Fundamenta Informaticae 57 (2003) 39–77
IOS Press

Learning Recursive Theories in the Normal ILP Setting

Donato Malerba
Dipartimento di Informatica
Università degli Studi di Bari
malerba@di.uniba.it

Abstract. Induction of recursive theories in the normal ILP setting is a difficult learning task whose complexity is equivalent to multiple predicate learning. In this paper we propose computational solutions to some relevant issues raised by the multiple predicate learning problem. A separate-and-parallel-conquer search strategy is adopted to interleave the learning of clauses supplying predicates with mutually recursive definitions. A novel generality order to be imposed on the search space of clauses is investigated, in order to cope with recursion in a more suitable way. The consistency recovery is performed by reformulating the current theory and by applying a layering technique, based on the collapsed dependency graph. The proposed approach has been implemented in the ILP system ATRE and tested on some laboratory-sized and real-world data sets. Experimental results demonstrate that ATRE is able to learn correct theories autonomously and to discover concept dependencies. Finally, related works and their main differences with our approach are discussed.

Keywords: machine learning, inductive logic programming (ILP), learning recursive theories, multiple predicate learning.

1. Introduction

Recursion is a fundamental concept in all abstract computation models with the same expressive power as Turing machines. It is also the only control flow mechanism available in pure logic programming, without which, few interesting functions might be actually implemented. Despite its computational relevance, recursion is rarely handled by inductive learning systems. There has been considerable debate on the actual usefulness of learning recursive programs in knowledge acquisition and discovery applications. It is a common opinion that very few real life concepts seem to have recursive definitions, rare examples

Address for correspondence: Dipartimento di Informatica, via Orabona 4, I-70126 Bari, Italy

being "ancestor" and natural language [6, 52]. Recursion is considered more useful in programming by examples [3, 4], where the main task is that of synthesizing programs that *compute* results, rather than generating programs that *classify* observations as instances of one concept or another. However, in the literature it is possible to find several other applications in which recursion has proved helpful, such as finite mesh design [7], dynamical systems [40], planning [31, 64] and automated telephony [66]. In all these cases, a pattern occurs repetitively in the same training observation, and the best way to capture such an occurrence is by means of recursive programs. The fact that one generally does not know in advance whether recursion is beneficial or not in a given application domain seems to justify the use of more general-purpose learning techniques that can induce both recursive and non-recursive programs [23].

Usually, explanations of recursion are based on the idea that the definition of a concept (formally expressed by either a function, a procedure or a predicate) is formulated in terms of the same concept. Although this is often true, such explanations skip those cases of *mutual recursion* in which several concepts are mutually defined. For instance, the following logic program:

$odd(succ(X)) \leftarrow even(X)$
$even(succ(X)) \leftarrow odd(X)$
$even(X) \leftarrow zero(X)$

provides a mutually recursive definition of odd and even numbers, although no clause, taken on its own, is recursive. This example shows that the problem of learning recursive programs is related to the problem of learning multiple concept definitions, and in the final analysis the two problems are equivalent.

This work can be framed in the area of inductive logic programming (ILP) [49, 12, 35, 4, 56], which uses computational logic as the representation formalism of both training observations and induced hypotheses. Therefore, we will assume that the concepts to be learned are represented by means of predicate symbols, and that the result of the learning process is a logical theory.

Most research in ILP has focused on learning in the so-called *normal* (or *strong* or *explanatory*) ILP setting [14, 15]. Given a background theory *BK*, a set of positive examples E^+ and a set of negative examples E^-, in the normal ILP setting, a hypothesis T is sought, such that: $BK \cup T \models E^+$ and $BK \cup T \not\models E^-$. In other words, the theory T has to be *complete* and *consistent* with respect to the set of training examples, given *BK*.

In this framework, inductive learning of recursive logical theories is equivalent to learning multiple predicate definitions from a set of examples. Recent renewed interest in learning multiple predicate definitions [32, 52], justified by the employment of ILP systems in more complex tasks, has induced us even more to investigate learning recursive theories.

Three important issues characterize multiple predicate learning problems.

i) De Raedt *et al.* [16] have shown that learning multiple predicates is more difficult than learning a single predicate. This is not due to the fact of having several predicate definitions to learn instead of only one; the main difficulty is that multiple target predicates involved in the same learning task might be someway related, so it is crucial to discover such dependencies, while learning predicate definitions. A wrong hypothesis on predicate dependencies may affect the learning results.

ii) A further difficulty lies in the generality order to be used in multiple predicate learning. The ordering typically used in ILP, namely θ-subsumption [58], is not sufficient to guarantee the completeness and consistency of learned definitions, with respect to logical entailment [27, 50, 55].

Therefore, it is necessary to consider a stronger generality order, which is consistent with the logical entailment for the class of logical theories we take into account.

iii) The main problems in multiple predicate learning can be explained in terms of an important property of the normal ILP setting: Whenever two individual clauses are consistent in the data, their conjunction need not be consistent in the same data [13]. This is called the *non-monotonicity property* of the normal ILP setting,[1] since it states that consistency is not preserved by adding new clauses to a theory T. Indeed, adding definite clauses to a definite program enlarges its least Herbrand model (LHM), which may then cover negative examples as well. For instance, the two clauses

C_1: $happy(X) \leftarrow loves(Y, X), woman(Y)$

C_2: $woman(Z) \leftarrow hag(Z)$

are individually consistent with respect to:

$BK = \{loves(daisy, donald), loves(amelia, scrooge), hag(amelia), hag(pemphredo)\}$

$E^+ = \{happy(donald), woman(daisy), woman(pemphredo)\}$

$E^- = \{happy(scrooge)\}$,

while the logical theory $T = \{C_1, C_2\}$ is not ($BK \cup T \models happy(scrooge)$). As a consequence of the non-monotonicity property, clauses supplying predicates with multiple definitions should not be learned individually but, in principle, they should be generated all together.

In order to overcome these problems, De Raedt and Lavrač [15] have proposed working on a *non-monotonic* setting of ILP, where clauses can be investigated independently of each other, since their interactions are no longer important. However, this setting produces properties of examples instead of rules generating examples. For instance, the following clause:

C_3: $loves(Y, X) \leftarrow happy(X)$

which expresses a necessary, but not sufficient, condition for being happy, can be generated in the non-monotonic setting. This kind of hypotheses cannot always be used for predicting the truth values of facts. When we are interested in predictions the normal ILP setting is more appropriate.

Several studies on the problem of learning restricted forms of recursive theories in the normal ILP setting have been presented in the literature. Cohen [11] proves positive and negative results on the pac-learnability of several classes of logic theories that are allowed to include a recursive clause. Cameron-Jones and Quinlan [9] investigate a heuristic method for preventing infinite recursion in single predicate definitions with the system FOIL. De Raedt *et al.* [16] propose an algorithm, named MPL, that performs a greedy hill-climbing search for learning multiple predicate definitions. Giordana *et al.* [26] define a bottom-up learning algorithm, called RTL, that first learns a hierarchical (i.e., non-recursive) theory T which is complete and consistent, and then tries to synthesize a simple recursive theory from T. Aha *et al.* [1] have developed a system called CRUSTACEAN (derived from LOPSTER by Lapointe and Matwin [34]), which is able to learn recursive definitions consisting of a unit clause and a two-literals recursive clause. PROGOL guides a top-down generalization process with a known bottom clause and can learn recursive clauses by inverting implication between function-free definite clauses [51]. Martin

[1] This property should not be confused with other properties of the alternative *non-monotonic ILP setting*, whose name is due to its relation to non-monotonic reasoning.

and Vrain [45] deal with the problem of inducing mutually recursive predicate definitions such that each clause of the learned theory extensionally covers some positive examples and rejects the negative ones. Boström [5] proposes an algorithm that, under some assumptions, correctly specializes a given recursive theory with respect to positive and negative examples. Idestam-Almquist [29] suggests a technique for efficiently learning recursive definitions including one base clause and one tail recursive clause from a random sample of examples. An iterative bootstrap induction method for learning recursive predicate definitions has been studied by Jorge and Brazdil [30]. Mofizur and Numao [48] adopt a top-down approach to learning recursive programs with only one recursive clause. An extension of MPL to normal programs is proposed by Fogel and Zaverucha [24]. Finally, an interactive multiple predicate learning system called Logan-H has been presented in the literature [32]. A thorough overview of achievements in the inductive synthesis of recursive logic programs can be found in [23].

In this paper, a new approach to the problem of learning multiple dependent concepts is proposed. It differs from other approaches for at least one of the following three aspects: the learning strategy, the generalization model, and the strategy to recover the consistency property of the learned theory when a new clause is added. These ideas have been implemented in a new version of the learning system ATRE [42], whose full description is reported in this paper.

The paper is organized as follows. Section 2 introduces the issues related to the induction of recursive logical theories. Section 3 illustrates the learning strategy adopted by ATRE. Section 4 is devoted to the generalization model whose implementation is also sketched. A solution to the problem of recovering non-monotonic theories is proposed in Section 5. In Section 6 the proposed approach is illustrated through the system ATRE, which is characterized by an object-centered representation of training examples, by the use of *seed* objects, and by the adoption of classical negation. Some experimental results and an application to the real-world problem of understanding multi-page printed documents are described in Section 7. Finally, Section 8 concludes, touches upon related work and discusses ideas for further work.

2. Problem specification

Henceforth, the term *logical theory* will denote a set of definite clauses. Every logical theory T can be associated with a directed graph $\gamma(T) = \langle N, E \rangle$, called the *dependency graph* of T, in which (i) each predicate of T is a node in N and (ii) there is an arc in E directed from a node a to a node b, iff there exists a clause C in T, such that a and b are the predicates of a literal occurring in the head and in the body of C, respectively.

A dependency graph allows representing the predicate dependencies of T, where a *predicate dependency* is defined as follows:

Definition 2.1. (predicate dependency)
A predicate p depends on a predicate q in a theory T iff (i) there exists a clause C for p in Tsuch that q occurs in the body of C; or (ii) there exists a clause C for p in T with some predicate r in the body of C that depends on q [12].

It is easy to notice that the direct (i) and indirect (ii) predicate dependencies of T are represented as arcs and paths respectively in $\gamma(T)$. This correspondence may be highlighted by reformulating the problem from an algebraic point of view. Let $\pi(T)$ be the set of predicates occurring in the logical theory T. The direct (i) predicate dependencies in T may be mathematically depicted as instances of a binary

relation on $\pi(T)$, namely $R_{dpd} \subseteq \pi(T) \times \pi(T)$. The binary relation R_{pd} for predicate dependencies is the transitive closure of R_{dpd}. Given that each binary relation can be associated with a directed graph, the graph corresponding to R_{dpd} is just the dependency graph $\gamma(T) = \langle \pi(T), R_{dpd} \rangle$.

Definition 2.2. (recursive theory)
A logical theory T is *recursive* if the dependency graph $\gamma(T)$ contains at least one cycle.

In *simple* recursive theories all cycles in the dependency graph go from a predicate p into p itself, that is, simple recursive theories may contain recursive clauses, but cannot express mutual recursion. An example of a dependency graph for a recursive theory is given in Figure 1.

Figure 1. A recursive theory and its corresponding dependency graph for the predicates *odd* and *even*.

Definition 2.3. (predicate definition)
Let T be a logical theory and p a predicate symbol. Then the *definition* of p in T is the set of clauses in T that have p in their head.

Henceforth, $\delta(T)$ will denote the set of predicates defined in T. It is worthwhile to notice that $\delta(T) \subseteq \pi(T)$. For instance, with reference to the theory in Figure 1, $\delta(T) = \{even, odd\}$, while $\pi(T) = \{even, odd, zero, succ\}$.

In a quite general formulation, the learning task in the normal ILP setting can be defined as follows:

Given

- A set of predicates $p_1, p_2, \ldots, p_r$ to be learned
- A set of positive (negative) examples $E_i^+ (E_i^-)$ for each predicate p_i, $1 \leq i \leq r$
- A background theory *BK*
- A language of hypotheses $\mathcal{L}_H$ that defines the space of hypotheses S_H

Find
a (possibly recursive) logical theory $T \in S_H$ defining the predicates $p_1, p_2, \ldots, p_r$ (that is, $\delta(T) = \{p_1, p_2, \ldots, p_r\}$) such that for each i, $1 \leq i \leq r$, $BK \cup T \models E_i^+$ (*completeness* property) and $BK \cup T \not\models E_i^-$ (*consistency* property).

Most studies on the problem of induction of recursive theories have concentrated on learning a simple recursive predicate definition [9, 29, 30, 48], that is a single predicate definition including some recursive clause. In this case, the main issue is how to guarantee that learned definitions are intensionally complete and consistent. Learning simple recursive theories is more complicated, since it is necessary to discover

the right order in which predicates should be learned [26], that is the dependency graph of the theory. Once such an order has been determined, possibly using statistical techniques, the problem can be boiled down to learning single predicate definitions [44]. The learning problem becomes harder for (mutually) recursive theories,[2] because the learning of one (possibly recursive) predicate definition should be *interleaved* with the learning of the other ones. One way to build such interleaving is by parallel learning clauses for different predicates. In fact this is the strategy adopted by ATRE.

