedure proposed in [12] works just like a standard SLD derivation [15], only when a literal cannot be proved the procedure does not fail immediately but first checks if it can be (or has already been) abductively hypothesized. In such a case, a consistency-check subroutine must ensure that no integrity constraints $\mathcal{I}$ is violated, by inductively or abductively deriving the falsity of at least one literal in each of them. Thus, the two procedures may call each other both when a new abductive assumption requires further consistency checks against the constraints and vice-versa.

2.2 Shifting Representation Language: Abstraction

Abstraction is defined as a mapping between representations that are related to the same reference set but contain less detail (typically, only the information

that is relevant to the achievement of the goal is maintained). It is useful in inductive learning when the current language bias proves not to be expressive enough for representing concept descriptions that can explain the examples.

Definition 2. *Given two clausal theories T (*ground theory*) and T' (*abstract theory*) built upon different languages $\mathcal{L}$ and $\mathcal{L}'$ (and derivation rules), an* abstraction *is a triple (T, T', f), where f is a computable total mapping between clauses in $\mathcal{L}$ and those in $\mathcal{L}'$.*

An Abstraction Theory (an operational representation of f) is used to perform such a *shift of language bias* [22, 2] to a higher level representation:

Definition 3. *An* abstraction theory *from $\mathcal{L}$ to $\mathcal{L}'$ is a consistent set of clauses $c : -d_1, \ldots, d_m$ where c is a literal built on predicates in $\mathcal{L}'$, and $d_j,\ j = 1, \ldots, m$ are literals built on predicates of $\mathcal{L}$.*

i.e., it is a collection of intermediate concepts represented as a disjunction of alternative definitions. *Inverse resolution* operators [17] (inter-construction, intra-construction and absorption) can be a valuable mechanism to build and exploit abstraction theories, as introduced in [9]. This work is interested in the case of Datalog programs, as in [19], where clauses are *flattened*, hence function-free.

Definition 4 (absorption). *Let C and D be two Datalog clauses. If $\exists\theta$ unifier such that $S = body(D)\theta \subset body(C)$, then applying the absorption operator yields the new clause C' such that $head(C') = head(C)$ and $body(C') = (body(C) \setminus S) \cup \{head(D)\theta\}$.*

i.e., if all conditions in D are verified in the body of C, the corresponding literals are eliminated and replaced by $head(D)$.

According to the framework proposed in [23], abstraction takes place by means of a set of operators, that generally includes operators for grouping indistinguishable objects into equivalence classes; grouping ground objects to form a compound object (that replaces them in the abstract world); ignoring terms (that disappear in the abstract world); merging values that are considered indistinguishable; reducing the arity of a function or relation (even up to elimination of *all* arguments). Modifications are performed by mappings.

2.3 Learning Background Knowledge for Multi-inference Strategies

As already pointed out, the exploitation of the two strategies reported above and their integration in an inductive concept learning framework is based on the assumption that the knowledge needed to use them is provided by an expert of the application domain. Here we propose an approach to automatically learn such knowledge to be exploited by the abduction/abstraction operators. It is worth recalling that the feasibility of reaching the target solution requires that the number of values for the domains to be identified and the amount of available knowledge about observations to be strictly proportional. Indeed, the more the values, the more the possible interrelations that can take place between them. If

the available observations are not sufficiently significant, i.e. too many existing interrelations are not recognizable in them, then knowledge about the actual biases in the given domain would be too loose for the algorithm to properly infer significant and correct information.

3 Learning Abduction Theories

The exploitation of the abductive proof procedure presented in Section 2.1 requires the specification of an abductive theory for the specific application domain. Usually, it is provided by a domain expert; in the following we propose a methodology for automatically inferring it starting from the available observations, in order to make the learning system completely autonomous.

3.1 Abducibles

In setting up an abductive logic programming task, the logic program is typically to be learnt, while abducibles and integrity constraints have to be provided by the domain expert. Thus, a first problem is deciding on which properties and/or relations abduction can be carried out, i.e. listing the abducibles. Indeed, abductive reasoning needs to know them in order to assess on which concepts abductions (i.e., guesses about unknown facts) can be made. We assume that all predicates that make up the description language, and have no definition in the theory (in order to fulfil the requirements for abducibles [4]), are considered as abducibles since by hypothesis some of their instances could be missing in the available dataset. Indeed, in the absence of further information, any fact that can help in solving the problem at hand is useful, and the automatic system should be allowed to hypothesize it, in order to provide the abductive reasoner with all the freedom it needs for hypothesizing information.

3.2 Integrity Constraints

The other issue, far more complex, concerns the definition of the integrity constraints. It is, at the same time, a fundamental and difficult task, whose quality can determine the very feasibility of the learning process. Hence, the motivation for automatically inferring such constraints, this way overcoming possible problems related to omissions and/or wrong formalization of the human expert.

