Machine Learning, 38, 133–156, 2000.

Multistrategy Theory Revision: Induction and Abduction in INTHELEX

FLORIANA ESPOSITO esposito@di.uniba.it
GIOVANNI SEMERARO semeraro@di.uniba.it
NICOLA FANIZZI fanizzi@di.uniba.it
STEFANO FERILLI ferilli@di.uniba.it
Dipartimento di Informatica, Universitá degli Studi di Bari, Via E. Orabona 4, 70126 Bari, Italy

Editors: Ryszard Michalski & Lorenza Saitta

Abstract. This paper presents an integration of induction and abduction in INTHELEX, a prototypical incremental learning system. The refinement operators perform theory revision in a search space whose structure is induced by a quasi-ordering, derived from Plotkin's θ-subsumption, compliant with the principle of Object Identity. A reduced complexity of the refinement is obtained, without a major loss in terms of expressiveness. These inductive operators have been proven *ideal* for this search space. Abduction supports the inductive operators in the completion of the incoming new observations. Experiments have been run on a standard dataset about family trees as well as in the domain of document classification to prove the effectiveness of such multistrategy incremental learning system with respect to a classical batch algorithm.

Keywords: incremental learning, induction, abduction, object identity, theory revision

1. Introduction and motivations

Automatic revision of logic theories is a complex and computationally expensive task. In fact, most systems for theory revision deal with propositional logic and try to modify an existing incomplete or incorrect theory to fit a set of preclassified training examples. They can integrate different reasoning methods and learning strategies. Among these systems we find RTLS (Ginsberg, 1990), DUCE (Muggleton, 1987), DUCTOR (Cain, 1991) and, finally, the system proposed by Matwin & Plante (1991) and EITHER (Mooney & Ourston, 1994), which explicitly use a deductive-inductive method for modifying a given domain theory. Empirical results have shown that revising an imperfect theory results in definitions that are more accurate from fewer examples than pure induction. Moreover, some theoretical results have been obtained by Mooney (1995) in analyzing theory revision by the formalization of the concept of "closeness" of the initial theory to the correct one and in discussing the problem of computational complexity in the propositional Horn-clause theory revision.

In the machine learning literature, there are also a number of systems that can revise first-order theories. Most of these systems try to limit the search space by exploiting information and, generally, require a wide, although incomplete, domain theory or a deep knowledge acquired (possibly interactively) from the user: the consequence is that the revised theory cannot be very dissimilar from the initial one. Some of them, such as MIS (Shapiro, 1983)

and CLINT (De Raedt, 1992), strongly rely on the interaction with the user to reduce the search space. Others, such as WHY (Saitta, Botta, & Neri, 1993), TRACEY (Bergadano & Gunetti, 1994) and KBR (Botta, 1994), do not require any interaction with the user during the induction process and adopt sophisticated search strategies or more informative search structures. Other systems for theory revision, such as FORTE (Richards & Mooney, 1991), and AUDREY (Wogulis, 1991), do not allow negative information items (negative literals) to be expressed in the theories because of computational complexity considerations. Therefore, half of the whole search space is not explored.

Many of the cited first order logic learning systems adopt multistrategy approaches integrating several types of inferential mechanisms: in particular, two important strategies to perform hypothetical reasoning (i.e., inferences from incomplete information) are induction and abduction. Induction means inferring from a certain number of significant observations regularities and laws valid for the whole population. Abduction, or "inference to the best explanation", is a kind of theory-forming or interpretive inference. In fact, it goes from the data to a hypothesis that accounts for the data (Josephson & Josephson, 1996). Among the systems that integrate different types of inferences are WHY, CLINT and AUDREY. WHY is a knowledge intensive system which, besides the target knowledge, uses a causal, a phenomenological and a control knowledge. It adopts an inference engine combining several kinds of inferences and is able to acquire and refine a target knowledge base. CLINT uses different types of inferences (abduction, induction, constructive induction by analogy, shifting bias) in a multivalued logic framework.

This paper presents INTHELEX, a first order logic learning system, which limits the search space by exploiting a bias on the generalization model. The system not necessarily needs an initial theory; this is a fundamental issue, since in many cases deep knowledge about the world is not available. The bias consists of the adoption of *θ-subsumption under Object Identity* (Esposito et al., 1996a) as a generalization model, which allows the definition of refinement operators meeting the condition of *ideality*. Ideality is satisfied when three desirable properties are fulfilled: *Local finiteness*—the operator must find the refinement, when existing, in a finite number of steps—, *completeness*—if a refinement exists, the operator must find it—and *properness*—a refinement must not be equivalent to the original knowledge.

The approach may be defined a "multistrategy" one since the system realizes the combination of two different forms of reasoning in the same symbolic paradigm: induction and abduction. As logical inference operations, both induction and abduction embody a form of "reversed" deduction: together with the background knowledge, the hypotheses should entail a given set of observations. The difference is in the way they satisfy and use this requirement. Induction is used to synthesize the information contained in the observations into a hypothesis that can account for the whole set of observations in a common way and, in addition, for other new observations. On the other hand, abduction is used to generate a case for the truth of the observation in terms of hypotheses that are typically specific for the situation and individual objects at hand. Hence, typically, an abductive hypothesis consists of facts further describing the situation at hand, while an inductive hypothesis consists of general rules pertaining to a whole class of situations.

The major differences of INTHELEX with respect to similar systems working in the refinement of first order theories, are in its incremental nature, in the reduced need of a deep background knowledge, since the system is able to infer a theory from scratch, continuously revising it on the grounds of new evidences (both positive and negative), and in the peculiar bias on the generalization model, which reduces the search space and does not limit the expressive power of the adopted representation language.

The plan of the paper is the following. In the next section, we give the basic definitions of the language we use throughout this paper, together with the framework induced by the concept of Object Identity. Then, in Section 3 the inductive refinement operators of INTHELEX are introduced and, in Section 4, the abductive reasoning procedure is presented and the way in which abduction has been integrated in the system is discussed. Experiments on a standard dataset about family relationships as well as in the real world task of document classification, performed with INTHELEX, are described and discussed in Section 5. Section 6 contains related work and, finally, in Section 7, we present some conclusions and outlines for future work.

2. Basic notions and definitions

First, we recall some basic definitions (a complete reference can be found in (Lloyd, 1987)):

- *Substitution*: a finite set of the form $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$, where each x_i is a variable, each t_i is a term distinct from x_i and the variables $x_1, \ldots, x_n$ are distinct.
- Given an n-ary predicate symbol p and n terms $t_1, \ldots, t_n$, we define $A = p(t_1, \ldots, t_n)$ an *atom*; A and its negation $\neg A$ are, respectively, a *positive* and a *negative literal*.
- A *clause* is a disjunction of literals where the variables are assumed to be universally quantified; the positive literals make up the *head* of the clause, while the negative ones make up its *body*. We will indifferently use the set notation and the Prolog notation.
- A clause is called definite clause if it contains one positive literal; a finite set of definite clauses is called a definite program.
- A *program clause* is a clause of the form: A :- $L_1, \ldots, L_n$, where A is an atom and $L_1, \ldots, L_n$ are literals; a *normal program* is a finite set of program clauses.

In the paper, we are only concerned with logic theories expressed as *hierarchical programs*, that is, as programs for which it is possible to find a *level mapping* (Lloyd, 1987) such that, in every program clause $p(t_1, t_2, \ldots, t_n)$:- $L_1, L_2, \ldots, L_m$, the level of any predicate symbol occurring in the body is less than the level of p. Such programs avoid any kind of recursion.

Another constraint on the representation language is that, whenever we write about clauses, we mean *Datalog linked* clauses. Here, we refer to (Ceri, Gottlob, & Tanca, 1990; Kanellakis, 1990) for the basic notions about Datalog. Informally, Datalog is a function-free first order logic language (the only allowed terms are constants and variables). An argument of a literal in a Datalog clause is linked if either the literal is the head of the clause or another argument in the same literal is linked; in turn, a literal is linked if at least one of its arguments is linked; lastly, a Datalog clause is *linked* if all of its literals are linked (Helft, 1987).