3. The learning strategy

The high-level learning algorithm in ATRE belongs to the family of *sequential covering* (or *separate-and-conquer*) algorithms [47], since it is based on the strategy of learning one clause at a time (procedure LEARN-ONE-RULE), by removing the covered examples and iterating the process on the remaining examples. Indeed, a recursive theory T is built step by step, starting from an empty theory T_0, and adding a new clause at each step. In this way we obtain a sequence of theories:

$$T_0 = \emptyset, T_1, \ldots, T_i, T_{i+1}, \ldots, T_n = T$$

such that $T_i \subset T_{i+1}$ and all theories in the sequence are consistent with respect to the training set. If we denote with $LHM(T_i)$ the least Herbrand model of a theory T_i, the stepwise construction of theories requires that $LHM(T_i) \subseteq LHM(T_{i+1})$, for each $i \in \{0, 1, \ldots, n-1\}$. Indeed, the addition of a clause to a theory can only augment the least Herbrand model of the theory. Henceforth, we will assume that both positive and negative examples of predicates to be learned are represented as *ground atoms* with a **+** or **-** label. Since examples are ground atoms and Herbrand models are sets of ground atoms by definition, it is possible to check whether an example belongs to $LHM(T_i)$, for each $i \in \{0, 1, \ldots, n\}$.

Let $pos(LHM(T_i))$ and $neg(LHM(T_i))$ be the number of positive and negative examples in $LHM(T_i)$, respectively. If we guarantee that $pos(LHM(T_i)) < pos(LHM(T_{i+1}))$, for each $i \in \{0, 1, \ldots, n-1\}$ and that $neg(LHM(T_i)) = 0$, for each $i \in \{0, 1, \ldots, n\}$, then, after a finite number of steps, a theory T, which is complete and consistent, is built. Whether the theory T is "correct", that is, whether it classifies correctly all other examples *not* in the training set, cannot be established, since no information on the generalization accuracy can be drawn from the same training data. In fact, the selection of the "best" theory is always made on the basis of an inductive bias embedded in some heuristic function or explicitly expressed by the user of the learning system (preference criterion).

In order to guarantee the first condition above, namely $pos(LHM(T_i)) < pos(LHM(T_{i+1}))$, we follow the same procedure adopted in INDUCE [46] and Progol [51]. First, a positive example e^+ of a predicate p to be learned is selected, such that $e^+ \notin LHM(T_i)$. The example e^+ is called *seed*. Then the space of definite clauses more general than e^+ is explored, looking for a clause C, if any, such that $neg(LHM(T_i \cup \{C\})) = 0$. In this way, we guarantee that the second condition above holds as well. When found, C is added to T_i giving T_{i+1}. If some positive examples are not included in $LHM(T_{i+1})$, then a new seed is selected and the process is repeated.

The most relevant novelties of the learning strategy sketched above are embedded in the design of the procedure LEARN-ONE-RULE being proposed. Indeed, this implements a parallel general-to-specific

[2] In general, the mutual recursion cannot be removed by reformulating a theory T without introducing additional new predicates. This also explains the additional degree of complexity in learning recursive theories, with respect to inducing simple recursive theories.

example-driven search strategy in the space of definite clauses, whose ordering (called *generalized implication*) is explained in Section 4. The search space is actually a forest of as many search-trees (called *specialization hierarchies*) as the number of chosen seeds, where at least one seed per incomplete predicate definition is kept. Each search-tree is rooted with a unit clause and a directed arc from a node C to a node C' exists if C' is obtained from C by a single refinement step. Operatively, the (downward) refinement operator considered in this work adds a new literal to a clause.[3]

The forest can be processed in parallel by as many concurrent tasks as the number of search-trees. Each task traverses the specialization hierarchies top-down (or general-to-specific), but synchronizes traversal with the other tasks at each level. Initially, some clauses at depth one in the forest are examined concurrently. Each task is actually free to adopt its own search strategy, and to decide which clauses are worth testing. If none of the tested clauses is consistent, clauses at depth two are considered. Search proceeds towards deeper and deeper levels of the specialization hierarchies, until at least one consistent clause is found.

Task synchronization is performed after all "relevant" clauses at the same depth have been examined. A supervisor task decides whether the search should carry on or not, on the basis of the results returned by the concurrent tasks. When the search has stopped, the supervisor selects the "best" consistent clause, according to the user's preference criterion.

The advantage of this strategy is that the simplest consistent clauses are found first, independently of the predicates to be learned.[4] Moreover, the synchronization allows tasks to save much computational effort when the distribution of consistent clauses in the levels of the different search-trees is uneven.

The parallel exploration of the specialization hierarchies for the predicates *odd* and *even* is shown in Figure 2. Suppose that the seeds *even(0)* and *odd(1)* are selected. A partial view of the two corresponding specialization hierarchies is shown in the figure, where consistent clauses are reported in italics. Levels refer to the specialization step. By exploring the two hierarchies level by level, ATRE finds several candidate clauses to add to the initial empty theory T_0. However, the simplest clause, that is:

even(X) ← *zero(X)*

is found first, and is added to T_0, so the base case of recursion is defined. To generate the second clause, two new seeds are considered, say *even(2)* and *odd(1)*. In this case, the first two consistent clauses generated by the system are:

odd(X) ← *succ(Y,X), zero(Y)*

odd(X) ← *succ(Y,X), even(Y)*.

In particular, the generation of the second clause is possible since a partial definition of even numbers has already been generated in the previous step. One of the two clauses will be selected and added to the theory $T_1 = \{even(X) \leftarrow zero(X)\}$.

It is noteworthy that both clauses entail the only positive example *odd(1)*, therefore, a selection of the most promising clause can be based exclusively on some form of search bias or user preference criterion. For instance, the second clause is preferred because it is more general than the first, given the partially learned theory (see the definitions of generality order in the next section). Another critical choice is that concerning the seeds. If the system had started from *even(2)* and *odd(1)*, the first clause added

[3] A discussion on properties of this operator is beyond the scope of this paper. A thorough description of upward and downward refinement operators can be found in [56].

[4] Note that in some recursive definitions a recursive clause can be syntactically simpler than the base clause. This might appear to cause problems in this strategy. However, the proposed strategy does not allow the discovery of the recursive clause until the base clause has been found, whatever its complexity is.

to the theory would have been *odd(X) ← succ(Y,X), zero(Y)*, thus resulting in a less compact, though still correct, theory for odd and even numbers. Therefore, it is important to explore the specialization hierarchies of several seeds for each predicate. When training examples and background knowledge are represented either as sets of ground atoms (flattened representation) or as ground clauses, the number of candidate seeds can be very high, so the choice should be stochastic. The object-centered representation adopted by ATRE has the advantage of reducing the number of candidate seeds by partitioning the whole set of training examples E into *objects*. Each object contains some of the seeds for the generation of base clauses in a recursive theory. Since in many learning problems the size of an object is not very high, a parallel exploration of all candidate seeds is feasible and the computational issue reported above is solved. More details on ATRE's object-centered representation are given in Section 6.

For a comparison of the proposed search strategy with respect to the other systems that attack the problem of multiple predicate learning see Section 8.1.

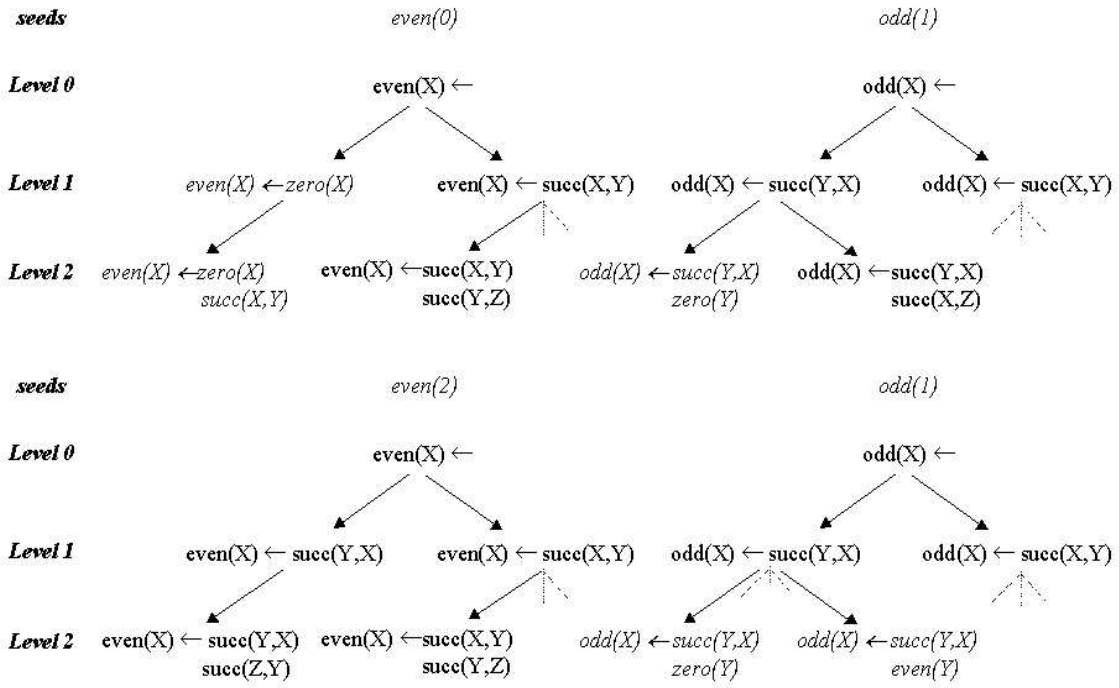

Figure 2. Parallel search for the predicates *odd* and *even*

4. The generalized implication model

A more precise definition of the search space of the LEARN-ONE-RULE stage is necessary. We follow the usual practice established in ILP of defining a generality order (or *generalization model*), which provides a basis for organizing this search space. Indeed, clarifying what is the generality order is the first step towards a reasoned design of a learning system.

Several generality orders have been reported in the literature, the most known being the θ-subsumption [58]. In the θ-subsumption, the objects of comparison are two clauses, say C and D, and no additional

source of knowledge (e.g., a theory T) is considered. This is also true for other generality order like *implication* or *T-implication* [28]. However, comparing two definite clauses, whatever the generality order is, may lead to incorrect results in the case of recursive theories.

Example 4.1. For instance, with reference to previous example on *odd* and *even* predicates, the clause

C: $odd(X) \leftarrow succ(Y, X), even(Y)$

logically entails, and hence can be correctly considered more general than

D: $odd(3) \leftarrow succ(0, 1), succ(1, 2), succ(2, 3), even(0)$

only if we take into account the theory:

T: $even(A) \leftarrow succ(B, A), odd(B)$

$even(C) \leftarrow zero(C)$

Therefore, we are only interested to those generality order that compare two clauses *relatively* to a given theory T, such as Buntine's *generalized subsumption* [8].

Definition 4.1. (generalized subsumption)
Let C and D be two definite clauses with disjoint variables:

C: $C_0 \leftarrow C_1, C_2, \ldots, C_n$

D: $D_0 \leftarrow D_1, D_2, \ldots, D_m$

C is *more general than D* under generalized subsumption with respect to a theory T, denoted $C \preceq_T D$, if a substitution σ exists, such that $C_0\theta = D_0$ and for each substitution σ that grounds the variables in D using new constants which do not occur in C, D, and T, it happens that:

$T \cup \{D_1\sigma \leftarrow, D_2\sigma \leftarrow, \ldots, D_m\sigma \leftarrow\} \models \exists(C_1, C_2, \ldots, C_n)\theta\sigma$.

Operatively, the above test can be performed by proving that

$T \cup \{D_1\sigma \leftarrow, D_2\sigma \leftarrow, \ldots, D_m\sigma \leftarrow\} \vdash_{SLD} (C_1, C_2, \ldots, C_n)\theta\sigma$

Informally, generalized subsumption requires that the heads of C and D refer to the same predicate, and that the body of D can be used, together with the background theory T, to entail the body of C.

Example 4.2. Let us consider the following clauses:

C: $even(X) \leftarrow succ(Y, X), succ(Z, Y), even(Z)$

D: $even(U) \leftarrow succ(V, U), succ(W, V), zero(W)$

and the following theory:

T: $even(A) \leftarrow zero(A)$

Then, the substitution $\theta = \{X \leftarrow U\}$ matches the head of C against that of D. Moreover, for each substitution σ that grounds the variables in D (i.e., U, V and W) using new constants which do not occur in C, D, and T, it happens that:

$succ(Y, U\sigma), succ(Z, Y), even(Z)$

can be derived from the definite program

1. $even(A) \leftarrow zero(A)$
2. $succ(V\sigma, U\sigma)$
3. $succ(W\sigma, V\sigma)$
4. $zero(W\sigma)$.

The derivation is reported in Figure 3. Therefore, $C \preceq_T D$.