Learning denials (the form in which integrity constraints are coded in an abductive theory) cannot be simply cast as a supervised learning task, since it aims at inducing rules whose head is empty. Rather, it can be seen as a specific case of unsupervised learning aimed at finding regularities (specifically, conditions that are never verified) in a first-order logic database. Thus, the data mining approaches are better suited to carry out this task. Some systems are present in the literature that can learn denials. One of them is Claudien [18], that actually implements a more general algorithm for finding regularities that occur

in a set of unlabelled observations represented as facts. It requires a template of the clauses to be induced, and can limit the corresponding search space using heuristics and resource bounds. By properly setting its parameters, it can be applied for learning classification rules, association rules, (non necessarily definite) clauses and also denials. Such a system inspired a number of successive works, among which the development of the systems Primus and its successor Tertius [8]. They are based on the generation of possible (H, B) couples, where H and B are sets of literals in the given description language to be interpreted, possibly negated, as candidate head and body, respectively, of a clause to be generated. The frequency with which each candidate rule is (or is not) verified in the dataset is computed, and statistical approaches are exploited to decide if such frequencies are significant, in which case a corresponding rule is generated. Background knowledge (i.e., derived predicates such as *ancestor* in a family environment) can be used, but increasing the number of literals in H and B causes high computational costs, thus sampling and non-redundant operators are exploited. Another widely known learning system that can induce integrity constraints is Aleph [21], that works in a similar way as Claudien. All of these systems can actually learn denials, but this is just a specific setting or a side-effect of a wider range of possibilities that the implemented algorithms provide. Thus, the aim of this paper is devising simpler procedures, purposely devoted to the generation of integrity constraints for an abductive theory, that being limited to this specific task can carry out it in a more focused and effective way.

The starting point in doing this is the fact that integrity constraints represent situations that cannot occur in the described world. Thus, the available observations cannot actively help in defining them. Rather, the aim is identifying combinations of descriptors and of the related arguments that cannot hold. In doing so, one possible strategy is generating a number of such combinations, according to a given strategy, and then exploiting the available observations passively to check if the generated combination occur in at least one case or not. In the former case, it cannot be a constraint, according to the assumption that observations are correct and report only true information. In the latter case, this can be taken as a suggestion, but not of course as a guarantee (since its absence could be due to just the fact that by chance that situation did not ever occur in the specific observations at hand), that the combination does not occur because it in fact makes no sense in the considered world[2]. This, of course, raises the problem of having a set of observations that is significant not only numerically, but also in the sense that they depict a significant amount of different cases. Nevertheless, such a significance should be assumed, because otherwise the learning task itself, to be carried out on such observations, would hardly make sense.

Now, the point is how to proceed in generating the literals (and variables) combinations to be tested. Indeed, it is clear that generating and testing all possible combinations becomes soon impossible even for relatively small datasets. Bounding the cardinality of the combinations to be generated to a given l, al-

[2] In any case, this makes useless counting the frequencies as in Tertius, since every combination that is verified is not a constraint, no matter how many times it happens.

though useful, is not sufficient to avoid the combinatorial explosion. Thus, it is necessary to identify specific classes of constraints that can be considered meaningful in general (i.e., without reference to specific datasets or environments) and thus are worth checking. A first important class is that of object properties, represented by unary predicates. Indeed, it is undoubtedly interesting to know which combinations of attributes are (im-)possible for a given object, in order for the abductive proof procedure to avoid them (e.g., it generally holds that a line is either tall or wide, but cannot be both at the same time). In this case, the problem can be significantly simplified since the presence of just one variable in the predicates allows to focus on just the predicates combinations, excluding the generation of duplicate literals and the presence of unrelated variables. The procedure is detailed in Algorithm 1.. *NotConstraints* and *Constraints* are two (initially empty) lists, containing the currently identified non-constraints and constraints, respectively. The presence of each potential constraint in the observations is checked: in case of success, it is added to the list of constraints, provided that the *not_trivial* function succeeds. A constraint is considered trivial if it is a superset of some other (shorter) constraint that is already present in the Constraints list, so the *not_trivial* function avoids generating (and learning) redundant constraints, just like in related work. In the first step, all possible n-tuples (with $2 \leq n \leq N$ for a fixed N) of unary predicates, all applied to the same variable, are generated and checked for occurrence in the available observations. The generation proceeds from lower to higher values of n. First, all pairs of unary predicates are generated and checked for occurrence: those that are not satisfied by the observations are considered constraints and added to the *Constraints* list; conversely, those that happen at least once are added to the *NotConstraints* list. Then, all non-constraints of cardinality 2 are extracted from *NotConstraints* and extended with one more unary predicate, checked for occurrence and added to *NotConstraints* or *Constraints* accordingly. Then, all newly found non-constraints of cardinality 3 are extended and checked, and so on until the fixed N is reached.

However, although very useful, constraints on properties are not sufficient. It is often important, for the purpose of learning a significant abduction theory, to consider also constraints built on n-ary predicates. Without loss of generality, in this work we restrict to binary predicates, and propose a set of typical relationships among the arguments that appear in pairs of such predicates that are deemed as significant to be exploited as constraints. Specifically, given two *predicate variables* P and Q (not necessarily distinct) ranging on binary predicates of the representation language $\mathcal{L}$, and three variables X, Y, Z, the *rules schemas* [13] (denials) that we propose to check are $\leftarrow P(X, X)$. (*reflexivity*), $\leftarrow P(X, Y), Q(Y, X)$. (*symmetry*), $\leftarrow P(X, Y), Q(Y, Z)$. (*transitivity*), $\leftarrow P(X, Y), Q(Z, Y)$. (*convergence*), and $\leftarrow P(X, Y), Q(X, Z)$. (*divergence*).

In the next step, all binary predicates are considered, and checked for occurrence of the reflexive, symmetric, transitive, converging and diverging relationships. Again, when a relationship has no counterpart in the available observations, it is added to the *Constraints*, otherwise it is added to the *NotConstraints*.