As an example, a linked clause is

$$C \equiv \textit{bicycle(X) :- wheel(Y,X), tire(Z,Y), rim(W,Y).}$$

Conversely, the clauses $D \equiv$ *bicycle(X):- tire(Z,Y), rim(W,Y)* and $F \equiv$ *bicycle(X):- tire(Z,Y), rim(W,V)* are not linked. Indeed, *tire(Z,Y), rim(W,Y)* are not linked in D, whereas *rim(W,V)* is not linked in F.

In our framework we adopted the *single representation trick*, according to which the same representation language is exploited both for examples and for hypotheses. The differences existing between them are the following:

- each example is represented by one ground clause with a unique literal in the head;
- each hypothesis is a set of program clauses with the same head.

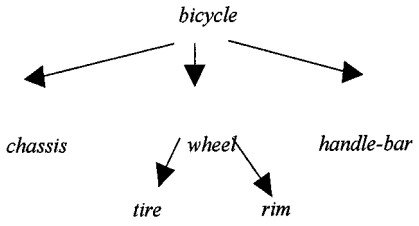
An *example* E is *positive* for a hypothesis H if its head has the same predicate letter and sign as the head of the clauses in H. The example E is *negative* for H if its head has the same predicate, but opposite sign.

Specifically, examples cannot contain variables or negated literals in the body, while clauses of a hypothesis can contain negated literals in the body.

It is possible to organize the concepts in the description language according to the following structure:

Definition (Dependency Graph). A *dependency graph* is a directed acyclic graph of predicate letters, where an edge (p, q) indicates that atoms with predicate letter q are allowed to occur in the hypotheses defining the concept denoted by p.

Since we are dealing with hierarchical theories, the dependency graph consists of a directed acyclic graph of concepts (represented by the corresponding predicate letters), in which parent nodes are assumed to be dependent on their offspring (in other words, offspring can concur to the definition of their parent).
For example, the following tree represents the concept hierarchy for the induction of the concept *bicycle*:



An instance of a positive example is:

$$E \equiv \textit{bicycle(b)} \text{ :- } \textit{wheel(r1,b), wheel(r2,b), saddle(s,b), handle}-\textit{bar}(h, b).$$

which means "*b* is a bicycle *since* it has (at least) two wheels, *r1* and *r2*, a saddle *s* and a handle-bar h". A negative example could be:

$N \equiv$ *not(bicycle(b))* :- *wheel(w1,m), wheel(w2,m), engine(e,m).*

to be read as "*m* is not a bicycle, *since* it has two wheels *w1* and *w2* and an engine *e*". Furthermore, the following could be a clause in a hypothesis:

$C \equiv$ *bicycle*(X) :- *wheel(Y,X), wheel(Z,X), not(engine(W,X)).*

to be interpreted as "Something that has two wheels and has not an engine is a bicycle".

Here, we introduce a logic language, called *Datalog*^{OI}, whose basic notion is that of *object identity*. This notion dates from the Machine Learning early years (Vere, 1975; Hayes-Roth & McDermott, 1978); we recall here the definition previously given in (Esposito et al., 1996a).

Definition (Object Identity). Within a clause, terms denoted with different symbols must be distinct.

An effect of introducing such a notion is the modification of the standard quasi-ordering upon Datalog clauses: indeed, the algebraic structure of the search space changes, with consequences on the properties of the refinement operators that can be defined in it (van der Laag & Nienhuys-Cheng, 1994). Moreover, it provokes the extension of the axiom set of the underlying equational theory—in our case Clark's Equality Theory (CET) (Lloyd, 1987)—with the addition of the following rewrite rule:

$\forall C \in L, \forall (t, s) \in \text{terms}(C) \times \text{terms}(C)$, if t and s are distinct then $t \neq s \in \text{body}(C)$ where L denotes the language of all the possible Datalog clauses built from a finite number of predicates and *terms(C)* denotes the set of terms in the clause C.

This new equational theory affects the inference rules of the calculus (resolution, factorization and paramodulation) (Plotkin, 1972) in order to cope with the *negation-as- failure* rule (Clark, 1978).

We denote with C_{OI} the result of adding the inequality literals, coming from this axiom, to a clause C. For example, the linked clause C in Datalog form, given above, i.e.,

$C \equiv$ *bicycle(X)* :- *wheel(Y,X), wheel(Z,X), not(engine(W,X)).*

leads to:

$$C_{OI} \equiv \textit{bicycle(X)} \text{ :- } \textit{wheel(Y,X), wheel(Z,X), not(engine(W,X))}, \\ X \neq Y, X \neq Z, X \neq W, Y \neq Z, Y \neq W, Z \neq W.$$

In spite of such language bias, Datalog^{OI} has the same expressive power as Datalog (Semeraro et al., 1998), as stated by the two following results, where $T_P \uparrow \omega$ denotes the Fixpoint characterization of the Least Herbrand Model (Lloyd, 1987):

Proposition. $\forall C \in$ *Datalog* $\exists C' = \{C_1, C_2, \ldots, C_n\} \subseteq$ *Datalog*^{OI}: $T_C \uparrow \omega = T_{C'} \uparrow \omega$, *i.e., for each Datalog clause we can find a set of Datalog*^{OI} *clauses which is equivalent to it.*

Corollary. $\forall P \subseteq$ *Datalog* $\exists P' \subseteq$ *Datalog*^{OI} : $T_P \uparrow \omega = T_{P'} \uparrow \omega$, *i.e., for each Datalog program we can find a Datalog*^{OI} *program which is equivalent to it.*

A generalization model is required to order the clauses in our representation language:

Definition Let C, D be two clauses of a language L and $\leq$ an ordering relation defined on L. If $C \leq D$ we say that *D is more general than or equivalent to C*. We write $C < D$ when $C \leq D$ and *not*$(D \leq C)$ and we say that *D is more general than C* or *C is more specific than D*. We write $C \sim D$, and we say that C and D are *equivalent clauses*, when $C \leq D$ and $D \leq C$.

Other equivalent terminology, more appropriate in a theory revision framework, is that *D is an upward refinement of C* or *C is a downward refinement of D* for $C \leq D$, and that *D is a proper upward refinement of C* or *C is a proper downward refinement of D* for $C < D$.

For example, in the ordering induced by the classical relation of θ-subsumption:

$C \equiv$ *bicycle(X) :- wheel(Y,X), wheel(Z,X), diameter(R,Y), diameter(S,Z), less-or-equal(R,R).*

is a proper downward refinement of:

$D \equiv$ *bicycle(X) :- wheel(Y,X), wheel(Z,X), diameter(R,Y), diameter(S,Z), less-or-equal(R,W), less-or-equal(W,V), less-or-equal(V,R).*

and C is equivalent to:

$E \equiv$ *bicycle(X) :- wheel(Y,X), wheel(Z,X), diameter(R,Y), diameter(S,Z), less-or-equal(R,R), less-or-equal(R,T), less-or-equal(T,R).*

We can now introduce the notion of refinement operators that perform the search for revisions in a given search space.

Definition (Downward/Upward Refinement Operators). Given a quasi-ordered set $(\mathbf{L}, \leq)$, a *refinement operator* is a mapping from $\mathbf{L}$ to $2^{\mathbf{L}}$, such that:
$\forall C \in \mathbf{L}$: $\rho(C) \subseteq \{D \in \mathbf{L} \mid D \leq C\}$ (*downward refinement operator*),
i.e., it computes a subset of the overall downward refinements of C
$\forall C \in \mathbf{L}$: $\delta(C) \subseteq \{D \in \mathbf{L} \mid C \leq D\}$ (*upward* refinement operator),

i.e., it computes a subset of the overall upward refinements of C

We can now apply the closure operator to these functions:

Definition (Closure of a Refinement Operator). Let τ be a refinement operator and C a clause in **L**, then the *closure* of τ (in symbols τ^*) is

$$\tau^*(\mathrm{C}) = \bigcup_{\mathrm{n}\geq 0} \tau^{\mathrm{n}}(\mathrm{C}) = \tau^0(\mathrm{C}) \cup \tau^1(\mathrm{C}) \cup \cdots \cup \tau^{\mathrm{n}}(\mathrm{C}) \cup \cdots$$

where $\tau^n(C)$ is inductively defined as follows:

$$\tau^0(C) = \{C\}$$
$$\tau^n(C) = \{D \mid \exists E \in \tau^{n-1}(C) \text{ and } D \in \tau(E)\}$$

Definition (Locally Finite, Proper, Complete, Ideal Refinement Operator). Given a quasi-ordered set $(\mathbf{L}, \leq)$:
ρ (resp. δ) is *locally finite* iff $\forall C \in \mathbf{L} : \rho(C)$ (*respectively* $\delta(C)$) is finite and computable
ρ (resp. δ) is *proper* iff $\forall C \in \mathbf{L}: \rho(C) \subseteq \{D \in \mathbf{L} \mid D < C\}$ $(\delta(C) \subseteq \{D \in \mathbf{L} \mid C < D\})$
i.e., the refinement is not equivalent to the original clause
$\rho(\delta)$ is *complete* iff $\forall C, D \in \mathbf{L}$, if $D < C$ then $\exists E$ such that (s.t.) $E \in \rho^*(C)$ and $E \sim D$
(if $C < D$ then $\exists E$ s.t. $E \in \delta^*(C)$ and $E \sim D$)
i.e., whenever a refinement exists it is able to find a clause that is equivalent to it.
$\rho(\delta)$ is *ideal* iff it is locally finite, proper and complete.