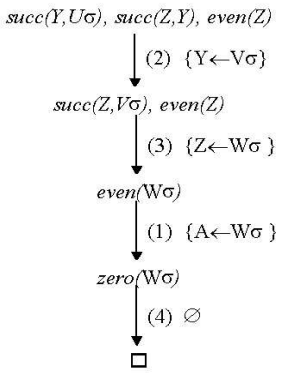

Figure 3. SLD resolution proving that $C \preceq_T D$. At each step both the number of the resolving clause and the most general unifiers are reported.

Unfortunately, generalized subsumption is too weak for recursive theories.[5]

Example 4.3. Let us consider the following clauses:

C: $odd(X) \leftarrow succ(Y, X), even(Y)$

D: $odd(U) \leftarrow succ(V, U), succ(W, V), succ(Z, W), zero(Z)$

and the following theory:

T: $even(A) \leftarrow succ(B, A), odd(B)$

$even(C) \leftarrow zero(C)$

The substitution $\theta = \{X \leftarrow U\}$ matches the head of C against that of D. However, if we consider the following grounding substitution $\sigma = \{U \leftarrow 3, V \leftarrow 2, W \leftarrow 1, Z \leftarrow 0\}$ the following goal clause:

$succ(Y, 3), even(Y)$

cannot be derived from the definite program

1. $even(A) \leftarrow succ(B, A), odd(A)$
2. $even(C) \leftarrow zero(C)$
3. $succ(2, 3)$
4. $succ(1, 2)$
5. $succ(0, 1)$
6. $zero(0)$.

Therefore, we cannot conclude that $C \preceq_T D$, although $T \cup \{C\} \models D$.

[5]Informally, an order is too strong for a class $\mathcal{L}$ of theories when it can be used to organize theories of a strictly wider class $\mathcal{L}' \supset \mathcal{L}$, according to logical entailment. If the organization of theories in $\mathcal{L}$ is not consistent with logical entailment, then the order is too weak.

An alternative generality order, known as *relative generalization*, was proposed by Plotkin [58, 59]. It is at the base of the better known Plotkin's definition of relative least general generalization [59].

Definition 4.2. (relative generalization)
Let H be a set of clauses, T a set of unit clauses (or a conjunction of atoms) and D a definite clause. H is *more general than* D given T, if and only if one can derive a definite clause C by resolution from T∪H which θ-subsumes D, and all resolutions in the derivation of C involve a clause of T.

Buntine [8] reports an extension of Plotkin's *relative generalization* to the case of a theory T composed of definite clauses (not necessarily unit clauses).

Definition 4.3. (relative generalization)
Let C and D be two definite clauses. C is *more general than* D under relative generalization, with respect to a theory T, if a substitution θ exists such that $T \models \forall(C\theta \Rightarrow D)$.

The following theorem holds for the this extended notion of relative generalization:

Theorem 4.1. Let C and D be two definite clauses and T a logical theory. C is more general than D under relative generalization, with respect to T, if and only if C occurs at most once in some refutation demonstrating $T \models \forall(C \Rightarrow D)$.

However, this extended notion of relative generalization is still inadequate. From one side, it is still weak. Indeed, if we consider the clauses and the theory reported in example 4.1, it is clear that a refutation demonstrating $T \models \forall(C \Rightarrow D)$ involves twice the clause C to prove both $odd(1)$ and $odd(3)$. On the other side, this notion is too strong for our goals. Taken literally, it would lead us to consider unintuitive solutions of the conquer stage as illustrated in the following example.

Example 4.4. Let us consider the following training set:
+: $\{p(a), p(b)\}$
−: $\{p(c)\}$
the following background knowledge:
BK: $\{q(a), r(a,b), s(b), r(c,b)\}$
and the following incomplete theory built at the first step:
T_1: $p(X) \leftarrow q(X)$.
Let $p(b)$ be the selected seed. A desirable search space of clauses more general than $p(b)$, given $BK \cup T_1$, would be the set of clauses whose head is $p(X)$, since our aim is to induce a predicate definition for p. This space contains, for instance, the clause $p(Z) \leftarrow s(Z)$. However, the space of clauses that are relatively more general than $p(b)$ given $BK \cup T_1$ includes other solutions, such as $q(Y) \leftarrow s(Y)$. This last clause is correct and even relatively more general than $p(Z) \leftarrow s(Z)$.[6] Nevertheless it is not intuitive because it does not seem to be related to the target predicate p. Moreover, this solution is not coherent with our formulation of the learning problem, since q is defined in the background knowledge and does not appear among the predicates whose definition had to be induced. In this work, we do not consider these solutions, *which would make the multiple predicate learning problems even harder to solve*.

[6] Indeed, given $T = BK \cup T_1$ and $\theta = \{Y \leftarrow Z\}$, it can be proven that $T \models \forall(q(Z) \leftarrow s(Z)) \Rightarrow (p(Z) \leftarrow s(Z))$, which is equivalent to proving that $T \cup \forall\{q(Z) \leftarrow s(Z)\} \models \forall(p(Z) \leftarrow s(Z))$.

This restriction is reflected by a new ordering, named *generalized implication*.

Definition 4.4. (generalized implication)
Let C and D be two definite clauses. C is *more general than* D under generalized implication, with respect to a theory T, denoted as $C \preceq_{T,\Rightarrow} D$, if a substitution θ exists such that $head(C)\theta = head(D)$ and $T \models \forall(C \Rightarrow D)$.

This generality order can be proved to be strictly stronger than generalized subsumption since the condition $T \models \forall(C \Rightarrow D)$ in Definition 4.4 entails the condition $T \cup \{D_1\theta \leftarrow, D_2\theta \leftarrow, \ldots, D_m\theta \leftarrow\ \} \models \exists(C_1, C_2, \ldots, C_n)\theta\sigma$ in Definition 4.1.[7] In view of Theorem 4.1 generalized implication reduces to generalized subsumption when C compares only at the root of the refutation demonstrating $T \models \forall(C \Rightarrow D)$. Moreover, the following properties hold.

Proposition 4.1. Let C, D and E be definite clauses and T be a theory. The generalized implication order satisfies the properties of:

i) *reflexivity*: $C \preceq_{T,\Rightarrow} C$

ii) *transitivity*: $C \preceq_{T,\Rightarrow} D$ and $D \preceq_{T,\Rightarrow} E$, then $C \preceq_{T,\Rightarrow} E$

Proof:
i) Trivial if the empty substitution is chosen.
ii) By definition, a substitution θ_1 exists such that $head(C)\theta_1 = head(D)$ and $T \models \forall(C \Rightarrow D)$, and a substitution θ_2 exists such that $head(D)\theta_2 = head(E)$ and $T \models \forall(D \Rightarrow E)$. Let $\theta = \theta_1\theta_2$ be the substitution obtained by the composition of θ_1 and θ_2. Then $head(C)\theta = head(C)\theta_1\theta_2 = head(D)\theta_2 = head(E)$. Moreover, from $T \models \forall(C \Rightarrow D)$ and $T \models \forall(D \Rightarrow E)$, $T \models \forall(C \Rightarrow E)$ follows. □

It can also be proved that the semi-decidability of the generalized implication, namely the termination of the generalized implication test, is guaranteed when $C \preceq_{T,\Rightarrow} D$. However, such a negative result is overcome when Datalog clauses [10] are considered. In fact, the restriction to function-free clauses is common in ILP systems, which remove function symbols from clauses and put them in the background knowledge by techniques such as flattening [61]. This transformation does not cause problems in handling numeric data, as we will show in Section 6.

4.1. Implementing generalized implication test

A naive implementation of the generalized implication test is obtained by testing that $head(C)\theta = head(D)$ for some θ and then by computing the least Herbrand models of $\{C\} \cup T$ and $\{D\} \cup T$: $C \preceq_{T,\Rightarrow} D$ if and only if $LHM(\{D\} \cup T) \subseteq LHM(\{C\} \cup T)$. Since the least Herbrand model

[7] Let θ be a substitution that unifies the heads of C and D. Since the implication is monotone, with respect to the application of a substitution, $T \models \forall(C \Rightarrow D)$ entails $T \models \forall(C\theta \Rightarrow D\theta)$. Moreover, it is sufficient to consider the logical equivalence between $C\theta \Rightarrow D\theta$ and $\neg D\theta \Rightarrow \neg C\theta$, and the theorem stating that proving $T \models \forall(\neg D\theta \Rightarrow \neg C\theta)$ is equivalent to proving $T \cup \{\neg D\theta\} \models \forall\neg C\theta$.

of a theory T coincides with the least fixed point of the immediate consequence operator τ_T,[8] we can compute the least Herbrand models operatively.

Definition 4.5. (immediate consequence operator)
Let $ground(T)$ be the set of all ground instances of clauses in T. Given a Herbrand interpretation $\mathcal{I}$, that is a set of ground atoms built on the same alphabet of T, the immediate consequence operator computes the following interpretation:

$\tau_T(\mathcal{I}) := \{H |\ H \leftarrow B_1, \ldots, B_m \in ground(T), \{B_1, \ldots, B_m\} \subseteq \mathcal{I}\}$

It can be shown that, for sets of definite clause, the infinite sequence of interpretations computed by the iterative application of the immediate consequence operator:

$\tau_T \uparrow 0 := \emptyset$
$\tau_T \uparrow 1 := \tau_T(\emptyset)$
$\tau_T \uparrow 2 := \tau_T(\tau_T(\emptyset))$
$\ldots$
$\tau_T \uparrow i := \underbrace{\tau_T(\tau_T \ldots \tau_T}_{i}(\emptyset))$
$\ldots$

converges to $LHM(T)$, which is the least fixpoint of τ_T [39]. Notationally, $\tau_T \uparrow \infty := LHM(T)$. Moreover, the convergence to the fixpoint in a finite number of iterations is guaranteed by the finiteness of the Herbrand base for Datalog theories.

One cause of inefficiency in the naive evaluation is that ground facts in $LHM(T)$ may be computed many times during the iterative application of τ_T. Semi-naïve evaluation partially overcomes this redundancy by partitioning T into n *layers*, such that $T = T^0 \cup \ldots \cup T^i \cup \ldots \cup T^{n-1}$ and $LHM(T) = LHM(LHM(\bigcup_{j=0,\ldots,n-2} T^j) \cup T^{n-1})$. Actually, more efficient methods than the semi-naïve one proposed in this section are reported in the literature. They have been mostly developed in the context of bottom-up evaluation of queries in deductive databases [10, 60]. However, the main objective of our proposal is that of *introducing some important properties of* layered *theories, which will be useful in the next section on recovering the consistency of a theory*.

It is worthwhile noticing that the computation of $LHM(LHM(\bigcup_{j=0,\ldots,i-1} T^j) \cup T^i)$ is equivalent to the iterative application of the immediate consequence operator to T^i, starting from the interpretation $LHM(\bigcup_{j=0,\ldots,i-1} T^j)$, that is $\tau_{T^i}(LHM(\bigcup_{j=0,\ldots,i-1} T^j))$. In this way, clauses in $T^0 \cup \ldots \cup T^{i-1}$ are no longer considered when computing the logical consequences of T^i.

Example 4.5. Let T' be the following theory:

C_1: $p(X) \leftarrow q(X)$
C_2: $q(Y) \leftarrow r(X, Y)$

and

BK: $\{q(a), r(a, b), s(b), r(c, b)\}$.

Let us suppose that the theory $T = BK \cup T'$ may be partitioned as follows: $T = T^0 \cup T^1 \cup T^2$, where $T^0 = \{r(a, b), s(b), r(c, b)\}$, $T^1 = \{q(a), C_2\}$ and $T^2 = \{C_1\}$.

Indeed, $LHM(T) = \{q(a), r(a, b), s(b), r(c, b), q(b), p(a), p(b)\}$, and

[8] The standard notation used in logic programming and deductive databases for the immediate consequence operator is T_P, where P is the logic program or the Datalog program. Henceforth, the operator will be denoted by τ_T, in order to avoid confusion with the symbol T used for the logical theories.

$LHM(T^0) = T^0$
$LHM(T^0 \cup T^1) = LHM(LHM(T^0) \cup T^1) = LHM(T^0 \cup T^1) =$
$= \{r(a,b), s(b), r(c,b), q(a), q(b)\} = \tau_{T^1}(\{r(a,b), s(b), r(c,b)\}) = \tau_{T^1}(LHM(T^0))$
$LHM(T^0 \cup T^1 \cup T^2) = LHM(LHM(T^0 \cup T^1) \cup T^2) =$
$= LHM(\{r(a,b), s(b), r(c,b), q(a), q(b)\} \cup \{C_1\}) = \{r(a,b), s(b), r(c,b), q(a), q(b), p(a), p(b)\} =$
$= \tau_{T^2}(\{r(a,b), s(b), r(c,b), q(a), q(b)\}) = \tau_{T^2}(LHM(T^0 \cup T^1))$

Notice that according to the classical iterative application of the immediate consequence operator the ground atoms $p(a)$ and $q(b)$ would be computed both in $\tau_T \uparrow 2$ and $\tau_T \uparrow 3$, since:

$\tau_T \uparrow 0 := \emptyset$
$\tau_T \uparrow 1 := \tau_T(\tau_T \uparrow 0) = BK$
$\tau_T \uparrow 2 := \tau_T(\tau_T \uparrow 1) = BK \cup \{p(a), q(b)\}$
$\tau_T \uparrow 3 := \tau_T(\tau_T \uparrow 2) = BK \cup \{p(a), q(b)\} \cup \{p(a), q(b), p(b)\}$.