Algorithm 1. Induction of Integrity Constraints made up of unary predicates

Create_Constraints(N**;** $\mathcal{E}$**;** $UnaryPreds$**;** $NotConstraints$**;** $Constraints$**);**
{ N: Maximal cardinality of constraints to generate; $\mathcal{E}$: Set of observations; $UnaryPreds$: Set of Unary Predicates; $NotConstraints$: Set of non-Constraints; $Constraints$: Set of Integrity Constraints }
$NotConstraints := \emptyset$; $Constraints := \emptyset$
for all $a, b \in UnaryPreds, a \neq b$ **do**
  if $\mathcal{E} \vdash \{a(X), b(X)\}$ **then**
    $NotConstraints := NotConstraints \cup \{\{a(X), b(X)\}\}$
  else
    $Constraints := Constraints \cup \{\{a(X), b(X)\}\}$
for $n := 3..N$ **do**
  for all $NC \in NotConstraints, |NC| = n - 1$ **do**
    for all $a(X) \in UnaryPreds$ **do**
      if $not\_trivial(Constraints, \{a(X)\} \cup NC)$ **then**
        if $\mathcal{E} \vdash \{\{a(X)\} \cup NC\}$ **then**
          $NotConstraints := NotConstraints \cup \{\{a(X)\} \cup NC\}$
        else
          $Constraints := Constraints \cup \{\{a(X)\} \cup NC\}$

Lastly, all possible combinations of non-constraints on binary predicates relationships and on unary predicates (applied to any of the variables appearing in the former), whose cardinality does not exceed the fixed N, are checked for occurrence and added to the *Constraints*, if it is the case, according to Algorithm 2.. It starts the process taking as input the list of *non-constraints*, both unary and binary, built so far. $UnaryNotConstrs$ and $BinaryNotConstrs$ are the sets of non-constraints found in the previous steps. Since all constraints on unary predicates have at least cardinality 2, a preliminary step in which all possible combinations of constraints on binary predicates with a single unary predicate must be separately checked. Note that, in this step, no candidate constraint can be trivial, since its binary component is not a constraint by itelf and its unary component is just a singleton. Conversely, in the loop that combinates unary and binary constraints, the only way a constraint can be trivial is being a superset of a constraint obtained in the previous loop, since none of its components is a constraint by itself.

Example 1. Consider the description language made up of the predicates: { block/1, line/1, low/1, medium/1, high/1, narrow/1, wide/1, part_of/2, on_top/2, to_right/2 }. Let the available observations be:
{ part_of(a,b), part_of(a,c), part_of(a,d), part_of(a,e), part_of(a,f), line(b), medium(b), narrow(b), block(c), high(c), wide(c), line(d), low(d), wide(d), block(e), medium(e), wide(e), block(f), medium(f), wide(f), on_top(d,b), on_top(d,e), on_top(d,f), on_top(b,c), on_top(e,c), on_top(f,c),to_right(b,e), to_right(f,b) } (representing the block world in Figure 1) and N be fixed to 4.

- Step 1:
  - Pairs of unary predicates: ***Constraints*** = { $\{block(X), line(X)\}$, $\{block(X), low(X)\}$, $\{block(X), narrow(X)\}$, $\{line(X), high(X)\}$,

Algorithm 2. Integrity Constraints made up of unary/binary predicates

Create_constraints_with_binary_and_unary_literals(N**;** $Unary$**;** $Constrs$**;** $UnaryNotConstrs$**;** $BinaryNotConstrs$**);**
{N: Maximal cardinality of constraints to generate; $\mathcal{E}$: Set of observations; $Unary$: Set of Unary Predicates; $Constrs$: Set of Integrity Constraints made up of unary or binary predicates; $UnaryNotConstrs$: Set of non-constraints made up of unary predicates; $BinaryNotConstrs$: Set of non-constraints made up of binary predicates.}
for all $NC \in BinaryNotConstrs, X \in vars(NC), p \in Unary$ **do**
  if $|NC| < N \wedge \mathcal{E} \not\models NC \cup \{p(X)\}$ **then** $Constrs := Constrs \cup \{\{p(X)\} \cup NC\}$
for all $BNC \in BinaryNotConstrs$ **do**
  V := vars(BNC); $TentativeConstr$:= BNC;
  for all $S \subseteq V$ **do**
    apply a $UNC \in BinaryNotConstrs$ to each $X \in S$, add it to $TentativeConstr$
    if $|TentativeConstr| \leq N \wedge not\_trivial(Constrs, TentativeConstr) \wedge$
    $\mathcal{E} \not\models TentativeConstr$ **then**
      $Constrs := Constrs \cup \{TentativeConstr\}$

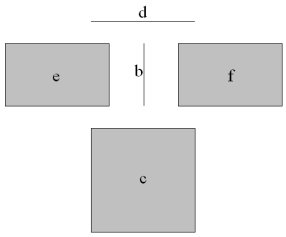

Fig. 1. Sample block world

$\{low(X), medium(X)\}, \{low(X), high(X)\}, \{low(X), narrow(X)\},$
$\{medium(X), high(X)\}, \{high(X), narrow(X)\}, \{narrow(X), wide(X)\}$ }
NotConstraints = { $\{block(X), medium(X)\}, \{block(X), high(X)\},$
$\{block(X), wide(X)\}, \{line(X), low(X)\}, \{line(X), medium(X)\},$
$\{line(X), narrow(X)\}, \{line(X), wide(X)\}, \{low(X), wide(X)\},$
$\{medium(X), narrow(X)\}, \{medium(X), wide(X)\}, \{high(X), wide(X)\}$ }

- Triplets of unary predicates (extending couples of *NotConstraints*):
Constraints=$\{\{line(X), medium(X), wide(X)\}, \{line(X), high(X), wide(X)\}\}$
NotConstraints = { $\{block(X), medium(X), wide(X)\},$
$\{block(X), high(X), wide(X)\}, \{line(X), low(X), wide(X)\},$
$\{line(X), medium(X), narrow(X)\}$ }
All other possible extensions of binary non-constraints are trivial.
- 4-tuples of unary predicates: all 4-tuples obtained extending ternary non-constraints are trivial, thus in this step both Constraints and NotConstraints are empty. As a side effect, there are no non-constraints of cardinality 4 to be extended, and hence no constraints of cardinality larger than 4 can be found.