An ideal operator ensures the efficiency and effectiveness of the refinement process, since it computes, with limited resources—both in time and in space (local finiteness)—, all (completeness) the actual refinements (properness) of a given clause.

By applying object identity to the classical θ-subsumption ordering, we obtain a new ordering relation called θ-subsumption under object identity—*θ_{OI}-subsumption*—upon the set of Datalog clauses (Esposito et al., 1996a). This ordering defines the new search space in which our operators shall work.

Let us define, now, the generalization model used in our learning framework:

Definition (θ_{OI}-subsumption ordering). Let C, D be two Datalog clauses. We say that D *θ-subsumes C under object identity* (or D *θ_{OI}-subsumes* C) if and only if (iff) there exists a substitution σ s.t. $D_{OI} \cdot \sigma \subseteq C_{OI}$. The ordering induced by θ_{OI}-subsumption is denoted by $\leq_{OI}$.

The θ_{OI}-subsumption ordering, $\leq_{\mathrm{OI}}$, has been proven to be a quasi-ordering upon the space of the Datalog clauses (Semeraro et al., 1998).

Looking back at the previous example, in the $\leq_{OI}$ ordering D is not comparable to C which, in turn, is a proper upward refinement of E. Only clauses obtained by a renaming substitution are equivalent.

Generally, the canonical inductive paradigm requires the synthesized theory to fulfill the properties of completeness and consistency with respect to a given set of examples. In case these properties do not hold, suitable refinement operators must be applied to restore them. More formally, we introduce the following definitions, where E^- and E^+ denote the sets of all the available negative and positive examples, respectively.

Definition (Inconsistency and Incompleteness). A *theory* T is *inconsistent* iff $\exists H \in T$, $\exists N \in E^-$: H is inconsistent wrt N.
A *hypothesis* H is *inconsistent* wrt N iff $\exists C \in H$: C is inconsistent wrt N.
A *clause* C is *inconsistent* wrt N iff $\exists\sigma$:
1) $\text{body}(C_{OI})\cdot\sigma \subseteq \text{body}(N_{OI})$ 2)$\neg$ $\text{head}(C_{OI})\cdot\sigma = \text{head}(N_{OI})$.

If at least one of the two conditions above is not met, we say that C is *consistent* wrt N.

A *theory* T is *incomplete* iff $\exists H \in T$, $\exists P \in E^+$: H is incomplete wrt P.
A *hypothesis* H is *incomplete* wrt P iff $\forall C \in H$: not($P \preceq_{OI} C$). Otherwise, it is *complete* wrt P.

When an inconsistent (incomplete) hypothesis is detected, a specialization (generalization) of the hypothesis is required in order to restore this property of the theory. In the former case, all the inconsistent clauses have to be revised by means of a downward refinement operator; in the latter, the whole hypothesis has to be refined by means of an upward refinement operator that generalizes an existing clause or introduces new ones.

In a logic framework for the revision of Datalog theories from facts, a fundamental problem is the definition of *ideal* refinement operators. Indeed, when the aim is to develop incrementally a logic theory, that should be *correct* with respect to the *intended model*, i.e. the target knowledge base, at the end of the development process, it becomes relevant to define operators that allow a stepwise (incremental) refinement of *too weak* or *too strong* theories (Komorowski & Trcek, 1994). The *ideality* of the refinement operators plays a key-role when the efficiency and the effectiveness of the revision process is an unnegligible requirement. Unfortunately, when full Horn clause logic is chosen as representation language and either θ*-subsumption* or *implication* is adopted as generalization model, there exist no ideal refinement operators (van der Laag & Nienhuys-Cheng, 1994). On the contrary, they do exist under the weaker, but more mechanizable and manageable, ordering induced by θ_{OI}*-subsumption*, as proved in (Esposito et al., 1996a).

For these reasons, ideal operators are indeed useful notwithstanding some problems waiting to be solved in the incremental learning paradigm. Revising a theory through refinement operators cannot answer for the quality of the induced hypotheses, since there could be different theories, with the same logic consequences, but having different syntactic characteristics such as compactness as well as human comprehensibility. Moreover, difficult problems concern also noisy or incomplete information coming from the environment. This is another line along which refinement operators should be developed that requires the adoption of other theory revision strategies.

Let us now briefly discuss the operators implemented in INTHELEX. The upward refinement operator δ_{OI} extends the concept of *least general generalization* (lgg) introduced by Plotkin (1970) to cope with the ordering induced by θ_{OI}-subsumption.

Definition (Least general generalization under object identity). A least general generalization under θ_{OI}-subsumption of two clauses is a generalization which is not more general than any other such generalization, that is, it is either more specific than or not comparable to any other such generalization.
Formally:

$$\mathrm{lgg}_{\mathrm{OI}}(C_1, C_2) = \{C \mid C_i \leq_{\mathrm{OI}} \mathrm{C},\ i = 1, 2 \text{ and } \forall D \text{ s.t. } C_i \leq_{\mathrm{OI}} D,\ i = 1, 2\text{: not}(D <_{\mathrm{OI}} C)\}$$

For example, given

$C_1 \equiv$ *bicycle(a):- wheel(v,a), wheel(w,a), radius(r,v), radius(s,w).*
$C_2 \equiv$ *bicycle(b):- wheel(v,b), wheel(Y,b), radius(R,Y).*

we have:

$$\mathrm{lgg}_{\mathrm{OI}}(C_1, C_2) = \{\textit{bicycle(X):- wheel(v,X), wheel(Z,X), radius(T,Z).}\}$$

while the clause $E \equiv$ *bicycle(X):- wheel(v,X), wheel(Z,X).* is a generalization but not a least general one.

The downward refinement operator ρ_{OI} relies on the addition of a non-redundant literal to a clause that turns out to be inconsistent with respect to a negative example, in order to restore the consistency property of the clause.

We can formally define the search space as the partially ordered set (*poset*) $(\mathbf{L}/\sim_{\mathrm{OI}}, \leq_{\mathrm{OI}})$, where $\mathbf{L}/\sim_{\mathrm{OI}}$ is the quotient set of the Datalog linked clauses and $\leq_{\mathrm{OI}}$ is the quasi ordering relation defined above, which can be straightforwardly extended to equivalence classes under $\sim_{\mathrm{OI}}$.

3. Inductive refinement operators in INTHELEX

INTHELEX (INcremental THEory Learner from EXamples) is a learning system for the induction of hierarchical theories from examples. INTHELEX is *fully incremental*: this means that, in addition to the possibility of taking as input a previously generated version of the theory, learning can also start from an empty theory and from the first available example. A partial investigation on the way the knowledge base evolves depending on the initial theory quality will be one of the aims of the experiments described later, where INTHELEX will work both from scratch and starting from a pre-existing theory. INTHELEX can learn simultaneously various concepts, possibly related to each other; furthermore, it is a *closed loop learning system*—i.e. a system in which the learned theory is checked to be valid on

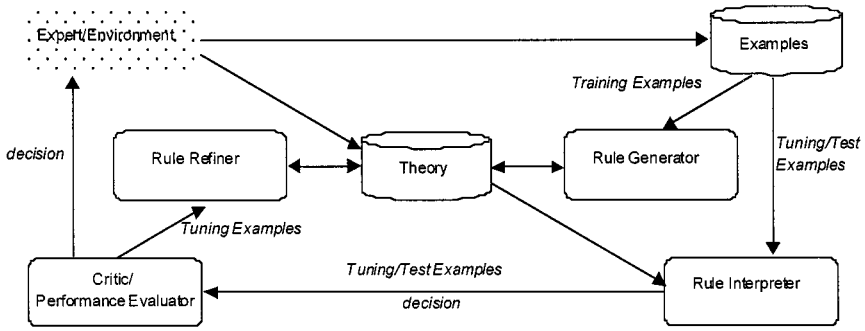


Figure 1. The architecture of INTHELEX.

any new example —, and in case of failure, a revision process is activated on it, in order to restore the completeness and consistency properties.