Issues related to the problem of finding layers of a recursive theory T such that $LHM(T) = LHM(\bigcup_{j=0,\ldots,n-1} T^j) = LHM(LHM(\bigcup_{j=0,\ldots,n-2} T^j) \cup T^{n-1})$ are to be dealt with. Difficulties arise because the dependency graph $\gamma(T)$ is a directed cyclic graph. In order to remove cycles from $\gamma(T)$ we resort to the notion of the *strongly connected component* of a directed graph [37].

Definition 4.6. (strongly connected component)
Two vertices of a directed graph G, v_1 and v_2, are said to be *strongly connected* if there is a directed path from v_1 to v_2 *and* a directed path from v_2 to v_1.

It can be proven that a directed graph $G = \langle V, E \rangle$ can be decomposed as an "acyclic graph" of strongly connected components in $O(|V|+|E|)$ time. The result of this decomposition is called collapsed dependency graph.

Definition 4.7. (collapsed dependency graph)
Let $\gamma(T)$ be the dependency graph of a logical theory T. The *collapsed dependency graph* of T, denoted as $\hat{\gamma}(T)$, is a directed acyclic graph (dag), obtained by collapsing each (maximal) strongly connected component of $\gamma(T)$ into a single node.

Nodes in $\hat{\gamma}(T)$ are equivalence classes with respect to the strong connectivity relation. Thus, given also the properties of *dag*'s, it is easy to compute the level of a predicate $p \in \pi(T)$ as the maximum distance of [p] from a terminal node in $\hat{\gamma}(T)$, where terminal nodes are nodes with no out-coming edges.

Definition 4.8. (predicate level)
Let $\hat{\gamma}(T)$ be the collapsed dependency graph of a logical theory T. The *level* of a predicate $p \in \pi(T)$ is given by:

$$level(p) = \begin{cases} 0 & \text{if [}p\text{] is a terminal node in } \hat{\gamma}(T) \\ 1 + max\{level(q) | q \in \pi(T) \text{ and [}q\text{] is a child of [}p\text{] in } \hat{\gamma}(T)\} & \text{otherwise} \end{cases}$$

Any logical theory can be layered on the basis of the level of its predicates.

Definition 4.9. (layered theory)
Let $\hat{\gamma}(T)$ be the collapsed dependency graph of a logical theory T. Then T can be partitioned into n disjoint sets of clauses $T = T^0 \cup \ldots \cup T^i \cup \ldots \cup T^{n-1}$, called *layers*, such that
$\forall i \in \{0, \ldots, n-1\} : \delta(T^i) = \{p \in \pi(T) | \quad level(p) = i\}$.

It is worthwhile observing that such a technique for the layering of a logical theory induces a total order on the layers, $T^0 \preceq \ldots \preceq T^i \preceq \ldots \preceq T^{n-1}$.

Example 4.6. Let T be a theory obtained by the union of a background knowledge
BK: $\{f(a), s(a,b), s(b,c), s(c,d), s(d,e)\}$
and a theory T' consisting of
C_1: $p(X) \leftarrow f(X)$
C_2: $q(Y) \leftarrow p(Z), s(Z,Y)$
C_3: $p(U) \leftarrow q(V), s(V,U)$
Given $T = BK \cup T'$, then $\pi(T) = \{f, s, p, q\}$, while $\gamma(T)$ is the following:

$$\begin{array}{ccc} p & \leftrightarrow & q \\ \downarrow & \searrow & \downarrow \\ f & & s \end{array}$$

Thus the equivalence classes with respect to strong connectivity are $[f]$, $[s]$, and $[p, q]$, and $\hat{\gamma}(T)$ is the following graph:

$$\begin{array}{ccc} & [p, q] & \\ \swarrow & & \searrow \\ [f] & & [s] \end{array}$$

with $level(f) = level(s) = 0$ and $level(p) = level(q) = 1$. Therefore, two layers $T^0 = BK$ and $T^1 = T'$ are extracted.

The following proposition can be proved.

Proposition 4.2. Let T be a logical theory which has been partitioned into n layers, $T = T^0 \cup \ldots \cup T^{n-1}$, and H be a ground atom with predicate symbol $p \in \delta(T^k)$, $k = 0, \ldots, n-1$. Then $H \in LHM(T)$, if and only if $H \in LHM(LHM(\bigcup_{r=0,\ldots,k-1} T^r) \cup T^k)$.

Proof:
($\Rightarrow$) Let $H \in LHM(T)$. For the fix-point theorem, $\exists i > 0$, such that $H \in \tau_T \uparrow i$, that is, by definition of immediate consequence operator, a ground clause of T exists, $H \leftarrow A_1, \ldots, A_m \in ground(T)$, where $A_j \in \tau_T \uparrow (i-1)$, $j = 1, 2, \ldots, m$. Since $p \in \delta(T^k)$ then $H \leftarrow A_1, \ldots, A_m \in ground(T^k)$.
The proof is by induction on the layer k.

$k = 0$ We want to prove that each $A_j \in LHM(T^0)$, from which $H \in LHM(T^0)$ follows. The proof continues by induction on the iteration step i.

- $i = 1$ Since $H \in \tau_T \uparrow 1$, then $H \in T$, that is, H is a ground fact of the theory T. More specifically, $H \in T^0$, therefore H is in the $LHM(T^0)$.
- $i > 1$ Each $A_j \in \tau_T \uparrow (i-1)$, therefore $A_j \in LHM(T)$. From the construction of the layers it follows that each A_j is a ground atom with predicate symbol $\delta(A_j) \subseteq \pi(T^0)$. By the induction hypothesis each $A_j \in LHM(T^0)$, $j = 1, 2, \ldots, m$.

$k > 0$ We want to prove that each $A_j \in LHM(LHM(\bigcup_{r=0,\ldots,k-1} T^r) \cup T^k)$, from which $H \in LHM(LHM(\bigcup_{r=0,\ldots,k-1} T^r) \cup T^k)$ follows. The proof continues by induction on the step i.

$i = 1$ Since $H \in \tau_T \uparrow 1$, then $H \in T$, that is, H is a ground fact of the theory T. More specifically, $H \in T^k$, therefore H is in the $LHM(T^k)$, and more generally H is in $LHM(LHM(\bigcup_{i=0,\ldots,k-1} T^i) \cup T^k)$, since the addition of a set of clauses (i.e., the ground clauses $LHM(\bigcup_{i=0,\ldots,k-1} T^i)$) to a theory T^k can only augment the least Herbrand model of the theory.

$i > 1$ From the construction of the layers it follows that each A_j is a ground atom with predicate symbol $\delta(A_j) \subseteq \delta(T^r)$, for some $r \leq k$. Moreover, $A_j \in \tau_T \uparrow (i-1)$, therefore $A_j \in LHM(T)$. By both inductive hypotheses (on k and i), we may say that $A_j \in LHM(LHM(\bigcup_{r=0,\ldots,k-1} T^r) \cup T^k)$, for each $j = 1, 2, \ldots, m$, and then conclude that $H \in LHM(LHM(\bigcup_{r=0,\ldots,k-1} T^r) \cup T^k)$.

($\Leftarrow$) Let $H \in LHM(LHM(\bigcup_{r=0,\ldots,k-1} T^r) \cup T^k)$. In particular, if $H \in LHM(\bigcup_{r=0,\ldots,k-1} T^r)$, then $H \in LHM(T)$, since $\bigcup_{r=0,\ldots,k-1} T^r \subseteq T$. Otherwise, H is obtained by applying iteratively the immediate consequence operator τ_{T^k}, starting with the interpretation $LHM(\bigcup_{r=0,\ldots,k-1} T^r) \subseteq LHM(T)$, that is, $H \in \tau_{T^k}(LHM(\bigcup_{r=0,\ldots,k-1} T^r)) \uparrow \infty$. Since τ_{T^k} is monotone and $T^k \subseteq T$, $\tau_{T^k}(LHM(\bigcup_{r=0,\ldots,k-1} T^r)) \uparrow \infty \subseteq \tau_{T^k}(LHM(T)) \uparrow \infty = LHM(T)$. Therefore $H \in LHM(T)$.

□

This proposition states that a necessary and sufficient condition for a ground atom with predicate symbol $p \in \delta(T^k)$ being in $LHM(T)$ is that H is computed by iteratively applying the immediate consequence operator τ_{T^k}, starting with the interpretation $LHM(\bigcup_{r=1,\ldots,k-1} T^r)$. Proposition 4.2 can be used to prove the following theorem on the least Herbrand model of a layered theory.

Theorem 4.2. Let T be a theory which has been partitioned into n layers according to the criterion given in Definition 4.9. Then

$\forall n \geq 1$: $LHM(T) = LHM(LHM(\bigcup_{r=0,\ldots,n-2} T^r) \cup T^{n-1})$.

Proof:

($\Rightarrow$) Let $P^0, P^1, \ldots, P^{n-1}$ be a partition of $LHM(T)$, such that each P^i, $i = 0, \ldots, n-1$, contains only ground atoms with a predicate symbol in $\delta(T^i)$. From Proposition 4.2 it follows that $P^i \subseteq LHM(LHM(\bigcup_{j=0,..,i-1} T^r) \cup T^i)$ and

$LHM(T)$=$P^0 \cup P^1 \cup \ldots \cup P^{n-1}$

$\subseteq LHM(T^0) \cup LHM(LHM(T^0) \cup T^1) \cup \ldots \cup LHM(LHM(\bigcup_{j=0,\ldots,n-2} T^r) \cup T^{n-1})$

$= LHM(LHM(\bigcup_{j=0,\ldots,n-2} T^j) \cup T^{n-1})$,

since $LHM(\bigcup_{j=0,\ldots,i-1} T^j) \subseteq LHM(LHM(\bigcup_{j=0,\ldots,i-1} T^j) \cup T^i)$.

($\Leftarrow$) Let H be in $LHM(LHM(\bigcup_{j=0,\ldots,n-2} T^j) \cup T^{n-1})$. If $H \in LHM(\bigcup_{j=0,\ldots,n-2} T^j)$ then $H \in LHM(T)$, since $\bigcup_{j=0,\ldots,n-2} T^j \subseteq T$. Otherwise, H is obtained by applying iteratively the immediate consequence operator $\tau_{T^{n-1}}$, starting with the interpretation $LHM(\bigcup_{j=0,\ldots,n-2} T^j) \subseteq \boldsymbol{LHM}(T)$, that is, $H \in \tau_{T^{n-1}}(LHM(\bigcup_{j=0,\ldots,n-2} T^j)) \uparrow \infty$. Since $\tau_{T^{n-1}}$ is monotone and $T^{n-1} \subseteq T$,

$\tau_{T^{n-1}}(LHM(\bigcup_{j=0,\ldots,n-2} T^j)) \uparrow \infty \subseteq \tau_{T^{n-1}}(LHM(T)) \uparrow \infty \subseteq \tau_T(LHM(T)) = LHM(T)$. Therefore $H \in LHM(T)$. □

To sum up the procedure, the layering of a theory provides a semi-naive way of computing the generalized implication test presented above. The importance of layering will be more evident when the problem of recovering consistency is dealt with (see next section).

5. The consistency recovery strategy

Another learning issue to be considered in multiple predicate learning is the non-monotonicity of the normal ILP setting: Whenever two individual clauses are consistent on the data, their conjunction does not need to be consistent on the same data [13]. Algorithmic implications of this property may be effectively illustrated by means of an example.

Example 5.1. Let the following sets be positive examples, negative examples and background knowledge respectively:

+: $\{p(5)$,$q(1)$, $q(4)\}$

-: $\{p(3), q(3)\}$

BK: $\{f(3)$,$g(0), g(1), s(0,1), s(1,2), s(2,3), s(3,4), s(4,5)\}$

Let us suppose that the following consistent, but not complete, recursive theory T_2 has been learned after two conquer stages:

C_1: $q(X) \leftarrow s(Y,X), f(Y)$

C_2: $p(Z) \leftarrow s(W,Z), q(W)$

Note that $C_1 \preceq_{\{C2\} \cup BK,\Rightarrow} \{q(4)\}$, and $C_2 \preceq_{\{C1\} \cup BK,\Rightarrow} \{p(5)\}$, that is T_2 explains two positive examples $q(4)$ and $p(5)$ given BK. Since T_2 is incomplete, the learner will generate a new clause, say

C: $q(U) \leftarrow s(V,U), g(V)$

which is consistent (it entails $q(1)$ and $q(2)$, given $T_2 \cup BK$), but when added to the recursive theory, it makes clause C_2 inconsistent ($C_2 \preceq_{\{C1,C\} \cup BK,\Rightarrow} \{p(3)\}$).