– Step 2:

- Reflexivity: ***NotConstraints*** = $\emptyset$
Constraints = { $\{part\_of(X, X)\}, \{on\_top(X, X)\}, \{to\_right(X, X)\}$ }

- Symmetry: $\boldsymbol{NotConstraints} = \emptyset$
  $\boldsymbol{Constraints}$=$\{\{part\_of(X,Y), part\_of(Y,X)\}, \{on\_top(X,Y), on\_top(Y,X)\}, \{to\_right(X,Y), to\_right(Y,X)\}, \{part\_of(X,Y), on\_top(Y,X)\}, \{part\_of(X,Y), to\_right(Y,X)\}, \{on\_top(X,Y), to\_right(Y,X)\}\ \}$
- Transitivity: $\boldsymbol{NotConstraints} = \{\ \{on\_top(X,Y), on\_top(Y,Z)\}, \{to\_right(X,Y), to\_right(Y,Z)\}, \{part\_of(X,Y), on\_top(Y,Z)\}, \{part\_of(X,Y), to\_right(Y,Z)\}, \{to\_right(X,Y), on\_top(Y,Z)\}\ \}$
  $\boldsymbol{Constraints}$=$\{\{part\_of(X,Y), part\_of(Y,Z)\}, \{on\_top(X,Y), part\_of(Y,Z)\}, \{to\_right(X,Y), part\_of(Y,Z)\}, \{on\_top(X,Y), to\_right(Y,X)\}\}$
- Convergence: $\boldsymbol{NotConstraints} = \{\ \{on\_top(X,Y), on\_top(Z,Y)\}, \{on\_top(X,Y), part\_of(Z,Y)\}, \{to\_right(X,Y), part\_of(Z,Y)\}, \{on\_top(X,Y), to\_right(Z,Y)\}\ \}$
  $\boldsymbol{Constraints} = \{\ \{part\_of(X,Y), part\_of(Z,Y)\}, \{to\_right(X,Y), to\_right(Z,Y)\}\ \}$
- Divergence: $\boldsymbol{NotConstraints} = \{\ \{part\_of(X,Y), part\_of(X,Z)\}, \{on\_top(X,Y), on\_top(X,Z)\}, \{on\_top(X,Y), to\_right(X,Z)\}\ \}$
  $\boldsymbol{Constraints} = \{\ \{to\_right(X,Y), to\_right(X,Z)\}, \{on\_top(X,Y), part\_of(X,Z)\}, \{to\_right(X,Y), part\_of(X,Z)\}\ \};$

– Step 3 (omitted due to lack of space)

3.3 Descriptors Type Domains and Abducibles

At the end of the procedure reported in Algorithm 1., the set of constraints of cardinality 2 can be input to the type induction procedure presented in [7] in order to infer type domains. Then, all pairs of unary predicates belonging to the same domain can be eliminated from the set *Constraints*, thus reducing the complexity of the abductive proof procedure, and a new kind of constraint will be introduced to represent types, such that no two values from the same type domain will be allowed applied to the same object. For example, if the descriptor type domain for the color property is {*blue*, *red*, *yellow*, *black*, *green*}, and the object X is part of an observation, it will be impossible to abduce two different color descriptors from the above set applied to X.

4 Learning Abstraction Theories

As pointed out in Section 2.2, abstraction aims at discarding or hiding the information that is irrelevant to achieve the goal. To be able to perform abstraction during the learning task, the system must be provided with an abstraction theory for the specific application domain, that (according to Definitions 2 and 3) contains the operators encoding the abstraction mapping f between languages $\mathcal{L}$ and $\mathcal{L}'$ represented as a set of clauses, i.e. domain rules.

Usually, such domain rules are hand-coded by the domain expert; this section proposes a methodology aimed at automatically learning them. The main idea underlying the proposed strategy consists in searching for correspondences that often or seldom hold among the available set of observations. These correspondences are then exploited to simplify the description language in two different