Incremental learning is necessary when either incomplete information is available at the time of initial theory generation, or the nature of the concepts evolves dynamically. The latter situation is the most difficult to handle since time evolution needs to be considered. In any case, it is useful to consider learning as a *closed loop* process, where feedback on performance is used to activate the theory revision phase (Becker, 1985).

INTHELEX learns theories, expressed as sets of Datalog^{OI} clauses, from positive and negative examples. It adopts a full memory storage strategy (Reinke & Michalski, 1985)—i.e., it retains all the available examples, thus the learned theories are guaranteed to be valid on the whole set of known examples—and it incorporates two refinement operators, one for generalizing hypotheses that reject positive examples, and the other for specializing hypotheses that explain negative examples.

Formally, we give the following definition regarding theories:

Definition (Answer set). Given a theory, its *answer set* is the set of examples it accounts for.

Both the operators of our system, when applied, change the answer set of the theory. Therefore, INTHELEX is a system for theory revision rather than for *theory restructuring*, according to the definition of theory restructuring as a process that does not change the answer set of a theory (Wrobel, 1996).

The architecture of INTHELEX is shown in figure 1. INTHELEX takes a set of examples of the concepts to be learned from the *Expert/Environment*. This set can be subdivided into three subsets, namely *training*, *tuning*, and *test examples*, according to the way in which examples are exploited during the learning process. Specifically, training examples, previously classified by the Expert, are exploited by the *Rule Generator* to generate a theory that is able to explain the provided examples. The initial theory can also be provided by the *Expert/Environment*. Subsequently, the theory is used by the *Rule Interpreter* to verify that the theory continues to be valid even when new examples become available. The Rule Interpreter takes

the set of inductive hypotheses and a tuning/test example as input and produces a decision. The *Critic/Performance Evaluator* compares the decision produced by the Rule Interpreter to the correct one. In case of incorrectness, it can locate the cause of the wrong decision and is able to choose the proper kind of correction, firing the theory revision process. In such a case, tuning examples are exploited incrementally by the *Rule Refiner* to modify incorrect hypotheses according to a *data-driven* strategy. The Rule Refiner consists of two distinct modules, a *Rule Specializer* and a *Rule Generalizer*, which attempt to correct too weak and too strong hypotheses, respectively. Test examples are exploited to prove the predictive capabilities of the theory, intended as the behavior of the theory on new observations.

Let us now briefly summarize the process of logic theory revision. A thorough description of such process can be found in (Semeraro et al., 1995). In order to perform its task, INTHELEX uses a previous theory (optional), a graph describing the dependence relationships among the concepts to be learned, and a historical memory of all the past (positive and negative) examples that led to the current theory. It is important to note that a positive example for a concept is not considered as a negative example for all the other concepts (unless it is explicitly stated). This allows us to achieve more generality, since we do not know if the dependency graph represents a "part of" or an "is a" hierarchy.

As regards the discussion of the operators that we have implemented in the learning system, we will describe the phases that make up the refinement process, namely saturation of the incoming examples, and tuning (through the generalization and specialization algorithms) of the current theory.

Preliminarily, a *saturation phase* is performed. Whenever a new example is taken into account, it is first checked to see if other concepts in the dependency graph can be recognized in its description, in which case literals concerning those concepts are added (opportunely instantiated) to the body of the clause that represents the example.

For instance, if we have the example

computer(c) :- has_monitor(c, m), has_printer(c, p),
has_central_processing_unit(c, u),
part_of(c, k), has_keys(k, 102), italian(k).

and the following rule in the theory

keyboard(X) :- has_keys(X, Y), italian(X).

we can add the literal *keyboard(k)* to the body of the example, by saturation.

The successive *tuning phase* concerns the possible revision of the current theory with respect to the new examples. This tuning has no effect on the theory if the new example is negative and not covered or positive and covered; in all the other cases, the theory needs to be revised.

In particular, when a positive example is not covered, a generalization of the theory is needed, starting from the current theory, the misclassified example along with the base of the stored examples. It ends with a revised new theory after storing the problematic example in the base of processed examples. Figure 2 shows a pseudo-code description of this process.

```
Procedure Generalize (E: positive example, T: theory, M: negative examples);
L := list of the clauses in the definition of the concept which E refers to
while not generalized and L ≠ ∅ do
    Select from L a clause C for generalization
    Remove from C all negative literals
    L' := lgg_OI(C, E)                          (* L' list of the least general generalizations*)
    for each G in L' do Add to G all the negative literals previously removed from C
    Remove from L' all generalizations without head or not linked or exceeding the generality limit
    Sort L' by decreasing size_OI
    while not generalized and L' ≠ ∅ do
        Select next best generalization C' from L'
        if T after replacing C with C' is consistent wrt M then
            Implement C' in T
    Skip to the next clause in the concept definition
if not generalized then
    C' := E with constants turned into variables
    if T after replacing C with C' is consistent wrt M then
        Implement C' in T
    else
        Implement E in T
```

Figure 2. The generalization algorithm.

First of all, the system performs a step of *blame assignment*: a clause to be generalized is chosen from those making up a concept description (we plan to find a heuristic for such a choice, at this moment the function selects the next clause in the definition—see future work). This functionality is specific of the Critic in figure 1 that was previously discussed. Then, the system tries to compute the least general generalization under object identity (lgg_{OI}) of this clause and the example.

The algorithm that computes the set of the least general generalizations under θ_{OI}-subsumption of two any Datalog$^{\mathrm{OI}}$ clauses is a straightforward extension to Datalog$^{\mathrm{OI}}$ of a similar algorithm given by Plotkin (1970). This extension is necessary since the space of Datalog clauses is not a lattice when ordered by θ_{OI}-subsumption (Semeraro et al., 1998), while it is a lattice when ordered by θ-subsumption (Plotkin, 1970).

If one of the $\mathrm{lgg}_{\mathrm{OI}}$'s is consistent with all the past negative examples, then it replaces the chosen clause in the theory, or else a new clause is chosen to compute the $\mathrm{lgg}_{\mathrm{OI}}$. If no clause in an incomplete hypothesis of a theory can be generalized so that the resulting theory is complete and consistent (and does not exceed a generality limit, computed on the ground of the notion of $size_{OI}$) (Esposito et al., 1996a), the system checks if the example itself, with the constants properly turned into variables, is consistent with the past negative examples. Such a clause is added to the theory, or else the example itself is added as an exception to the theory. These functions are provided by the Rule Refiner module depicted in figure 1.

When, on the other hand, a negative example is covered, a specialization of the theory must be performed. Starting from the current theory, the misclassified example and the base of the processed examples, the specialization algorithm outputs a revised new theory after storing the example that fired the revision process in the base of examples. The pseudo-code procedure describing the algorithm above is given in figure 3.

Among the program clauses occurring in the SLD-derivation of the example, INTHELEX tries to specialize one at the lowest possible level (which corresponds to a clause defining

```
Procedure Specialize (E: negative example; T: theory; M: positive examples);
while exists a (new) derivation D of E from T do
    Sort the list of input program clauses in D by depth in derivation;
    while not specialized and exists another derivation C in the list above do
        P := ρOI_pos(C)
        while P ≠ ∅ and not specialized do
            Select next best specialization C' from P;
            if T after replacing C with C' is complete wrt M then
                Implement C' in T
    if not specialized then
        N := ρOI_neg(C);
        while N ≠ ∅ and not specialized do
            Select next best specialization C' from N
            if T after replacing C with C' is complete wrt M then
                Implement C' in T
    if not specialized then
        Implement E in T as an exception
```

Figure 3. The specialization algorithm.

a concept whose level mapping is the lowest, according to the dependency graph), in order to refine the concepts that are used in the definitions of other concepts, by adding to it one (or more) positive literal(s), which can discriminate all the past positive examples from the current negative one (again, currently this selection is randomly performed). The space in which such a literal should be searched for is potentially infinite and, in any case, its size is so large that an exhaustive search is infeasible. Thus, this operator is able to restrict the search within that portion of the space containing the solution. We decided to specialize lower level concepts since they may appear in the definition of many higher level concepts; the underlying heuristic is that the latter will hopefully benefit from the revised/better definition of the former.