There are several ways to remove such inconsistency by revising the learned theory. Nienhuys-Cheng and de Wolf [54] describe a complete method of specializing a logic theory with respect to sets of positive and negative examples. The method is based upon unfolding, clause deletion and subsumption. These operations are not applied to the last clause added to the theory, but may involve any clause of the inconsistent theory. As a result, clauses learned in the first inductive steps could be totally changed or even removed. This theory revision approach, however, is not coherent with the stepwise construction of the theory T presented in Section 3, since it re-opens the whole question of the validity of clauses added in the previous steps. An alternative approach consists of simple syntactic changes in the theory, which eventually creates new *layers* in a logical theory, just as the stratification of a normal program creates new strata [2].

Our recovery strategy proposal follows the latter approach, since it is based on the layering technique illustrated in Section 4.1 (see Definition 4.9).

Example 5.2. In the previous example, it is possible to define only three layers for $T_2 \cup BK$ (see Figure 4):

1 $T^0 = BK$ with $\delta(T^0)$=$\{f, g, s\}$,
2 $T^1 = \{C_1, C\}$ with $\delta(T^1)$=$\{q\}$, and
3 $T^2 = \{C_2\}$ with $\delta(T^2)$=$\{p\}$.

By reformulating C_1 and C_2 as follows:
C_1': $q'(X) \leftarrow s(Y, X), f(Y)$
C_2': $p(Z) \leftarrow s(W, Z), q'(W)$
and by adding the following two clauses:
C_3: $q(A) \leftarrow q'(A)$
C: $q(U) \leftarrow s(V, U), g(V)$
the new theory T_2' will present three different layers:

1. $T'^0 = BK$ with $\delta(T'^0)$=$\{f, g, s\}$,
2. $T'^1 = \{C_1'\}$ with $\delta(T'^1)$=$\{q'\}$, and
3. $T'^2 = \{C_2', C_3, C\}$ with $\delta(T'^2)$=$\{q, p\}$.

It is easy to see that the theory T_2' is consistent.

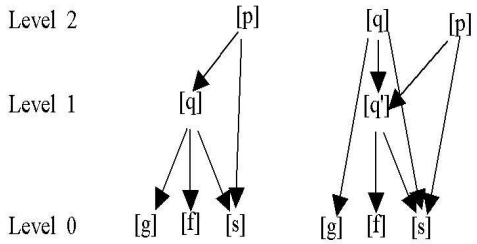

Figure 4. Collapsed dependency graph for T_2 and T_2'.

Due to theory restructuring, the number of layers may increase, as proved by the following proposition.

Proposition 5.1. Let T be a consistent theory partitioned into n layers, $T = T^0 \cup \ldots \cup T^i \cup \ldots \cup T^{n-1}$ and C be a definite clause whose addition to the theory T makes a clause in T^i inconsistent. Let $p \in \{p_1, p_2, \ldots, p_r\}$ be the predicate in the head of C, that is $\delta(\{C\}) = \{p\}$. Let T'' be a theory obtained from T by substituting all occurrences in T of the predicate p with a new predicate symbol p'. Then the theory $T' = T'' \cup \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\} \cup \{C\}$ has a number of layers greater than or equal to T.

Proof:
Obviously, if the addition of C to T makes T^i inconsistent then p is represented in the collapsed dependency graph $\hat{\gamma}(T)$. Let l be the $level(p)$ in $\hat{\gamma}(T)$. Since p' replaces p in T, it has the same level of p before theory restructuring, namely $level(p') = l$ in $\hat{\gamma}(T')$. Moreover, $level(p) > level(p')$ in $\hat{\gamma}(T')$. If l equals the maximum level of a node in $\hat{\gamma}(T)$, then the new theory T' has a predicate at a greater level, that is, the number of layers in T' increases. Conversely, the level of all predicates q depending on p in T can either increase, because of the breaking of equivalence classes, or remain stable. □

{*Pre-conditions*: T is consistent with respect to E; C is consistent with respect to E given T; $\delta(\{C\}) \subseteq \{p_1, p_2, \ldots, p_r\}$}
procedure *verify_global_consistency(C, T, E)*
partition the theory T into $T^0 \cup \ldots \cup T^i \cup \ldots \cup T^{n-1}$
partition E into $E^0 \cup \ldots \cup E^i \cup \ldots \cup E^{n-1}$ such that E^i contains instances of $\delta(T^i)$
for each $i \in \{0, 1, \ldots, n-1\}$
if a negative example $e^- \in E^i$ exists such that $T \cup \{C\} \models e^-$ **then**
{the addition of C to T makes a clause in T^i inconsistent}
let p be the predicate in $head(C)$
let p' be a new predicate symbol
T'' := renaming p with p' in T
return $T'' \cup \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\} \cup \{C\}$
end for each
return $T \cup \{C\}$
{*Post-condition*: T'= verify_global_consistency(C, T, E), T' is consistent with respect to E, $LHM(T) \subseteq LHM(T')$}

Figure 5. Procedure for theory layering. The input are the induced theory T, the new induced clause C, the background knowledge BK and the whole set of examples E. The procedure returns a new theory, which is consistent and explains at least all examples explained by T.

An example showing an increase of the number of layers and a breaking of equivalence classes is reported in [19].

It is noteworthy that, under the assumption of T partially defining the predicates $\{p_1, p_2, \ldots, p_r\}$ to learn, the addition of a clause C can make T^i inconsistent only if at least another clause for $p \in \delta(\{C\})$ has already been added to T. In other words, inconsistency may occur when the following two conditions hold:

a. $\gamma(T)$ contains a dependence $p \leftarrow q$

b. the definition of p needs at least two clauses to explain all positive examples of p itself.

The procedure for theory layering is reported in Figure 5. Since the computational complexity of the first partitioning step is linear in the number of vertices and arcs of the collapsed dependency graph $\hat{\gamma}(T)$, the real computational burden of the procedure *verify_global_consistency* is in the test $T \cup \{C\} \models e^-$ for each negative example e^- in E. Obviously, supposed that the induced theory is a Datalog program, the above logical entailment can be computed by the terminating semi-naïve procedure for the computation of generalized implication (see Section 4.1).

The procedure returns a new theory, which is consistent and explains at least all examples explained by T, as proved by the following two propositions.

Proposition 5.2. Let T be a consistent theory partitioned into n layers, $T = T^0 \cup \ldots \cup T^i \cup \ldots \cup T^{n-1}$ and C be a definite clause which is consistent given T but makes a clause in T^i inconsistent when added

to T. Let $p \in \{p_1, p_2, \ldots, p_r\}$ be the predicate in the head of C, that is $\delta(\{C\}) = \{p\}$. Let T'' be a theory obtained from T by substituting all occurrences of p in T with a new predicate symbol, p'. Then the theory $T' = T'' \cup \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\} \cup \{C\}$ is consistent.

Proof:
By definition $LHM(T') = LHM(T'' \cup \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\} \cup \{C\})$.

Since the conclusions of C cannot affect the conclusions of $T'' \cup \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\}$ because of the layering effect we have:

$LHM(T'' \cup \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\} \cup \{C\}) =$

$= LHM(LHM(T'' \cup \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\}) \cup \{C\}) =$

$LHM(LHM(T) \cup \{p'(t_1, \ldots, t_n) |\, p(t_1, \ldots, t_n) \in LHM(T)\} \cup \{C\})$

since $T'' \cup \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\}$ simply renames p with p' and adds the clause $p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)$. Moreover, the following equivalence holds:

$LHM(LHM(T) \cup \{p'(t_1, \ldots, t_n) |\, p(t_1, \ldots, t_n) \in LHM(T)\} \cup \{C\}) =$

$= LHM(LHM(T) \cup \{C\}) \cup \{p'(t_1, \ldots, t_n) | p(t_1, \ldots, t_n) \in LHM(T)\}$

since new ground atoms $p'(t_1, \ldots, t_n)$ are generated on the basis of $LHM(T)$ alone.

By chaining all these equivalences we have:

$LHM(T') = LHM(LHM(T) \cup \{C\}) \cup \{p'(t_1, \ldots, t_n) |\, p(t_1, \ldots, t_n) \in LHM(T)\}$.

Suppose that T' is inconsistent, that is, a ground atom $H \in LHM(T')$ exists, such that H is a negative example in the training set. Obviously, $H \notin \{p'(t_1, \ldots, t_n) |\, p(t_1, \ldots, t_n) \in LHM(T)\}$, since negative examples are instances of the predicates $p_1, p_2, \ldots, p_r$. Moreover, $H \notin LHM(LHM(T) \cup \{C\})$ since clause C is consistent given T, that is, $LHM(LHM(T) \cup \{C\})$ does not contain negative examples. The contradiction follows. □

Corollary. Let T be a consistent theory partitioned into n layers, $T = T^0 \cup \ldots \cup T^i \cup \ldots \cup T^{n-1}$ and C be a definite clause whose addition to the theory T makes a clause in T^i inconsistent. Let $p \in \{p_1, p_2, \ldots, p_r\}$ be the predicate in the head of C, that is $\delta(\{C\}) = \{p\}$. Let T'' be a theory obtained from T by substituting all occurrences of p in T with a new predicate symbol, p', and $T' = T'' \cup \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\} \cup \{C\}$. Then

$LHM(T) \subseteq LHM(T') \backslash \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\}$.

Proof:
It follows immediately from proof of Proposition 5.2, since

$LHM(T') = LHM(LHM(T) \cup \{C\}) \cup \{p'(t_1, \ldots, t_n) |\, p(t_1, \ldots, t_n) \in LHM(T)\}$

and $LHM(T') \backslash \{p(t_1, \ldots, t_n) \leftarrow p'(t_1, \ldots, t_n)\} = LHM(LHM(T) \cup \{C\}) \supseteq LHM(T)$.

Set inclusion is strict if C explains an example not previously explained by T. □

In short, the new theory T' obtained by layering is consistent when C is consistent given T and it keeps the original coverage of T.

It is noteworthy that, in the proposed approach to consistency recovery, new predicates are invented, which aim to accommodate previously acquired knowledge (theory) with the currently generated hypothesis (clause).

6. The ATRE system

ATRE is a multiple-concept learning system, which solves the following problem:

Given

- a set of concepts $K_1, K_2, \ldots, K_r$ to be learned,
- a set of objects O described in a language $\mathcal{L}_O$,
- a background knowledge *BK* described in a language $\mathcal{L}_{BK}$,
- a language of hypotheses $\mathcal{L}_H$ that defines the space of hypotheses S_H
- a user's preference criterion *PC*,

Find

a (possibly recursive) logical theory $T \in S_H$, defining the concepts $C_1, C_2, \ldots, C_r$, such that T is complete and consistent with respect to the set of observations and satisfies the preference criterion *PC*.

Two differences, with respect to the learning problem formulated in Section 2, are: the introduction of the concept of the preference criterion and the replacement of the terms "predicates" and "examples" with the words "concepts" and "objects", respectively. The introduction of the preference criterion, which was not required to describe the logical foundations of our learning procedure, is now necessary to deal with the problem of selecting the "best" theory among those satisfying the completeness and consistency properties. The second difference is due to ATRE's representation formalism, which is explained in the following subsection.

6.1. Representation issues

In ATRE the basic component of the representation languages $\mathcal{L}_O$, $\mathcal{L}_{BK}$, $\mathcal{L}_H$ is the *literal*, which takes two distinct forms:

$f(t_1, \ldots, t_n) = Value$ (*simple literal*) and $f(t_1, \ldots, t_n) \in [a..b]$ (*set literal*),

where f and g are function symbols called *descriptors*, t_i's and s_i's are terms and $[a..b]$ is a closed interval. Descriptors can be either *nominal* or *linear*, according to the ordering relation defined on their domain values. In particular, no ordering relation is defined on the domain of nominal descriptors, thus they can only appear in simple literals. On the contrary, a total ordering relation is defined for linear domains, thus linear descriptors can also appear in set literals. Some examples of literals are: *color(X)=blue*, *distance(X,Y)=463.09*, *extension(*X_1*)*$\in$[982.207 .. 983.103], and *close_to(*X, Y*)=true*.

The last example shows the lack of predicate symbols in the representation languages adopted by ATRE. Therefore, the first-order literals $p(X, Y)$ and $\neg p(X, Y)$ will be represented as f_p*(X,Y)=true* and f_p*(X,Y)=false*, respectively, where f_p is the function symbol associated to the predicate p. This means that ATRE can deal with *classical negation*, $\neg$, but not with *negation by failure*, *not* [39]. Henceforth, for the sake of simplicity, we will adopt the usual notation $p(X, Y)$ and $\neg p(X, Y)$, instead of f_p*(X,Y)=true* and f_p*(X,Y)=false*, respectively.

Each concept K_i to be learned is represented by a simple literal. Concepts sharing the same descriptor (and arity), but having different values, define a *multi-class* problem [65], which imposes the membership

of a training example in exactly one class, so that positive examples of one class are negative examples for the other mutually exclusive classes. For instance, with reference to the application domain of document image understanding reported in the next section, the concepts

K_1: *logic_type(X)=title*

K_2: *logic_type(X)=abstract*, and

K_3: *logic_type(X)=author*

define a three-class problem. Training examples of *logic_type* can be either instances of *title*, or *abstract*, or *author* or any *other* logic type, but they cannot be instances of two (or more) logic types simultaneously. Training examples of *logic_type(X)=title* will be considered negative for the concepts *logic_type(X)=abstract* and *logic_type(X)=author*, and viceversa. Moreover, training examples of *logic_type(X)=other*, which is not a concept we are interested in learning, are considered to be negative examples of *title*, *abstract* and *author*.