Algorithm 3. Identification of shifting/neglecting rules

Require: $\mathcal{E}^+$: set of positive observations; $\mathcal{E}^-$: set of negative observations; e: seed;
Provide: AT: set of domain rules that make up an abstraction theory;
if $\exists$ unary predicates in e **then**
$S := \emptyset$, $UnaryPreds :=$ set of unary predicates in e
$C := \{c_1, c_2, \ldots, c_n\}$ set of constants in the description of e
for all $c_i \in C$ **do**
$S_i := \{l_i \in UnaryPreds$ s.t. c_i is argument of $l_i\}$
if $\mid S_i \mid \neq 0$ and $\mid S_i \mid \neq 1$ **then** $S := S \bigcup S_i$
for i=1..n **do**
for all $S_j \in S$ **do**
find all the subsets s_{jm} of S_j s.t.
$(0 - \alpha \leq Score(s_{jm}) \leq 0 + \alpha)$ OR $(Max - \alpha \leq Score(s_{jm}) \leq Max + \alpha)$
create the rule: $rule_{s_{jm}}(c_i) \leftarrow s_{jm}$
replace in $\mathcal{E}^+$, in $\mathcal{E}^-$ and in e, s_{jm} with $rule_{s_{jm}}(c_i)$
while F (:= set of all leaf predicates of e) $\neq \emptyset$ **do**
for all $l_i \in F$ **do**
if l_i has only one parent (let $g_i(a_i, \ldots, a_n)$ be the l_i's parent) **then**
create the rule: $rule_{l_i}(a_i, ...a_n) \leftarrow g_i, l_i$; H := true
replace in $\mathcal{E}^+$, in $\mathcal{E}^-$ and in e, g_i, l_i with $rule_{l_i}(a_i, ...a_n))$
for all $rule_i \leftarrow l_{i_1}, \ldots, l_{i_n}$ generated **do**
if $\{l_{i_1}, \ldots, l_{i_n}\}$ occurs in some rule $rule_j$ **then**
replace $l_{i_1}, \ldots, l_{i_n}$ in $rule_j$ by $rule_i$
eliminate $rule_i$ form the set of rules generated
Evaluate the set of generated rules

ways: by generating *shifting rules* that replace significant, characteristic or discriminant groups of literals by one single literal representing their conjunction, or by generating *neglecting rules* that eliminate groups of literals that are not significant[3]. Both kinds of rules will be applied in order to perform the shift of language bias according to the absorption operator presented in Definition 4.

Algorithm 3. sketches the overall procedure conceived to discover common paths in the application domain that potentially could make up the Abstraction Theory. It firstly generates domain rules involving unary predicates only, that represent the characteristics of an object in the description, and then the rules involving predicates whose arity is greater than 1, that represent the relationships between two or more objects contained in the descriptions. Crucial point of the algorithm is the choice of the observation (referred to in the following as the *seed*) that will act as the representative of the concept that one would abstract. It is currently selected as the first encountered observation.

Once the seed is identified, for each constant c_i in its description, the algorithm finds the set of all the unary predicates having that constant as an argument, and computes all its possible subsets (except the empty set, that does

[3] The loss in detail is evident in the latter, while in the former derives from the impossibility to handle independently subsets of the grouped literals.

not give information about the object, or the singleton subsets, that express just single properties of the objects). Each subset identified in this way is a potential candidate to become the body of a rule, in the Abstraction Theory, made up of unary predicates. The selection among these subsets is done considering those that are the best representative for the class of the concept to be abstracted according to the seed e. Specifically, each subset is assigned a *score* based on the number of times that it occurs in the descriptions of positive and negative examples in the whole training set. Such a value represents the *coverage rate* of the subset with respect to the observations and indicates the quality of the subset. The selection aims at choosing those subsets that are neither too specific, because they are present in few observations, nor too general, because they are encountered in almost all the observations. Each selected subset s_j, interpreted as a conjunction of literals, becomes the body of a rule in the Abstraction Theory, formulated in the following way:

$$abstract\_predicate(c_i) \leftarrow s_j \quad \text{iff} \quad score(s_j) \geq P \qquad \text{(shifting rule)}$$
$$\leftarrow s_j \quad \text{iff} \quad score(s_j) \leq P \qquad \text{(neglecting rule)}$$

where P is a threshold that depends on the application domain at hand[4]. In the former case s_j is present in almost all the observations, hence it is considered significant as a whole for the learning process and thus it is taken as the body of a shifting rule, to be replaced by a single abstract predicate. In the latter case s_j is assumed to indicate a detail in the description that is not very significant for the learning process and thus it is eliminated (replaced by an empty head). In both cases, the abstraction operators will replace each occurrence in the description of the observations of the rule's body with the corresponding head, this way reducing the description length of observations and hence making the learning process more efficient.

The algorithm continues with the identification of rules made up of predicates whose arity is greater than 1 representing the relationships between two or more objects. Thus, once the abstraction rules, that are identified in the previous step, are replaced in all the observations, they don't contain any unary predicates. At this point, an iteration that groups together the n-ary predicates is performed until one of the following conditions succeeds: 1) the description of the seed e does not contain *leaf predicates* (predicates that share arguments with at least another predicate); 2) the step n generates exactly the same rules already built in the step $n-1$. The search of the leaf predicates is particularly complex due to the large number of relationships that could hold between the objects in the descriptions. The identification of such predicates is done by representing the observation with a tree in which each level is determined by the propagation of the variables: the root is the head of the observation and its direct descendants are all the predicates that share with it at least one argument. This procedure is iterated until all the predicates in the description have been inserted in the tree (a considered predicate does not participate anymore to the tree construction).

[4] In order to make P independent on the specific domain, the score can be normalized as a percentage of the maximum score actually computed in the given dataset.

After the tree is constructed, we select the set of leaf nodes (predicates) that have only one parent, let it be $L = l_1, l_2, \ldots, l_n$. Successively, for each element $l \in L$ its parent, let be it the literal $g(a_1, \ldots, a_m)$, is extracted from the tree and the following rule is generated:

$$rule(a_1, \ldots, a_m) \leftarrow g(a_1, \ldots, a_m), l$$

Finally, for each rule $rule_i \leftarrow l_{i_1}, \ldots, l_{i_n}$ if the body of $rule_i$, i.e. $l_{i_1}, \ldots, l_{i_n}$, appears in some rule $rule_j$ then $l_{i_1}, \ldots, l_{i_n}$ is replaced in $rule_j$ by $rule_i$ and $rule_i$ is eliminated by the set of rules that are being generated. At the end of this step again the evaluation phase of the potential rules to make up the Abstraction Theory is performed according to the procedure above mentioned.