In case of failure, it tries to add the negation of a literal, which discriminates the negative example from all the past positive ones, to the first clause of the SLD-derivation (related to the concept the example is an instance of). In our framework, specializing means merely adding a proper literal to an inconsistent clause, in order to avoid it covering a negative example.

This means that revisions performed by this operator are always minimal (Wrobel, 1994), since all clauses in the theory contain only variables as arguments (a specialization obtained by turning a variable into a constant is not provided for). Moreover, this operator has been proven to be ideal (Esposito et al., 1996a), i.e. locally finite, proper and complete according to the definition given by Nienhuys-Cheng & van der Laag (1994), in the space of constant-free clauses.

If none of the clauses obtained makes the theory complete and consistent again, then INTHELEX adds the negative example to the theory as an exception. An exception contains an exact description of the observation it represents, as it occurs in the tuning set; new incoming observations are always checked with respect to the exceptions before the rules of the related concept. This does not lead to rules which do not cover any example, since exceptions refer to specific objects, while rules contain variables, so they are still applicable to other objects than those in the exceptions.

It is worth noting that, in analogy with human behavior, INTHELEX never rejects examples, but always refines the theory. Moreover, it does not need to know *a priori* what is the whole set of concepts to be learned, but it learns a new concept as soon as positive/negative examples about it are available. Before definitively incorporating the candidate refinements of the rules, their correctness (in conjunction with the overall set of rules) with respect to the entire set of examples is always checked.

4. Integration of abduction in INTHELEX

In this section, we discuss the role that abduction plays in our theory revision framework. After defining a general paradigm that is common to both inductive and abductive reasoning strategies, we specify the changes that were needed in our system to cope with an abductive operator and list the issues raised by the integration with the refinement operators described in the previous section and the decisions we took.

Theory revision generalizes *inductive concept learning* by using, apart from induction, a variety of other operators such as rule deletion, rule synthesis, etc.

In the following, we intend to present potential ways to perform the integration of abduction and induction in the context of theory revision. Indeed, abduction has been used in this context, where a system is given different observations about the world (which does not necessarily mean that they represent an explicit learning problem), and it is required to assimilate them into its knowledge base. A limited form of revision consists in providing an abductive explanation that was previously unknown and then adding it to the theory. Hence, abduction can be one of the operators of theory revision systems. In addition, abduction has been used in theory revision as a method for identifying the specific parts of the theory that need to be revised. We intend to exploit this operator for dealing with cases of incompleteness of the observations.

From a logic point of view, both induction and abduction are not truth-preserving. Thus, their conclusions can be controversial; accordingly they are both non-monotonic. In Artificial Intelligence the inductive and abductive problems can be regarded as dual to each other since they share the same basic formal specification (Dimopoulos & Kakas, 1996):
Given a theory T and a hypothesis H for a set of examples E concerning the same concept:

$$T \cup H \models E.$$

The hypothesis H stands for the set of rules that define the target concept.

In the inductive learning framework, the set of observations E consists of training examples and the aim is defining a proper hypothesis H, whose conditions refer to the predicates in T. In the abductive learning framework, the role of H and T is exchanged: the aim is now to draw abductive explanations from T such that, together with the hypothesis H, they account for the examples E.

In other words, abduction can be exploited in case of incompleteness, when some partially undefined predicates are to be explained. Indeed, abduction is known also as a form of *reasoning to the best explanation*. Thus, usually abduction has been used together with causal theories.

Starting from a purely inductive behaviour, INTHELEX has been developed further on by providing it with an additional operator, namely an *abductive proof procedure* which could help in managing situations in which not only the set of all observations is partially known, but each observation could be incomplete too. In particular, we adopted and adapted to our purposes the algorithm by Kakas & Mancarella (1990), in its modified version (Esposito et al., 1996b; Lamma et al., 2000) to deal with exceptions and undefined predicates.

Some representational changes had to be made. We adapted the usual Abductive Logic Programming (ALP) framework (Esposito et al., 1996b) to our theory revision problem. An *abductive logic program* (theory) is a triple (P, A, IC) where:

- P is a normal logic program (Lloyd, 1987);
- A is the set of abducible predicates;
- IC is a set of integrity constraints.

In our case, we considered only the 0-level predicate letters in the dependency graph (see Section 2) as members of the abducible set A. Moreover, in the IC set the abductive proof procedure requires the explicit representation of constraints like:

$ic([p_1(x),p_2(x), \ldots, p_n(x)])$. to be interpreted as:
$:\text{-}p_1(x), p_2(x), \ldots, p_n(x), !, fail.$

In the procedure, integrity constraints of the form:

:- *not p, p, !, fail*

are implicitly assumed for the default negation.

An *abductive derivation* from the goal G_1 to the goal G_n yielding a set of abductions Δ is a sequence of goals with the form $G_i = :\text{-} L_1, L_2, ..., L_m$.

On each step from G_1 to G_n, a literal L_j is chosen according to some selection rule, then:

1. if L_j is not abducible then G_{i+1} is the resolvent of G_i and some clause of P
2. if L_j is abducible and has been previously abduced ($L_j \in \Delta$) then L_j is dropped from G_i to give G_{i+1}
3. if L_j is abducible and has not been previously abduced ($L_j \notin \Delta$) then, if a *consistency derivation* is possible from $\{L_j\}$ to $\{\}$, L_j is dropped from G_i to give G_{i+1} and Δ is augmented with L_j and the set of assumptions made in the consistency derivation.

A consistency derivation is started when new assumptions have to be made to prove the current selected literal together with the previous assumptions (case 3): the abductive derivation algorithm may be invoked during the consistency derivation when neither the current selected literal nor its negation are in Δ.

The integration of this different form of reasoning with the inductive operators could not be straightforward, due to issues about abduction that are still unclear. Thus, a number of problems arose on which decisions had to be made since such integration can take place at several degrees of interrelation.

First of all, it was necessary to decide whether using it to add specific facts, related to the observed examples, to the theory, as in many ALP systems, or using it to complete the (possibly) partial information of the observations and then to generate/refine the theory according to the "extended" (by abduction) examples obtained. We chose this latter option, since it preserves a characteristic of INTHELEX: it does not allow specific facts in the theory, but attaches the abduced information directly to the observation it belongs to.

Abduction can be exploited as a means for identifying learning problems hidden in the new observation given to the system. This form of integration of abduction and theory revision was referred to as *Identification Problem* by Dimopoulos & Kakas (1996). This learning process is stratified in two levels. In the former, abduction identifies the actual training data to be supplied to a generalization problem. The observations are translated into information that has some pattern and that can be identified as a generalization problem. The latter involves a purely inductive process in which such observations are "synthesized" in the knowledge base.

Therefore, another choice was to be made. We could integrate abduction in the two refinement operators in order to obtain abductive theories (i.e., theories in which the classical resolution proof is replaced by the abductive proof). Alternatively, we could regard it as an identification problem, and complete the observations in such a way that they could be either covered (if positive) or ruled out (if negative) by the already generated theory. We tried this latter way, since it avoids, whenever possible, the use of the operators (and the consequent modification of the theory) and preserves the characteristic that theories generated by INTHELEX are also executable Prolog modules. Moreover, this solution limits the influence of abduction when it is not necessary and, as a consequence, the amount of information that was not directly subsumed by the theory but would be rather assumed by abductive inference. Thus, in our system, abduction is preliminarily used to generate suitable or relevant background data on which the inductive generalization is based.

The last issue concerned the set of literals that are abduced for supporting new observations (which will be added to their description). It could be a minimal set, or an enlarged set of all literals that necessarily satisfy all the integrity constraints of the theory (even those not involved in the observations). The former approach, followed in (Dimopoulos & Kakas, 1996), was chosen since it ensures that the inductive operators use abducibles only when really needed.