It is noteworthy that also in multi-class problems it is possible to have concept dependencies expressed by recursive theories. For instance, with reference to the same document image understanding domain, the following clause can be learned:

logic_type(X)=author ← *on_top(Y,X)* , *logic_type(Y)=title*

which expresses the dependence between the position of a block title and the position of a block author in a page layout.

Concepts to be learned can also have different descriptors, as in the typical multiple predicate learning problem concerning the 'family' domain:

K_1: *mother(X,Y)=true*

K_2: *father(X,Y)=true*, and

K_3: *ancestor(X,Y)=true*

This flexibility in the formulation of the learning problem is a distinguishing characteristic of the system.

ATRE's *language of observations* $\mathcal{L}_O$ is *object-centered*, in the sense that observations are represented as ground multiple-head clauses [38], called *objects*, which have a conjunction of simple literals in the head. The following is an instance of an object taken from the blocks-world:

O_1: $type(blk1) = lintel \wedge type(blk2) = column \leftarrow pos(blk1) = hor, pos(blk2) = ver, on\_top(blk1, blk2)$

Note that this multiple-head clause is semantically equivalent to the definite program:

$type(blk1) = lintel \leftarrow pos(blk1) = hor, pos(blk2) = ver, on\_top(blk1, blk2)$

$type(blk2) = column \leftarrow pos(blk1) = hor, pos(blk2) = ver, on\_top(blk1, blk2)$

but is not equivalent to the *disjunctive* clause:

$type(blk1) = lintel, type(blk2) = column \leftarrow pos(blk1) = hor, pos(blk2) = ver, on\_top(blk1, blk2)$

whose comma in the head is interpreted as a disjunction and not a conjunction.

The notion of multiple-head clauses in ATRE adapts the notion of *interpretation*, which is common to many relational data mining systems [18]. It presents two main advantages with respect to definite clauses: better comprehensibility and efficiency. The former is basically due to the fact that multiple-head clauses provide the system with a compact description of multiple properties to be predicted in

complex objects. The second advantage is the possibility of having a unique representation of known properties shared by a subset of observations. In fact, ATRE distinguishes *objects* from *examples*. As said before, objects are ground multiple-head clauses (i.e., interpretations) which can be uniquely identified by an object identifier OID. Examples are described as pairs $\langle L$, *OID*$\rangle$, where L is a literal in the head of the object indicated by the object identifier *OID*. Examples can be considered *positive* or *negative*, according to the concept to be learned. For instance $\langle$*type(blk1)=lintel,* $O_1\rangle$ is a positive example of the concept *type(X)=lintel*, a negative example of the concept *type(X)=column*, and it is neither a positive nor a negative example of the concept *stable(X)=true*. The *body* of an example $\langle L$, *OID*$\rangle$, *body*($\langle L$,*OID*$\rangle$), is the body of the multiple-head clause identified by OID.

The object identifiers define a partitioning of the set of training examples. This makes the choice of the seeds in the separate-and-parallel-conquer search strategy more efficient. Indeed, the basic assumption made in ATRE is that *each object contains examples explained by some base clauses of the underlying recursive theory*. Therefore, by choosing as seeds all examples of different concepts represented in one training object, it is possible to induce some of the correct base clauses. Mutually recursive concept definitions will be generated only after some base clauses have been added to the theory. Problems caused by incomplete object descriptions violating the above assumption are not investigated in this work, since they require the application of abductive operators, which are not available in the current version of the system.

The *language of hypotheses* $\mathcal{L}_H$ is that of *linked*, *range-restricted* definite clauses [12] with simple and set literals in the body and one simple literal in the head. It is noteworthy that ATRE also deals with numeric descriptors. More precisely, given an n-ary function symbol, $f(X_1, ..., X_n)$, taking values in a numerical domain, ATRE can produce hypotheses with set literals $f(X_1, ..., X_n) \in [a..b]$, where $[a..b]$ is a numerical interval computed according to the same information theoretic criterion used in INDUBI/CSL [43]. Much related work can also be found in other contexts, such as qualitative and relational regression in inductive logic programming, and learning numerical constraints in inductive constraint logic programming. An updated review can be found in the work by [36].

The *background knowledge* defines the relevant domain knowledge. It is expressed in a language $\mathcal{L}_{BK}$ with the same constraints as the language of hypotheses. The following is an example of spatial background knowledge:

close_to(X,Y) $\leftarrow$ *distance(X,Y)* $\in [0..20]$,

which states that two objects whose distance is between 0 and 20 are also close.

The representation languages used by ATRE do not seem to fit very well into the ILP framework, but it is easy to transform ATRE's definite clauses into Datalog clauses, extended with built-in predicates. The transformation of literals like *on_top(X,Y)=true* or *on_top(X,Y)=false* is straightforward.[9] In general, a simple literal $f(t_1, \ldots, t_n) = Value$ can be transformed into an $(n + 1)$-ary predicate $f(t_1, \ldots, t_n, Value)$, while a set literal $f(t_1, \ldots, t_n) \in Range$, where *Range* is an interval $[a..b]$, can be transformed into $f(t_1, \ldots, t_n, Z), Z \geq a, Z \leq b$. The relational operators $\geq$ and $\leq$ are built-in predicates.

Thanks to this transformation it is possible to apply to ATRE all concepts and properties developed in standard first-order logic languages. In particular, a clause can still be considered a set of literals, and the definitions of resolution and θ-subsumption for clauses with simple literals remain unchanged. The only extension is due to the presence of set literals, whose transformation introduces the built-in

[9] Negation can be removed by replacing the classical negation of each predicate p with a new predicate p^- [25].

predicates $\geq$ and $\leq$. In this case, a clause C can be partitioned into two subsets of literals, those with ordinary predicates, C_o, and those with built-in predicates, C_b. A clause C θ-subsumes a clause D if a substitution θ exists, such that $C_o\theta \subseteq D_o$ and the set of solutions of the constraints $C_b\theta$ is a non-empty set including the set of solutions of the constraints D_b. For instance, the following clause

C: $p(X) = a \leftarrow q(X) \in [0.5..1.5]$ $\quad C_o = \{p(X, a), q(X, Y)\}$ $\quad C_b = \{Y \geq 0.5, Y \leq 1.5\}$

θ-subsumes the following clause D for $\theta = \{X \leftarrow Z, Y \leftarrow W\}$,

$D : p(Z) = a \leftarrow q(Z) \in [0.7..1.2]$ $\quad D_o = \{p(Z, a), q(Z, W)\}$ $\quad D_b = \{W \geq 0.7, W \leq 1.2\}$

as well as the ground clause D' for $\theta = \{X \leftarrow x\}$,

$D' : p(x) = a \leftarrow q(x) = 0.9$ $\quad D'_o : \{p(x, a), q(x, 0.9)\}$ $\quad D'_b = \emptyset$

but it does not θ-subsume the clause:

$D'' : p(x) = a \leftarrow q(x) = 1.9$ $\quad D''_o : \{p(x, a), q(x, 1.9)\}$ $\quad D''_b = \emptyset$

since the set of $C_b\theta$ solutions is empty.

Finally, we observe that the resolution of two definite clauses C and D expressed in ATRE's hypothesis language $\mathcal{L}_H$, corresponds to the classical resolution principle, since the heads of C and D are always simple literals. Consequently, the generalized implication definition given in Section 4 can be easily applied to ATRE.

6.2. Algorithmic issues

The main procedure of ATRE is shown in Figure 6. The system input is a set of objects, a background knowledge, a set of concepts to be learned, and a preference criterion that guide the heuristic search in the space of possible hypotheses.

To illustrate the algorithm, let us consider the input data in Table 1.

The first step towards the generation of inductive hypotheses is the *saturation* of all objects with respect to the given *BK* [61], so that information that was implicit in the example, given the background knowledge, is made explicit (procedure *saturate_objects*). In the above example, the saturation of O_1 involves the addition of the nine literals logically entailed by BK, that is, *close_to(zone*$_2$*, zone*$_1$*)*, *close_to(zone*$_1$*, zone*$_3$*)*, *close_to(zone*$_3$*, zone*$_1$*)*, *close_to(zone*$_7$*, zone*$_1$*)*, *close_to(zone*$_4$*, zone*$_2$*)*, *close_to(zone*$_5$*, zone*$_4$*)*, *close_to(zone*$_5$*, zone*$_6$*)*, *close_to(zone*$_6$*, zone*$_5$*)* and *close_to(zone*$_8$*, zone*$_6$*)*.

Initially, all positive and negative examples (pairs $\langle L$,*OID*$\rangle$) are generated for every concept to be learned, the learned theory is empty and the set of concepts to be learned contains all K_i. With reference to the above input data, the system generates two positive examples for K_1 (*downtown(zone*$_1$*)* and *downtown(zone*$_7$*)*), two positive examples for K_2 (*residential(zone*$_2$*)* and *residential(zone*$_4$*)*) and eight negative examples equally distributed between K_1 and K_2 ($\neg$*downtown(zone*$_2$*)*, $\neg$*downtown(zone*$_3$*)*, $\neg$*downtown(zone*$_4$*)*, $\neg$*downtown(zone*$_5$*)*, $\neg$*residential(zone*$_1$*)*, $\neg$*residential(zone*$_5$*)*, $\neg$*residential(zone*$_6$*)*, $\neg$*residential(zone*$_7$*)*).

The conquer stage performs a general-to-specific beam search to generate a set of consistent, linked and range-restricted clauses for the concepts to be learned. A seed is associated with each specialization hierarchy. Seeds are chosen according to the textual order in which objects are given to the system. If O_k is the first object with an example still uncovered of concept K_i, then O_k is taken to generate seeds for K_i. In particular, all examples of K_i in O_k still uncovered will be selected as seeds, so it is possible to have several specialization hierarchies for each concept.

Ground literals in the body of seed objects are generalized. In particular, the generalization of a ground literal $f(t_1, \ldots, t_n) = Value$ is obtained by turning distinct constants into distinct variables,

procedure *learn_recursive_theories(O, BK,* $\{K_1, \ldots, K_n\}$*, PC)*
  Saturated_O := saturate_objects(O, BK)
  O := Saturated_O
  E := generate_positive_negative_examples (O, $\{K_1, \ldots, K_n\}$*)*
  T := $\emptyset$
  Learned_K := $\{K_1, ..., K_n\}$
  repeat
    Consistent_clauses := parallel_conquer_by_beam_search(Learned_K, E, PC)
    C := find_best_clause(Consistent_clauses, PC)
    Consistent_T := verify_global_consistence(C, T, E, O)
    T := Consistent_T $\cup$ $\{C\}$
    O := saturate_objects(Saturated_O,T)
    E := update_examples(T, E)
    foreach $K_i \in$ *Learned_K* ***do***
      if *pos_example(*K_i*)=*$\emptyset$ ***then*** *Learned_K := Learned_K* $\setminus \{K_i\}$ ***endif***
    endforeach
  until *Learned_K =* $\emptyset$
  return T

Figure 6. ATRE 2.0: Main procedure.

Table 1. An example of input data to the main procedure

Objects	O_1	*downtown(*$zone_1$*)*$\wedge\neg$*residential(*$zone_1$*)*$\wedge$ *residential(*$zone_2$*)*$\wedge$ $\neg$*downtown(*$zone_2$*)* $\wedge\neg$*downtown(*$zone_3$*)* $\wedge$ *residential(*$zone_4$*)* $\wedge$ $\neg$*downtown(*$zone_4$*)* $\wedge\neg$*downtown(*$zone_5$*)* $\wedge\neg$*residential(*$zone_5$*)* $\wedge$ $\neg$*residential(*$zone_6$*)* $\wedge$ *downtown(*$zone_7$*)* $\wedge\neg$*residential(*$zone_7$*)* $\leftarrow$ *onthesea(*$zone_1$*), high_business_activity(*$zone_1$*), close_to(*$zone_1$*,*$zone_2$ *),* *low_business_activity(*$zone_2$*), close_to(*$zone_2$*,*$zone_4$*),* *adjacent(*$zone_1$*,*$zone_3$*), onthesea(*$zone_3$*),* *low_business_activity(*$zone_3$*), low_business_activity(*$zone_4$*),* *close_to(*$zone_4$*,*$zone_5$*), high_business_activity(*$zone_5$*),* *adjacent(*$zone_5$*,*$zone_6$*), low_business_activity(*$zone_6$*),* *close_to(*$zone_6$*,*$zone_8$*), low_business_activity(*$zone_8$*),* *close_to(*$zone_1$*,*$zone_7$*), onthesea(*$zone_7$*), high_business_activity(*$zone_7$*)*
BK		*close_to(X,Y)* $\leftarrow$ *adjacent(X,Y)* *close_to(X,Y)*$\leftarrow$ *close_to(Y,X)*
Concepts	K_1 K_2	*downtown(_)=true* *residential_zone(_)=true*
PC		Minimize/maximize the number of negative/positive examples explained.

and replacing all occurrences of a constant t_i with the same variable X_i (*simple inverse substitution* [61]). Clause specialization is performed either by adding a new generalized seed literal that preserves the property of linkedness of the clause or by restricting the interval of a set literal already in the body. When a consistent, range-restricted clause is found it is put aside: The search stops when at least M consistent, range-restricted clauses have been determined.