As to the *score* function, we need a statistical model able to take into account the significance of the subset for (i.e., its frequency in) the descriptions. Specifically, a significant subset should be able to characterize a concept, or to discriminate it from the others, better than other subsets. Conversely, a subset that is not characterizing or discriminant could be assumed as non-significant. An indication for such a setting could come from the distribution of the subset in the whole set of observations: in this perspective, an high significance value is associated to subsets that appear frequently in instances of one concept but rarely in instances of others (and hence help to distinguish a concept from the others), while a low significance value is associated to subsets that appear uniformly throughout different concepts (and hence are superfluous for the learning process). A statistical model that reflects such considerations is represented by the *Term Frequency - Inverse Document Frequency* (TF-IDF) [20]. Here, it must be adapted to a work context facing with positive and negative observations. In the following a brief description of the adapted method is provided.

Each subset s_i is associated with a vector $V_i = (V_{i1}, V_{i2}, \ldots, V_{iN})$ where N is the number of available observations and V_{ij} is the weight of the i-th subset in the j-th observation, computed as:

$$V_{ij} \;=\; FREQ_{ij} \;*\; (\lg \tfrac{N}{IFREQ_i} \;+\; 1)$$

The term $(\lg \frac{N}{IFREQ_i} + 1)$ represents the inverse of the frequency of subset s_i in the whole set of observations. Notice that the result of this computation will be positive if the j-th observation is positive, negative otherwise, thus the resulting vector will be of the form $V_i = (+, -, +, +, -, +, \ldots)$. This will allow to distinguish the significance of the subset according to its presence in the positive and negative observations. Now, having for each subset s_i the vector of its weights in the various observations, its score can be computed as follows:

$$score(s_i) \;=\; |\textstyle\sum_{j=1,\ldots,N} V_{ij}|$$

It is worth noting that this score will be around zero if the subset equally occurs in both positive and negative observations, in which case it is considered insignificant and could be exploited as a neglecting rule in the abstraction phase. Conversely, an high absolute value indicates a strong correlation of the subset with the positive or the negative observations. Specifically, highly positive (resp., negative) scores indicate that the subset is very frequent in the positive (resp.,

negative) observations. In both cases, it is considered significant and hence it could be exploited to build shifting rules for the abstraction phase.

Example 2. Let $h(1) : -p(1,2), p(1,4), p(1,5), c(2,3), f(5,6), d(4), s(6)$ the seed chosen in the observations (in this case it represents the set of observations too).

- **Step 1**:
  - *Grouping unary predicates:* $S = \emptyset$ (no groups of unary predicates, referred to the same constant, with cardinality strictly greater than 1 can be recognized);
- **Step 2**:
  - *Recognize Leaf Nodes:* $F = \{c(2,3),\ d(4),\ s(6)\}$, indeed $c(2,3)$ has only one parent $p(1,2)$; $d(4)$ has only one parent $p(1,4)$; $s(6)$ has only one parent $f(5,6)$.
  - *Create the rules -* $rule_{l_i}(a_i, ...a_n) \leftarrow g_i, l_i$*:*
    $c(2,3)$ with parent $p(1,2) \rightarrow rule1(X,Y) : -p(X,Y), c(Y,Z)$.
    $d(4)$ with parent $p(1,4) \rightarrow rule2(X,Y) : -p(X,Y), d(Y)$.
    $s(6)$ with parent $f(5,6) \rightarrow rule3(X,Y) : -f(X,Y), s(Y)$.
  - *Replace the rule in the set of the observations:*
    $h(1) : -p(1,2), p(1,4), p(1,5), c(2,3), f(5,6), d(4), s(6). \rightarrow$
    $h(1) : -rule1(1,2), rule2(1,4), p(1,5), rule3(5,6)$.
- **Step 3**:
  - *Recognize Leaf Nodes:*
    $F = \{rule3(5,6)\}$, indeed $rule3(5,6)$ has only one parent $p(1,5)$.
  - *Create the rules -* $rule_{l_i}(a_i, ...a_n) \leftarrow g_i, l_i$*:*
    $rule3(5,6)$ with parent $p(1,5) \rightarrow rule4(X,Y) : -p(X,Y), rule3(Y,Z)$.
  - *Replace the rule in the set of the observations:*
    $h(1) : -rule1(1,2), rule2(1,4), p(1,5), rule3(5,6). \rightarrow$
    $h(1) : -rule1(1,2), rule2(1,4), rule4(5,6)$.
- **Step 4**: END - No more Leaf Nodes can be recognized

The procedure continues with the evaluation of the generated rules, that are:
$rule1(X,Y) : -p(X,Y), c(Y,Z)$. $\quad rule2(X,Y) : -p(X,Y), d(Y)$.
$rule3(X,Y) : -f(X,Y), s(Y)$. $\quad rule4(X,Y) : -p(X,Y), rule3(Y,Z)$.
Now, supposing that $P = 95\%$ and that the scores of the rules are:
$Score1 = 95\%$; $Score2 = 99\%$; $Score3 = 75\%$; $Score4 = 86\%$,
$rule1$ and $rule2$ will be shifting rules, while $rule3$ and $rule4$ will be neglecting rules:
$rule1(X,Y) : -p(X,Y), c(Y,Z)$. $\quad rule2(X,Y) : -p(X,Y), d(Y)$.
$: -f(X,Y), s(Y)$. $\quad : -p(X,Y), rule3(Y,Z)$.