The pseudo-code of the tuning algorithm, illustrating how abduction was integrated in INTHELEX, is reported in figure 4. When a new observation is available, the abductive proof procedure mentioned above is started, parameterized on the current theory, the example and an empty set of abductive assumptions Δ. If the procedure succeeds, the resulting set of assumptions Δ, that were necessary to correctly classify the observation, is added to the example description before storing it (of course, being Δ minimal by definition, if no assumption is needed for the correct classification, the example description is not affected). Otherwise the usual refinement procedure (generalization or specialization) is performed.

The negative literals that might appear in the set Δ in order to satisfy some integrity constraints (Kakas & Mancarella, 1990) are not added to the description. Instead, they are handled by means of the Closed World Assumption (Lloyd, 1987).

```
Procedure Tuning (E: example; T: theory);
Δ := ∅;
if   found an abductive derivation of E from T with Δ list of abducted predicates
then
     Delete from Δ the negative literals;
     Add Δ to body(E)
else
     if E is a positive example
          then
               Generalize T with respect to E and maintaining consistency wrt all the past negative examples
          else
               Specialize T with respect to E and maintaining completeness wrt all the past positive examples
Store E in the historical memory of the processed examples
```

Figure 4. Tuning algorithm.

5. Experimental results

A first experiment has been run on a standard dataset already used by the FORTE system (Richards & Mooney, 1995), with the purpose of comparing the performance of INTHELEX to that of other well-known learning systems.

The dataset we chose concerns some family relationships, namely: *wife, husband, father, mother, brother, sister, son, daughter, aunt, uncle, niece,* and *nephew*. Starting from a set of facts regarding sex, marriage and parenthood of people involved in a given family tree, 744 examples were derived, by building their description in a uniform way for all the concepts to be learned. This method consisted in taking all the known marriage and parenthood facts involving the two persons of the example, then all the parenthood facts involving the persons newly introduced at the previous step and finally adding the sex attributes for all these persons. Of course, it could be the case that such a method results weaker for the description of more "far" family relationships, such as *aunt, uncle, niece* and *nephew*, where the information in the description is more "sparse". However, a uniform procedure was necessary not to affect the experiment and this one was estimated to be sufficient to give the necessary information for all the concepts to be learned.

In this method, the original negated examples were not taken into account, since they were implicit, due to the Closed World Assumption. The reduced dataset was made up of collections of examples of the following sizes: *wife* (25), *husband* (25), *father* (60), *mother* (60), *brother* (59), *sister* (47), *son* (66), *daughter* (54), *aunt* (82), *uncle* (92), *niece* (92), *nephew* (82).

Then, 30 runs of the experiment were performed, each one with the whole learning set given to INTHELEX, in order to generate a corresponding theory starting from an empty one. In particular, for each run the learning set was built by randomly taking the 40% of the examples for each concept and putting them together, while the remaining 60% were put together to form the test set. As a Background Knowledge, in this case represented by inference rules, INTHELEX was provided with the definition of "siblings" and "au" (this last one corresponding to the uncle/aunt relationship, regardless of the person's sex).

Despite the low percentage of examples used to generate the theory and the incremental fashion in which it was generated, we reached an average predictive accuracy of 94.27% (standard deviation 2.45), with a maximum of 98% and a minimum of 90%.

A different experimental setting was used to perform the real-world task of document classification.

Different experiments have been devised to analyze closely the effectiveness of the incremental methodologies on the problem of classifying scientific papers, which consists in classifying the documents into a set of distinct classes characterized by certain standard layout and logical structures (Esposito, Malerba & Semeraro, 1994).

We considered a database of 92 front pages of scientific papers, whose description has been automatically derived based on the geometrical properties and relationships of their layout components. The documents in the database belong to three different classes: 30 instances of class *ISMIS* (Proceedings of the International Symposium on Methodologies for Information Systems), 34 of class *PAMI* (IEEE Transactions on Pattern Analysis and Machine Intelligence) and 28 of class *ICML* (Proceedings of the International Conference on Machine Learning). Each paper is a positive example for the class it belongs to and, at the same time, is a negative example for all the other classes. For each class, the learning process has been performed both in a *batch* and in an *incremental* way.

We performed 33 runs of the experiments, by randomly splitting the database of 92 papers into a *learning* set and a *test* set, composed respectively of 70% and 30% of the whole dataset. In turn, the learning set has been subdivided into *training* and *tuning* sets, made up respectively of 30% and 70% of the whole learning set. In each run the learning set has been exploited in three distinct ways, according to the mode—batch or incremental—adopted for the learning process. In the first case, this set has been entirely given to INDUBI/H (Esposito, Malerba and Semeraro, 1994), an empirical learning system operating in a batch way, as its input. In the second case, it has been entirely given to INTHELEX in order to generate a corresponding theory for classification by starting from the empty one (simulating in this way the batch learning). Finally, for the incremental mode, only the training set has been used by INDUBI/H in order to produce the first version of the classification theory, and then the resulting rules were refined in an incremental way by INTHELEX, in case of incompleteness or inconsistency, by means of the tuning set. The tuning set is made up of both positive and negative examples in each run. Lastly, the test set has been exploited to evaluate the predictive accuracy of the learned theories on unclassified documents.

The average results of the experiments, as regards the predictive accuracy of the generated theories (expressed as percentages) on the test set are graphically shown in Table 1.

According to our expectations, the batch theories generated by INDUBI/H were more accurate than those generated incrementally by INTHELEX starting from the empty theory, even though, as regards the class PAMI, the difference was not considerable.

This is probably due to the fact that, being PAMI a journal, it has a strict layout standard imposed by the publisher, while both the other classes are printed from camera ready copies provided by the authors, and thus suffer from more variability in following the guidelines. For this reason, few examples are sufficient to characterize the class PAMI, while for the other classes, having all the examples available at once allows to focus on the most characterizing features.

The worse predictive accuracy of the theories produced by the incremental system, when compared to the accuracy of the theories produced in batch mode, starting from the empty theory, probably suffers from the order in which examples are presented to the system; we gave INTHELEX all the positive examples first, and then the negative ones, but probably a study about a more varied distribution interweaving positive and negative examples would lead to better results.

On the other hand, predictive accuracy when INTHELEX refines a theory initially generated by INDUBI/H, is very close to that obtained by running the batch system on the whole learning set.

Table 2 illustrates the results of the *t* test, exploited to evaluate the significance of the observed differences as to predictive accuracy for each class. This test has been performed as a two-sided paired test at a 0.05 level of significance, by considering the predictive accuracy achieved by the theories produced in each run along the modes already mentioned and comparing them couplewise.

From Tables 1 and 2, it is possible to note that in five cases there is a statistically significant difference (framed boxes in Table 2) and they are in favor of INDUBI/H, which showed an higher average predictive accuracy throughout all of the runs. It is worthwhile to note that, for the PAMI class, where INTHELEX showed a better performance on the average, both starting from an empty theory and even more when refining a starting theory previously generated by INDUBI/H, the results are not statistically significant.

As to the efficiency of the system, it is noteworthy the fact that the computational timings obtained on a SUN SparcStation 10 architecture are far in favor of the incremental system, both refining first versions of the theories and starting from the empty one.

Table 1. Results on document classification.