In the above example seeds are generated from the unique object O_1. The procedure *parallel_conquer_by_beam_search* generates a set of consistent clauses, whose minimum number is defined by the user. By requiring the generation of at least one consistent clause with respect to the above examples, this procedure returns the following set of clauses:

$downtown(X) \leftarrow onthesea(X), high\_business\_activity(X)$.

$downtown(X) \leftarrow onthesea(X), adjacent(X, Y)$.

$downtown(X) \leftarrow adjacent(X, Y), onthesea(Y)$.

In fact, the hypothesis space of the concept *residential* has been simultaneously explored, but when only the three consistent clauses for the concept *downtown* have been found, no consistent clause for *residential* has yet been discovered. Thus, the parallel-conquer procedure stops, since the number of consistent clauses is greater than one.

At this point, the best one is selected according to the user's preference criterion (procedure *find_best_clause*). The default criterion is the maximization of the number of positive examples covered and the minimization of the complexity of the clause (here represented by the number of literals in the body). In the above example, the first of the three clauses is selected.

Since the addition of a consistent clause may lead to an augmented, inconsistent theory, ATRE applies the layering technique explained in Section 5 to recover the consistency (procedure *verify_global_consistence*). The only difference between the procedure reported in Figure 5 and that invoked in ATRE's main procedure is that also the set of objects O has to be passed in order to reconstruct the body of the examples in E. The selected clause is used to re-saturate the object, so that recursive clauses could be generated in the next call of the procedure *parallel_conquer_by_beam_search*. Continuing the previous example, the two literals added to O_1 are $downtown(zone_1)$ and $downtown(zone_7)$. This operation enables ATRE to also generate the definition of the concept *residential*, which depends on the concept *downtown*.

Finally, the procedure *update_examples* tags positive examples explained by the current learned theory, so that they will no longer be considered for the generation of new clauses. The loop terminates when all positive examples are tagged, which means that the learned theory is complete and consistent. In the above example, $\langle downtown(zone_1), O_1\rangle$ and $\langle downtown(zone_7), O_1\rangle$ are tagged, that is, a complete definition of $downtown(\_) = true$ has been learned. Since not all positive examples are tagged, the procedure *parallel_conquer_by_beam_search* is re-invoked and returns the clause:

$residential(X) \leftarrow close\_to(X, Y), downtown(Y), low\_business\_activity(X)$.

By re-saturating the object with both learned clauses, it becomes possible to generate a recursive clause at the third iteration, namely:

$residential(X) \leftarrow close\_to(X, Y), residential(Y), low\_business\_activity(X)$.

Therefore, the following sets of clauses is learned:

$downtown(X) \leftarrow onthesea(X), high\_business\_activity(X)$

$residential(X) \leftarrow close\_to(X, Y), downtown(Y), low\_business\_activity(X)$

$residential(X) \leftarrow close\_to(X, Y), residential(Y), low\_business\_activity(X)$

which is a simple recursive theory.

6.3. Computational complexity

The generation of consistent clauses requires the exploration of a search space whose size is finite but increases exponentially with the number of literals in the bodies of the selected seeds. Indeed, all clauses in the specialization hierarchy of a seed example $e^+ = \langle L, OID \rangle$ will be obtained by adding a set of literals to the following clause:

$$f(X_1, \ldots, X_n) = Value \leftarrow$$

obtained by turning all constants in L into variables. Since literals used in the specialization process must be generalizations of literals in the body of e^+ obtained by turning constants to variables and possibly by determining an interval (in the case of linear descriptors), the number of clauses is $2^{|body(e^+)|}$.

Nevertheless, ATRE explores only a polynomially bounded portion of this space. More precisely, at the first step at most $|body(e^+)|$ clauses will be considered. Soon afterwards, P of them are selected for the next specialization step (P is the beam of the search). During the second step each selected hypothesis can be specialized in at most $|body(e^+)| - 1$ different ways. In general, at the i-th step, each selected hypothesis can be specialized in at most $|body(e^+)| - i - 1$ different ways. To sum up, the number of generated clauses is:

$$|body(e^+)| + \sum_{i=1}^{|body(e^+)|-1} P \cdot i = |body(e^+)| + \frac{P}{2} |body(e^+)| \cdot (|body(e^+)| - 1)$$

This analysis confirms the efficiency of the system while searching for a consistent clause, since the number of tested hypotheses is linear in the beam of the search, and quadratic in the maximum number of literals of a training object. Neither the arity of the function symbols nor the number of variables in any learned clause affect the cost of the search, as it happens in other systems [57]. Since the number of specialization hierarchies equals the number of training examples at worst, we can conclude that the computational complexity of the procedure *parallel_conquer_by_beam_search* is polynomial in the number of training examples, in the beam of the search, and in the maximum number of literals of a training object.

Unfortunately, this analysis does not take into account the fact that the saturation of objects may increase the number of literals of a training object. In the worst case, such a number might be exponential in the number of constants in the body of objects, thus making the upper bound of the computational complexity of the search exponential.

7. Experimental results

ATRE has been implemented in Prolog and C++. In this section we show the results for some multiple concept learning problems. The first experiments refer to the domain of family relations used to test MPL [15]. Then we present a real world application, namely, document image understanding, where interrelated definitions of logic components are possible. Other results concerning the induction of a mutual recursive theory for odd and even numbers and an application to cognitive modeling are reported in [42] and are available in the web site of the system. Finally, ATRE has been applied to geographical knowledge discovery [41], although the main feature of the system tested in that application is the handling of numerical attributes and relations in a first-order context.

7.1. The domain of family relations

De Raedt and Lavrač [15] defined a class of experiments on the domain of family relations. The first experiment aims at learning the definitions of ancestor, father and mother from a complete set of positive and negative examples. In MPL the negative examples were generated under close world assumption, and the knowledge base contained some ground atoms concerning the predicates *male*, *female* and *parent*. A convenient representation for ATRE is a single object in which all positive and negative examples of ancestor, father and mother are explicitly reported in the head, while all instances of male, female and parent are reported in the body.

The simple recursive theory learned by MPL is the very elegant:

$ancestor(X,Y) \leftarrow parent(X,Y)$

$father(X,Y) \leftarrow parent(X,Y), male(X)$

$mother(X,Y) \leftarrow parent(X,Y), female(X)$

$ancestor(X,Y) \leftarrow parent(X,Z), ancestor(Z,Y).$

Clauses are reported in the order in which they are learned. For this problem ATRE learns a correct theory in 68s on a PentiumIII PC – 1GHz, though different and less intuitive, namely:

$ancestor1(X1,X2) \leftarrow parent(X1,X2)$

$father(X1,X2) \leftarrow ancestor1(X1,X2), male(X1)$

$mother(X1,X2) \leftarrow ancestor1(X1,X2), female(X1)$

$ancestor(X1,X2) \leftarrow ancestor1(X1,X2)$

$ancestor(X1,X2) \leftarrow father(X1,X3), ancestor(X3,X2)$

$ancestor(X1,X2) \leftarrow mother(X1,X3), ancestor(X3,X2).$

In the above theory a new predicate *ancestor1* has been 'invented' due to consistency recovery. Taking into account that *ancestor1* and *parent* are semantically equivalent, the interpretation of the above theory is clearer. The explanation of this result is straightforward. The first three clauses generated by ATRE are:

C_1: $ancestor(X1,X2) \leftarrow parent(X1,X2)$

C_2: $father(X1,X2) \leftarrow ancestor(X1,X2), male(X1)$

C_3 : $mother(X1,X2) \leftarrow ancestor(X1,X2), female(X1).$

In fact, ATRE overgeneralizes the definitions of *father* and *mother*, but at this point of the learning process the system does not know the complete definition of *ancestor* and considers the clauses

$father(X1,X2) \leftarrow parent(X1,X2), male(X1)$

$mother(X1,X2) \leftarrow parent(X1,X2), female(X1)$

equivalent to C_2 and C_3 respectively. ATRE discovers its overgeneralization error when the new clause

C_4 : $ancestor(X1,X2) \leftarrow father(X1,X3), ancestor(X3,X2)$

is added to the theory. At this point, the system applies the consistency recovering strategy and invents the new predicate *ancestor1*.

It is noteworthy that these results are obtained by using a beam equal to 5. By enlarging the beam to 15, ATRE takes 180s to learn a theory analogous to that induced by MPL, namely:

$ancestor(X,Y) \leftarrow parent(X,Y)$

$ancestor(X,Y) \leftarrow parent(X,Z), ancestor(Z,Y)$

$father(X,Y) \leftarrow parent(X,Y), male(X)$

$mother(X,Y) \leftarrow parent(X,Y), female(X).$

This example also suggests some improvement of the system. Currently ATRE checks that $C_2 \preceq_{\{C1\}\cup BK,\Rightarrow} e$ for each example e of the concept *father*. The only examples of ancestor that are considered in this test are those inferred by the partial definition C_1. By augmenting the BK with the set OH of *all* positive examples of ancestor available in the head of the object, that is, by testing $C_2 \preceq_{\{C1\}\cup BK\cup OH\Rightarrow} e$, ATRE would not overgeneralize, at least not in this experiment, in which all positive and negative examples of ancestor are input to the system. The reason for this modified test is that, sooner or later, ATRE will generate clauses to cover all examples in OH. So we can anticipate some of the conclusions to be drawn from prospectively generated clauses. This trick has also been adopted in the work by Lamma *et al.* [33]. However, it does not work when a set of positive and negative examples is incomplete, which explains why the consistency recovery strategy based on theory layering is still necessary.

The second experiment of the family domain aims at learning *male_ancestor* and *female_ancestor* from father and mother. Once again, the training set is complete. MPL learned the following theory:

$female\_ancestor(X,Y) \leftarrow mother(X,Y)$
$male\_ancestor(X,Y) \leftarrow father(X,Y)$
$female\_ancestor(X,Y) \leftarrow mother(X,Z), female\_ancestor(Z,Y)$
$male\_ancestor(X,Y) \leftarrow father(X,Z), female\_ancestor(Z,Y)$
$male\_ancestor(X,Y) \leftarrow father(X,Z), male\_ancestor(Z,Y)$
$female\_ancestor(X,Y) \leftarrow female\_ancestor(X,Z), male\_ancestor(Z,Y).$

ATRE learned a slightly different but equally correct reformulation of the MPL's theory, that is:

$male\_ancestor(X1,X2) \leftarrow father(X1,X2)$
$female\_ancestor(X1,X2) \leftarrow mother(X1,X2)$
$male\_ancestor(X1,X2) \leftarrow male\_ancestor(X1,X3), female\_ancestor(X3,X2)$
$male\_ancestor(X1,X2) \leftarrow male\_ancestor(X1,X3), male\_ancestor(X3,X2)$
$female\_ancestor(X1,X2) \leftarrow female\_ancestor(X1,X3), female\_ancestor(X3,X2)$
$female\_ancestor(X1,X2) \leftarrow male\_ancestor(X3,X2), female\_ancestor(X1,X3).$

The beam used in this experiment is 5 and the learning time is 660s.

Finally, the third experiment on the family domain aims at learning *father* and *grandfather* from an incomplete example set. The example set contains all positive examples of *father* and *grandfather*. The negative examples are complete for grandfather and incomplete for father, since they are generated by means of the rule:

$father(X,Y) = false \leftarrow parent(X,Y) = false.$

Results reported for MPL are:

C_5: $father(X,Y) \leftarrow parent(X,Y)$
C_6: $grandfather(X,Y) \leftarrow male(X), parent(X,Z), parent(Z,Y)$

while the theory learned by ATRE in 530s is:

C_7 : $father(X1,X2) \leftarrow parent(X1,X2)$
C_8: $grandfather(X1,X2) \leftarrow male(X1), father(X1,X3), father(X3,X2).$

It is noteworthy that $C_8 \preceq_{\{C7\}\Rightarrow} C_6$, that is ATRE generates a more general theory. This means that ATRE is more prone to make an error as soon as a mother is seen, since the definition of father is wrong. However, ATRE also generated C_6 and considered it indistinguishable from C_8, with respect to covered examples. Therefore, the result is simply influenced by the order in which these equivalent clauses have been added to the set of solutions.

7.2. Application to the document image understanding problem

ATRE has also been also applied to the problem of processing printed documents and its induced logical theories are used by an intelligent document processing system, named WISDOM++ (see the web site http://www.di.uniba.it/∼malerba/wisdom++/) [20]. Henceforth, only the specific problem of learning rules for document image understanding will be dealt with. The main innovation with respect to previous work is the automated discovery of possible concept dependencies, as well as the consideration of multi-page documents.