5 Experiments

The proposed methods were implemented in SICStus Prolog, and tested on the learning system INTHELEX with various experiments, whose results are reported in the following. 33 repetitions of each learning task were carried out, in each of which the dataset was randomly split into a training set (including 70% of the observations), exploited also to induce the rules for the abstraction operators) and a test set (made up of the remaining 30%).

Table 1. System performance with the exploitation of the discovered abductive theories

		Without abduction	With abduction Without type domains	With abduction With type domains
Lgg	Min - Max	3 - 13	2 - 8	0 - 2
	Med - StDev	7.72 - 2.08	5.48 - 1.5	1 - 0.66
Claus.	Min - Max	2 - 6	2 - 5	1 - 3
	Med - StDev	4.09 - 1.12	3.18 - 0.95	1.72 - 0.72
Accur.	Min - Max	89% - 100%	94% - 100%	91% - 100%
	Med - StDev	96.24% - 2.27	99.32% - 1.61	98.75% - 3.02
Runtime	Min - Max	3.20 - 13.36	4.98 - 170.36	3.06 - 84.40
	Med	5.16	40.05	24.29

5.1 Exploitation of the Learned Abductive Theories

The first experiment aimed at checking whether the *abducibles* and the *integrity constraints* automatically learned according to the proposed algorithms are effective to allow the abductive procedure implemented in INTHELEX to handle cases of missing information in the observations. The experiments concern the induction of layout-based classification rules for scientific papers belonging to `ICML` series. The available dataset was corrupted by eliminating the 8% of the descriptors for each observation contained in the tuning set. The learning system was applied on this dataset firstly without exploiting the abductive procedure. Successively, the learning process was repeated, allowing the system to exploit its abductive capability and the abduction theory automatically learned. We focused our attention on binary constraints made up of unary and binary predicates. One more experiment was run to test the usefulness of replacing groups of simple integrity constraints belonging to the same type by means of type constraints automatically inferred.

Table 1 reports the system performance in the various cases as regards the amount of performed refinements, lgg's and added clauses, predictive accuracy and runtime (sec). As we can note, the system performance improved with the exploitation of abduction with respect to all parameters except runtime. Actually, runtime increases because of the additional reasoning carried out by the abductive procedure; however, as expected, exploiting the type domains significantly reduces runtime because of the fewer constraints to be taken into account. According to a paired t-test, all differences are statistically significant except the predictive accuracy between the second and third rows. Thus, exploiting the automatically learned abduction theory allows the system to significantly improve its performance in the presence of missing data. The number of theory refinements and learned clauses decreases both using abduction and, even more, when type domains are exploited, indicating that the system was able to correctly complete the corrupted observations without applying the refinement procedure. Noticeably, except for accuracy, also the standard deviation constantly decreases, revealing more stability in the system behavior.

Table 2. System performance exploiting the discovered abstraction theories

	ICML		SVLN		IEEET	
	With Abs	No Abs	With Abs	No Abs	With Abs	No Abs
Lgg	5.81	5.54	7.36	8.12	8.03	8.30
Cl	1.21	1.27	2.75	2.69	2.03	2.27
Accuracy	96.93%	96.75%	86.54%	87.36%	90.69%	90.57%
Runtime	2.00	3.16	11.34	19.46	7.64	27.55

Table 3. Abstraction on ICML logic type components

	Author		Page Number		Title	
	With Abs	No Abs	With Abs	No Abs	With Abs	No Abs
Lgg	8.9	8.96	8.15	8.12	8.81	9.09
Cl	2.33	2.06	2.39	2.45	2.42	2.54
Accuracy	97.18%	97.12%	97.81%	97.54%	98.12%	97.87%
Runtime	14.44	29.07	34.06	76.22	27.70	51.67

5.2 Exploitation of the Abstraction Theories

The second experiment aimed at checking the effectiveness of the abstraction theories learned according to the proposed algorithms. Such rules were provided to INTHELEX, that was allowed to exploit the abstraction operators. The learning tasks involved the induction of classification rules for three classes of scientific papers (96 documents of which 28 for `ICML`, 32 for `SVLN`, 36 for `IEEET`), and of rules for identifying the logical components *Author* [36+, 332-], *Page Number* [27+, 341-] and *Title* [28+, 340-] in the `ICML` papers (in square brackets the number of positive and negative instances for each label are reported). To build neglecting rules, the threshold for considering low significance (i.e. the score near to zero) was empirically set to $P = 5\%$. To build shifting rules that have high significance (i.e. very frequent in positive observations and rarely present in negative observations and *vice versa*) the threshold was empirically set to $P = 95\%$ for the classification task and to $P = 75\%$ for the understanding task.

The average results on the 33 folds, along with the number of refinements and of clauses learned, the predictive accuracy of the learned theories and the runtime (sec), are reported in Tables 2 and 3. According to a paired t-test, there is no statistical difference between the results with and without abstraction, except for runtime. Having the same performance (predictive accuracy) and behavior (no. of clauses and refinements) both with and without abstraction means that the proposed technique was actually able to eliminate superfluous details only, leaving all the information that was necessary for the learning task, which was a fundamental requirement for abstraction. Conversely, runtime was dramatically reduced when using abstraction thanks to the shorter descriptions obtained by eliminating the details, which was exactly the objective of using abstraction.