	Predictive accuracy %		
	INDUBI	*INTHELEX*	*INDUBI+INTHELEX*
ISMIS	89,63	82,17	91,33
PAMI	88,93	86,70	88,87
ICML	96.23	83	90,13

Table 2. *t*-test significance results on the document classification experiment outcomes

	INDUBI	*INDUBI*	*INTHELEX*
	INTHELEX	*INDUBI+ INTHELEX*	*INDUBI+ INTHELEX*
ISMIS	2,42E-06	0,28802	0,000159
PAMI	0,203127	0,970127	0,399112
ICML	4,43E-06	0,002864	0,02268

class_icml(A) :-	*contain_in_pos_top(A,B), width_medium_large(B), height_smallest(B), type_hor_line(B), contain_in_pos_top(A,C), height_smallest(C), type_text(C).*
"A is an ICML document since it contains in its top position an horizontal line B, whose width is medium-large and height is smallest, and a text block C in its top position, whose height is smallest"	
class_ismis(A) :-	*width_medium_small(B), contain_in_pos_center(A,B).*
class_ismis(A) :-	*contain_in_pos_top_right(A,B), align_only_left_col(B,C).*
class_ismis(A) :-	*contain_in_pos_right(A,B), width_very_very_small(B), height_very_large(B), type_ver_line(B), toright(C,B), width_medium(C), height_medium_large(C), type_text(C), contain_in_pos_top(A,D), width_medium_small(D), height_smallest(D), type_text(D), contain_in_pos_center(A,C).*
class_pami(x1) :-	*width_medium_small(x2), width_medium_small(x3), height_very_small(x2), height_medium_small(x3), type_text(x2), type_mixture(x3), contain_in_pos_top(x1,x2), contain_in_pos_center(x1,x3), !, fail.*
class_pami(A) :-	*width_medium_large(B), height_very_very_small(B), contain_in_pos_top(A,B).*
class_pami(A) :-	*height_very_small(B), contain_in_pos_top(A,B).*
class_pami(A) :-	*contain_in_pos_top(A,B), width_medium(B), height_smallest(B), type_text(B), contain_in_pos_top(A,C), height_very_very_small(C), type_text(C), contain_in_pos_center(A,D), width_medium_large(D), type_mixture(D), contain_in_pos_bottom(A,E), width_very_small(E), height_smallest(E), type_text(E).*

Figure 5. A theory generated for the three classes ICML, ISMIS and PAMI.

These results are very encouraging, since there are tasks whose nature, for instance because of the variability of the observations, is such that they are infeasible in batch mode and call for continuous refinement of the generated theories.

An example of generated theory for the three classes ICML, ISMIS and PAMI, is given in figure 5. It is possible to note the straight readability and understandability of the clauses, in addition to the presence of some exceptions (coded through the "*!, fail*" couple in the body).

Here, longer clauses are those generated by INTHELEX. Even if shorter ones are more practical, it should be noted that domain experts, when presented with automatically generated theories, place generally more confidence in longer clauses, probably because shorter ones seem to have poor characterization power.

6. Related work

A thorough survey of the work related to the learning systems available in the theory revision framework, both in the propositional and first-order learning (as well as in other areas like belief revision and qualitative modeling) has been already carried out by Richards & Mooney (1995). For this reason, in the following we will analyze our system, with respect to this discussion, and in particular compared to FORTE.

Many systems for theory revision deal with propositional logic. They can integrate different reasoning methods and learning strategies. These systems, such as EITHER (Mooney & Ourston, 1994) that is FORTE's conceptual predecessor, can deal with the use of a deductive-inductive method for modifying a given domain theory, but they cannot handle relations and so are limited as regards their expressive power.

Conversely, INTHELEX is a learning system, which is able to handle a first-order knowledge representation. In particular, as described in Section 2, it exploits the Datalog language, i.e. a function-free first-order language, with the Object Identity bias: the advantage is that ideal operators have been proven to exist in this reduced search space (Esposito et al., 1996a). In addition, such reduction leaves unchanged the expressive power with respect to an unbiased relation language (Semeraro et al., 1998). Thus, our system, as FORTE, is fully automatic since it avoids the need for user interaction that is exploited by many ILP systems as an external oracle for reducing the search space.

Some knowledge-based systems exploit first-order domain theories to bias the learning of operational conceptual descriptions without a modification of the domain theory. FORTE allows the definition of a portion of the domain theory as correct *a priori*. Similarly to FORTE, our system has a conservative behaviour, aiming at preserving as much as possible the initial theory. The "closeness" to the starting theory guarantees for the comprehensibility of the revisions when compared to purely inductive methods. Moreover in our framework both the rules and the examples share the same nature of Datalog clauses. In INTHELEX the whole theory is modifiable and, though a set of meta-rules is definable, it does not need for a theory coming from an external source. Rather than being thought as a system for "repairing" incorrect theories like FORTE, our system adopts a fully incremental learning algorithm to the extent that the starting theory might even be empty.

Other systems perform learning in one step, through an algorithm that starts from scratch and does not actually refine an initial theory. However, if a (good) initial theory is available, we can expect theory refinement to perform better as to their efficiency. Compared to FORTE, our system is inherently incremental, by which we mean that it does not assume the availability of the whole training set and processes the incoming new examples one at a time. As a consequence, the learned theory is ensured—at any moment—to be correct with respect to the entire set of examples taken into account.

It is also claimed that FORTE is able to perform the refinement of recursive theories. The mechanism relies on the positive examples of a concept recursively defined as its extensional definition. However, this approach suffers from the need of complete example sets for such concepts (all the possible instances below a certain size must be given). This is another reason why it cannot be considered an inherently incremental system. Moreover, FORTE requires the initial theory to contain such concepts already in recursive form: it cannot learn them otherwise. "This approach is not foolproof", but it is claimed to be "often effective", with the exception of three cases which cannot be automatically detected by the system.

FORTE can refine rules referring to subconcepts and related concepts for which examples are not explicitly provided by "abducing" them through the application of the rules for higher-lever concepts to the training examples. This requires these rules to be correctly defined at the moment when such process is carried out. However, it is not clear how such rules can be validated, since the lower-level concepts are possibly incorrect while being learnt; even declaring these rules as belonging to the "fundamental domain theory" (i.e. correct and, thus, not modifiable) would not explain why examples are given for concepts that are not to be learnt. For such reason, INTHELEX performs refinements by exploiting only the examples explicitly given for each concept and subconcept.

Systems that performed the theory refinement in one direction only (specializing/generalizing) have been improved (like GOLEM) by plugging in the dual operator to be exploited in case the former operator produces over-refinements. It is claimed that FORTE behaves better for it integrates both kinds of operators from the start and thus needs fewer examples. Our system adopts the same strategy, but only one of the possible refinements in the version space is kept as the current correct theory, instead of exploiting the training examples to converge to a correct theory. We can be sure that the current theory belongs to the set of correct theories, so far, in the version space since, for the full-memory policy adopted, we always validate any candidate refinement with respect to all the processed examples. The navigation along this space is guided by some heuristics, namely the generality of the candidate generalization, measured by the $size_{OI}$, and the depth, in the erroneous proof of the negative example, of the clause to be specialized.

As regards the specialization operator implemented in FORTE, named relational pathfinding, we point out that because of the bias of *linkedness* of the clauses refined by our operators, also in our framework the choice of antecedents added to a clause in order to refine it is restricted to the linked ones.

Differently from INTHELEX, many other systems, including FORTE, do not allow negative information items (negative literals) to be expressed in the theories because of computational complexity considerations. As a consequence, half of this potential search space is left not explored.

7. Conclusions and future work

This paper described the incremental learning system INTHELEX, a first order logic system able to acquire and revise its theory, defined as a set of concepts, in a completely automated way, on the grounds of positive and negative instances of the concepts. INTHELEX is a multi-conceptual system, being able to learn simultaneously different possibly related concepts, and integrates ideal refinement operators defined on the search space induced by the θ_{OI}-subsumption ordering together with an abductive refinement operator. Moreover, the system maintains the coherence of the revised theory with all the past and new instances, restoring the completeness and consistency properties when necessary.

There is currently no sophisticate method for localizing the causes of erroneous theories. One of the directions of our future research will be defining and implementing new heuristics for this purpose. Moreover, the possibility of introducing the recursion is being studied, trying to make use of the techniques developed in ATRE (Malerba, Esposito & Lisi, 1998), a first order logic learning system which learns recursive theories.

We reported the results of some experiments performed with INTHELEX, both with a standard dataset and with data concerning a real world application in the field of document classification. Other experiments are being carried out with the aim of comparing the performance of INTHELEX, in terms of predictive accuracy and computational time, to the ones of other multistrategy learning systems. New experiments are currently being carried out in the field of document understanding, concerning the possibility of acquiring information on the content of parts of documents basing on spatial layout. In this case, we have to learn

models of the logical components that refer to a part of the document rather than to the whole document.

Future work will concern also the use of a distance measure for a twofold purpose. First, we can exploit it when choosing the next candidate clause to be generalized, according to the strategy proposed by Mooney in the propositional case (1995). Second, to face the problem of selecting the most promising example when more than one observation contradicts the theory (Esposito, Malerba & Semeraro, 1992).