An example of neglecting rule identified with the proposed strategy is:

```
:- type_graphic(A), pos_lower(A).
```

by which we understand that a graphics being placed in lower position is not discriminant between positive and negative examples. As expected, exploiting the abstraction operators the system learns shorter clauses. For instance, the theory learned for *author* contains two clauses made up of 18 and 15 literals (against the 19 and 37 without using abstraction):

```
logic_type_author(A) :- height_medium_small(A), pos_upper_type_text(A),
   part_of(B, A), part_of(B, C), height_very_small_type_text(C),
   pos_upper_type_text(C), part_of(B, D), width_very_large(D),
   height_smallest(D), type_hor_line(D), pos_center_pos_upper(D),
   alignment_left_col(D, E), on_top(F, E), part_of(B, E), part_of(B, F),
   part_of(B, G), type_text_width_medium_large(G), pos_left_type_text(G).
logic_type_author(A) :- part_of(B, A), part_of(B, C),
   pos_upper_type_text(A), pos_center_pos_upper(A),
   pos_upper_type_text(C), pos_left_type_text(C),
   height_very_very_small_type_text(C), on_top(C, D),
   part_of(B, D), on_top(E, A), width_very_large(E), height_smallest(E),
   pos_center_pos_upper(E), on_top(F, E), alignment_center_col(F, E).
```

where the presence of several abstract predicates confirms that the automatically generated abstraction theory was able to identify discriminative intermediate concepts. An example of shifting rule learned (and exploited above) is:

```
pos_upper_type_text(A) :- type_text(A), pos_upper(A).
```

6 Conclusion and Future Works

This paper presented a technique for automatically inferring meta-information needed to apply abduction and abstraction operators in and inductive learning framework, exploiting the same observations that are input to the inductive algorithm. Application of the proposed technique in a real learning system proved their viability for learning from incomplete observations without loosing predictive accuracy and for significantly improving learning time in complex real-world domains. Future work will concern a deeper investigations of which properties can be considered significant to infer integrity constraints for abduction, development of strategies to improve the generation of abductive theories, and design of techniques that can provide information for further abductive operators.

References

[1] P.R. Cohen and E.A. Feigenbaum, editors. *The Handbook of Artificial Intelligence*, volume 3. Morgan Kaufmann, 1981.

[2] L. De Raedt. *Interactive Theory Revision - An Inductive Logic Programming Approach*. Academic Press, 1992.

[3] Y. Dimopoulos, S. Džeroski, and A. Kakas. Integrating explanatory and descriptive learning in ILP. In *Proceedings of IJCAI97*, pages 900–906, 1997.

[4] Y. Dimopoulos and A. Kakas. Abduction and learning. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 144–171. IOS Press, 1996.

[5] F. Esposito, S. Ferilli, N. Fanizzi, T.M.A. Basile, and N. Di Mauro. Incremental multistrategy learning for document processing. *Applied Artificial Intelligence: An Internationa Journal*, 17(8/9):859–883, 2003.
[6] F. Esposito, E. Lamma, D. Malerba, P. Mello, M. Milano, F. Riguzzi, and G. Semeraro. Learning abductive logic programs. In *Proceedings of the ECAI96 Workshop on Abductive and Inductive Reasoning*, 1996.
[7] S. Ferilli, F. Esposito, T.M.A. Basile, and N. Di Mauro. Automatic induction of first-order logic descriptors type domains from observations. In Rui Camacho, Ross D. King, and Ashwin Srinivasan, editors, *ILP*, volume 3194 of *LNCS*, pages 116–131. Springer, 2004.
[8] P.A. Flach and N. Lachiche. Confirmation-guided discovery of first-order rules with `Tertius`. *Machine Learning*, 42(1/2):61–95, 2001.
[9] A. Giordana, D. Roverso, and L. Saitta. Abstracting concepts with inverse resolution. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 142–146, Evanston, IL, 1991. Morgan Kaufmann.
[10] A.C. Kakas, R. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6), 1993. 718–770.
[11] A.C. Kakas and P. Mancarella. Generalized stable models: a semantics for abduction. In *Proceedings of ECAI90*, pages 385–391. Pitman Publishing, 1990.
[12] A.C. Kakas and P. Mancarella. On the relation of truth maintenance and abduction. In *Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence*, Nagoya, Japan, 1990.
[13] J-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *ILP91*, pages 107–126, 1991.
[14] E. Lamma, P. Mello, M. Milano, F. Riguzzi, F. Esposito, S. Ferilli, and G. Semeraro. Cooperation of abduction and induction in logic programming. In A. Kakas and P. Flach, editors, *Abductive and Inductive Reasoning: Essays on their Relation and Integration*. Kluwer, 2000.
[15] J. W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, second edition, 1987.
[16] R.S. Michalski. Inferential theory of learning. developing foundations for multistrategy learning. In R.S. Michalski and G. Tecuci, editors, *Machine Learning. A Multistrategy Approach*, volume IV, pages 3–61. Morgan Kaufmann, 1994.
[17] S.H. Muggleton and L. De Raedt. Inductive logic programming. *Journal of Logic Programming: Theory and Methods*, 19:629–679, 1994.
[18] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26(2):99–146, 1997.
[19] C. Rouveirol and J. Puget. Beyond inversion of resolution. In *Proceedings of ICML97*, pages 122–130, Austin, TX, 1990. Morgan Kaufmann.
[20] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.
[21] A. Srinivasan. The aleph manual version 4, 2003. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/.
[22] P.E. Utgoff. Shift of bias for inductive concept learning. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, volume II, pages 107–148. Morgan Kaufmann, Los Altos, CA, 1986.
[23] J.-D. Zucker. Semantic abstraction for concept representation and learning. In R. S. Michalski and L. Saitta, editors, *Proceedings of the 4th International Workshop on Multistrategy Learning*, pages 157–164, 1998.