References

Becker, J. M. (1985). *Inductive learning of decision rules with exceptions: methodology and experimentation*. B.S. diss., UIUCDCS-F-85-945, Dept. of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Bergadano, F. & Gunetti, D. (1994). Learning clauses by tracing derivations. In S. Wrobel (Ed.), *Proceedings of the 4th International Workshop on Inductive Logic Programming, ILP-94* (pp.1–29).

Botta, M. (1994). Learning first order theories. In Z. W. Ras & M. Zemankova (Eds.), Methodologies for Intelligent Systems, *Lecture Notes in Artificial Intelligence* (Vol. 869: pp. 356–365). Springer-Verlag.

Cain, T. (1991). The Ductor: a theory revision system for propositional domains. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 485–489). San Mateo, CA: Morgan Kaufmann).

Ceri, S., Gottlob, G., & Tanca, L. (1990). *Logic programming and databases*. Springer-Verlag.

Clark, K. L. (1978). Negation as failure. In H. Gallaire & J. Minker (Eds.), *Logic and databases*, Plenum Press.

De Raedt, L. (1992). *Interactive theory revision—an inductive logic programming approach*. AP.

Dimopoulos, Y. & Kakas, A. (1996) Abduction and learning. In L. De Raedt (Ed.), *Advances in inductive programming* (pp. 144–171). IOS Press.

Esposito, F., Malerba, D., & Semeraro, G. (1992). Classification in noisy environments using a distance measure between structural symbolic descriprions. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 14*, 390–402.

Esposito, F., Malerba, D., & Semeraro, G. (1994). Multistrategy learning for document recognition. *Applied Artificial Intelligence: An International Journal, 8*, 33–84.

Esposito, F., Laterza, A., Malerba, D., & Semeraro, G. (1996a). Locally Finite, Proper and complete operators for refining datalog programs. In Z. W. Ras & M. Michalewicz (Eds.), *Lecture Notes in Artificial Intelligence* (Vol. 1079: Foundations of Intelligent Systems) (pp. 468–478) Springer.

Esposito, F., Lamma, E., Malerba, D., Mello, P., Milano, M., Riguzzi, F., & Semeraro, G. (1996b). Learning abductive logic programs. *ECAI 96 Workshop on Abductive and Inductive Reasoning*, Workshop Notes (pp. 23–30).

Ginsberg, A. (1990). Theory reduction, theory revision, and retranslation. *Proceedings of the 8th National Conference on Artificial Intelligence (AAAI-90)* (pp. 777–782).

Hayes-Roth, F. & McDermott, J. (1978). An interference matching technique for inducing abstractions. *Communications of the ACM, 21*, 401–410.

Helft, N. (1987). Inductive Generalization: A logical framework. In I. Bratko & N. Lavrac (Eds.), *Progress in machine learning—proceedings of EWSL 87: 2nd European Working Session on Learning* (pp. 149–157). Wilmslow: Sigma Press.

Josephson, J. R. & Josephson, S. G. (Eds.), (1996). *Abductive inference*. Cambridge University Press.

Kakas, A. C. & Mancarella, P. (1990). On the relation of truth maintenance and abduction. *Proceedings of the 1st Pacific Rim International Conference on Artificial Intelligence, PRICAI 90*, Nagoya, Japan.

Kanellakis, P. C. (1990). Elements of relational database theory. In J. Van Leeuwen (Ed.), *Handbook of Theoretical Computer Science* (*Volume B, Formal Models and Semantics*, pp. 1073–1156). Elsevier Science Publishers.

Komorowski, J. & Trcek, S. (1994). Towards refinement of definite logic programs. In Z. W. Ras & M. Zemankova (Eds.), *Lecture Notes in Artificial Intelligence* (Vol. 869: *Methodologies for Intelligent Systems*). (pp. 315–325) Springer-Verlag.

Lamma, E., Mello, P., Milano, M., Riguzzi, F., Esposito, F., Ferilli, S., & Semeraro, G. (2000). Cooperation of abduction and induction in logic programming. *Abductive and Inductive Reasoning: Essays on their Relation*

and Integration. Kluwer.

Lloyd, J. W. (1987). *Foundations of logic programming* 2nd edn. Springer-Verlag.

Malerba, D., Esposito, F., & Lisi, F. A. (1998). Learning recursive theories with ATRE. *Proceedings of 13th European Conference on Artificial Intelligence*, Brighton (pp. 435–439). John Wiley & Sons.

Matwin, S. & Plante, B. (1991). A deductive-inductive method for theory revision. *Proceedings of the 1st International Workshop on Multistrategy Learning*, Harper's Ferry, WVA (pp. 160–174). GMU Press.

Mooney, R. & Ourston, D. (1994) A multistrategy approach to theory refinement. In R. S. Michalski & G. Tecuci (Eds.), *Machine learning: a multistrategy approach* (Vol. 4), pp. 141–164 San Mateo, CA: Morgan Kaufman.

Mooney, R. (1995) A preliminary PAC analysis of theory revision. In T. Petsche, S. J. Hanson, & J. Shavlik (Eds.), *Computational learning theory and machine learning systems*. (Vol. 3, pp. 43–53), *MIT Press*.

Muggleton, S. (1987). Duce, an Oracle based approach to constructive induction. *Proceedings of the 10th International Joint Conference on Artificial Intelligence, IJCAI87* (pp. 287–292).

Plotkin, G. D. (1970). A note on inductive generalization. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence* (Vol. 5, pp. 153–163). Edinburgh University Press.

Plotkin, G. D. (1972). Building-in equational theories. In B. Meltzer & D. Michie (Eds.), *Machine Intelligence*, (Vol. 7, pp. 73–90). Edinburgh University Press.

Reinke, R. E. & Michalski, R. S. (1985). Incremental learning of concept descriptions: a method and experimental results. In D. Michie (Ed.), *Machine Intelligence*, (Vol. 11). Edinburgh University Press.

Richards, B. L. & Mooney, R. J. (1991). First-order theory revision. *Proceedings of the 8th International Workshop on Machine Learning* (pp. 447–451).

Richards, B. L. & Mooney, R. J. (1995). Refinement of first-order Horn-clause domain theories. *Machine Learning*, *19*(2), 95–131.

Saitta, L., Botta, M., & Neri, F. (1993). Multistrategy learning and theory revision. *Machine Learning*, *11*, 153–172.

Semeraro, G., Esposito, F., Fanizzi, N., & Malerba, D. (1995). Revision of logical theories. In M. Gori & G. Soda (Eds.), *LECTURE NOTES in ARTIFICIAL INTELLIGENCE*. (Vol 992: Topics in Artificial Intelligence), pp. 365–376. Springer-Verlag.

Semeraro, G., Esposito, F., Malerba, D., Fanizzi, N., & Ferilli, S. (1998). A logic framework for the Incremental Inductive Synthesis of Datalog Theories. In N. E. Fuchs (Ed.), *Proceedings of the 7th International Workshop on Logic Program Synthesis and Transformation* (pp. 300–321). *Lecture Notes in Computer Science* (Vol. 1463), Springer-Verlag.

Shapiro, E. Y. (1983). *Algorithmic program debugging*. MIT Press.

van der Laag, P. R. J. & Nienhuys-Cheng, S.-H. (1994). Existence and nonexistence of complete refinement operators. In F. Bergadano & L. De Raedt (Eds.), *Machine Learning: ECML-94-Proceedings of the European Conference on Machine Learning* (pp. 307–322). *LECTURE NOTES in ARTIFICIAL INTELLIGENCE* (Vol. 784), Springer-Verlag.

Vere, S. A. (1975). Induction of concepts in the predicate calculus. *Proceedings of the 4th International Joint Conference on Artificial Intelligence, IJCAI-75* (pp. 281–287).

Wogulis, J. (1991). Revising relational domain theories. *Proceedings of the 8th International Workshop on Machine Learning* (pp. 462–466).

Wrobel, S. (1994). *Concept formation and knowledge revision*. Kluwer Academic Publishers.

Wrobel, S. (1996). First order theory refinement. In L. De Raedt (Ed.), *Advances in Inductive Logic Programming* (pp. 14–33). Amsterdam: IOS Press.

Received Dec 22. 1998

Accepted June 18, 1999

Final manuscript June 18, 1999