A document is characterized by two different structures representing both its internal organization and its content: the *layout* (or *geometrical*) structure and the *logical* structure. The former associates the content of a document with a hierarchy of layout objects, such as text lines, vertical/horizontal lines, graphic/photographic elements, pages, and so on. The latter associates the content of a document with a hierarchy of logical objects, such as sender/receiver of a business letter, title/authors of an article, and so on. Here, the term *document image understanding* denotes the process of mapping the layout structure of a document into the corresponding logical structure. The document image understanding process is based on the assumption that document images can be understood on the basis of their layout structures alone.

The mapping of the layout structure into the logical structure can be represented as a set of rules. Traditionally, such rules were hand-coded for particular kinds of document [53], requiring much human tuning and effort. We propose the application of inductive learning techniques to generate the rules automatically from a set of training examples. The user-trainer is asked to label some layout components of a set of training documents according to their logical meaning. Those layout components with no clear logical meaning are not labeled. Therefore, each document generates as many training examples as the number of layout components.

In this application, we have a multi-class learning problem. Concepts correspond to the distinct logical components to be recognized in a document. They are defined by different values taken by the descriptor *logic_type*. The unlabelled layout objects act as counterexamples for all the concepts to be learned, since they are instances of the concept $logic\_type(X) = undefined$.

Each training document is represented as an object in ATRE, where different constants represent distinct layout components of a page. The description of a document page is reported in Figure 7, while Table 2 lists all the descriptors used to represent a page layout of a multi-page document.

The following four rules are used as background knowledge, in order to automatically associate information on page order to layout blocks.

$at\_page(X) = first \leftarrow part\_of(Y, X), page(Y) = first$
$at\_page(X) = intermediate \leftarrow part\_of(Y, X), page(Y) = intermediate$
$at\_page(X) = last\_but\_one \leftarrow part\_of(Y, X), page(Y) = last\_but\_one$
$at\_page(X) = last \leftarrow part\_of(Y, X), page(Y) = last$

Three long papers, which appeared in the January 1996 issue of the IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), have been considered. The papers contain thirty-seven pages, each of which has a variable number of layout components (about ten on average). The total number of layout components is 380 but only 170 of them can be associated with one of the following eleven logical labels: *abstract*, *affiliation*, *author*, *biography*, *caption*, *figure*, *index_term*, *page_number*, *references*, *running_head*, *title*. Remaining blocks are considered instances of the logical component *body*, for which we are not interested in learning recognition rules.

Table 2. Descriptors used by WISDOM++ to represent the page layout of a multi-page document

Descriptor	***Domain***
page(page)	*Nominal domain:* first, intermediate, last_but_one, last
width(block)	*Integer domain:* (1..640)
height(block)	*Integer domain:* (1..875)
x_pos_centre(block)	*Integer domain:* (1..640)
y_pos_centre(block)	*Integer domain:* (1..875)
type_of(block)	*Nominal domain:* text, hor_line, image, ver_line, graphic, mixed
part_of(page,block)	*Boolean domain:* true if page contains block
on_top(block1,block2)	*Boolean domain:* true if block1 is above block2
to_right(block1,block2)	*Boolean domain:* true if block2 is to the right of block1
alignment(block1,block2)	*Nominal domain:* only_left_col, only_right_col, only_middle_col, both_columns, only_upper_row, only_lower_row, only_middle_row, both_rows

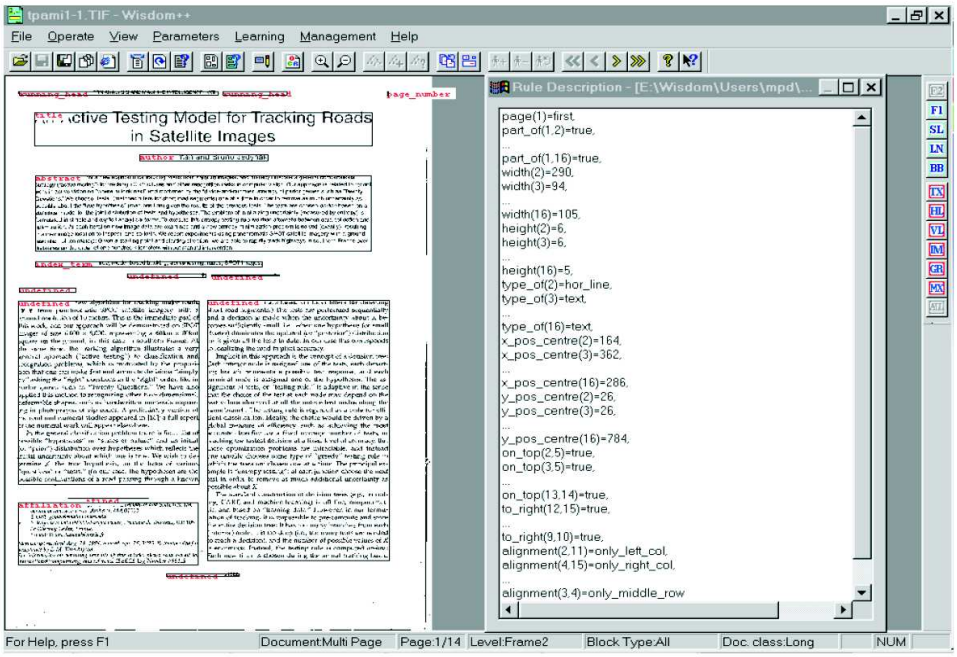

Figure 7. Layout of the first page of a multi-page document (left) and its partial description in a first-order logic language (right)

Learning rules for the recognition of semantically relevant layout components in a document raises issues concerning the induction of recursive theories. Simple and mutual concept dependencies are to be handled, since the logical components refer to a part of the document rather than to the whole document and they may be related to each other. For instance, in the case of papers published in journals, the following dependent clauses:

$running\_head(X) \leftarrow top\_left(X), text(X), even\_page\_number(X)$

$running\_head(X) \leftarrow top\_right(X), text(X), odd\_page\_number(X)$

$paragraph(Y) \leftarrow ontop(X,Y), running\_head(X), text(Y)$

express the fact that a textual layout component at the top left (right) hand corner of an odd (even) page is a running head, while a textual layout component below a running-head is a paragraph of the paper. Moreover, the recursive clause

$paragraph(Y) \leftarrow ontop(X,Y), paragraph(X), text(Y)$

is useful to classify all textual layout components below the upper-most paragraph. Therefore, document understanding seems to be the kind of application that may benefit from learning strategies for multiple predicate learning.

By running ATRE on the training set described above, the theory in Table 3 is returned. The beam used in this experiment is 15 and the learning time is about 1,086s. Clauses are reported in the order in which they are learned. They have been filtered out from candidate clauses during the evaluation step, performed according to a composite preference criterion that minimizes the number of negative examples covered, maximizes the number of positive examples covered, and minimizes the cost of the clause. The theory contains some concept dependencies (see clauses 11 and 15, 17, 19 and 20), which are all plausible. The significance of these clauses is more evident by looking at the number of positive examples covered by each of them (see the fourth column). In particular ATRE discovers the following concept dependencies: *page_number* ← *running_head* and *figure* ← *caption*. Therefore, the learned theory has two distinct layers: *abstract, affiliation, author, biography, figure, index_term, page_number, references,* and *title* are in one layer, while *running_head* and *caption* are in the other.

In order to test the predictive accuracy of the learned theory, we considered the fourth long article published in the same issue of the transactions used for training. WISDOM++ segmented the fourteen pages of the article into 169 layout components, sixty of which were instances of one of the concepts used in the training set, while the remaining 109 are instances of the concept *body* for which no rule was generated. The learned theory may commit both omission and commission errors. In particular, it performed sixteen omission errors, that is about 26% (16/60) of the blocks to be labeled were not correctly recognized by the theory, and fourteen commission errors, that is about 1.1% (14/(109×11+60×10)) of possible commission errors (the worst case being the association of each block with all concepts except for the correct one). Many of the omission errors are due to near misses. For instance, the title of the first page is not recognized simply because its height is 54, while the range of *width* values determined by ATRE in the training phase is [18..53] (see clause 13). Significant recovery of omission errors can be obtained by relaxing the definition of subsumption between definite clauses [21].

8. Discussion

In this paper we have discussed and proposed computational solutions to some relevant issues raised by the induction of recursive theories in the normal ILP setting. A separate-and-parallel-conquer search

1	logic_type(X)=running_head ← height(X) ∈ [6..9], y_pos_centre(X) ∈ [18..35], width(X) ∈ [77..544]	36	36
2	logic_type(X) = figure ← height(X) ∈ [90..313], width(X) ∈ [9..255], at_page(X)=intermediate	20	20
3	logic_type(X)=page_number ← y_pos_centre(X) ∈ [19..29], width(X) ∈ [2..8]	14	14
4	logic_type(X)=page_number ← width(X)∈[4 .. 8], y_pos_centre(X)∈[22..40]	10	20
5	logic_type(X)=figure ← type_of(X)=image, at_page(X)=intermediate	7	17
6	logic_type(X)=figure ← type_of(X)=graphic	25	35
7	logic_type(X)=abstract ← at_page(X)=first, width(X)∈[487..488]	3	3
8	logic_type(X)=affiliation ← at_page(X)=first, y_pos_centre(X)∈[720..745]	3	3
9	logic_type(X)=author ← at_page(X)=first, y_pos_centre(X)∈[128..158]	3	3
10	logic_type(X)=biography ← at_page(X)=last, height(X)∈[65..234]	3	3
11	logic_type(X)=caption ← alignment(Y,X)=only_middle_col, logic_type(Y)=figure, type_of(X)=text, height(X)∈[18 .. 75]	13	13
12	logic_type(X)=index_term ← height(X)∈[8 .. 8], y_pos_centre(X)∈[263..295]	3	3
13	logic_type(X)=title ← at_page(X)=first, height(X)∈[18 .. 53]	5	5
14	logic_type(X)=references ← width(X)∈[268..268], height(X)∈[332..355]	2	2
15	logic_type(X)=running_head ← alignment(Y,X)=only_upper_row, logic_type(Y)=page_number	2	14
16	logic_type(X)=references ← height(X)∈[339..355]	2	3
17	logic_type(X)=caption ← height(X)∈[9..40], on_top(Y,X), logic_type(Y)=figure, width(Y)∈[5..364]	11	22
18	logic_type(X)=caption ← height(X)∈[9..9], y_pos_centre(X)∈[417..605]	2	4
19	logic_type(X)=caption ← on_top(Y,X), alignment(Y,Z)=only_lower_row, logic_type(Y)=figure	1	2
20	logic_type(X)=caption ← width(X)∈[419..546], on_top(X,Y), logic_type(Y)=figure	5	9
	Total	**170**	

Table 3. Theory learned for the document image understanding problem. The third column reports the number of additional training examples explained by each clause when added to the theory. The fourth column reports the number of training examples actually explained by each clause.

strategy has been adopted to synchronize and interleave the learning of clauses by supplying predicates with mutually recursive definitions. A novel generality order, called generalized implication, has been imposed on the search space of clauses, in order to cope with recursion in a more suitable way. A layering technique based on the collapsed dependency graph has been investigated to recover the consistency of a partially learned theory. These ideas have been implemented in the ILP system ATRE, which is also characterized by the object-centered representation of training examples. Some experimental results are reported for some laboratory-sized data as well as for data obtained by processing multi-page document images.

We continue this work by discussing both related researches and ideas for future developments.

8.1. Related work

ATRE presents several innovations with respect to previous works on multiple predicate learning, namely MPL [16, 15] and its extension to normal logic programs NMPL [24], MULT_ICN [45], and RTL [26].

First, the *separate-and-parallel-conquer* search strategy provides ATRE with a solution to the problem of interleaving the induction process for distinct predicate definitions. According to this strategy, the generation of clauses that introduce a (mutual) dependence of a predicate p on a predicate q (or p itself) is possible only after at least one recursion base clause for q (p) has been found. Moreover, ATRE operates in a one-shot way, that is, with no additional synthesis step, as in RTL, and the iterative bootstrap induction method by Jorge and Brazdil [30].

MPL solves the interleaving problem by performing a greedy hill-climbing search for theories and a beam search for each single clause. Clauses can be generated by means of two types of refinement operators, one for the body and one for the head. In particular, it is possible to generate the body of a clause without specifying its head. The subsequent refinement of the heads introduces possibly different predicates and the system chooses the most promising clause according to an interestingness measure [16]. The generation of clauses like $p \leftarrow p$ is not forbidden, and it is possible to generate a recursive clause for a predicate p, before a base clause for p has been found.

The strategy adopted in NMPL is slightly different from that of MPL. At the high level NMPL follows the classical separate-and-conquer search strategy. The innovation is in the conquer strategy, where the same refinement (literal addition) is applied to clauses with different heads. The heuristics for evaluating the best refinement in this greedy conquer strategy is based on the Laplace estimate [17]. Unfortunately, two aspects are unclear. First, how to specialize with the same literal, say $less\_than(A, B)$, two clauses whose heads have different arity, such as, $ordered(L)$ and $between(X, Y, Z)$. Second, how the generation of infinite recursive definitions (e.g., $p \leftarrow p$) is prevented or dealt with.

A completely different approach is adopted in MULT_ICN. Again, at the high level, the separate-and-conquer search strategy is used. During the conquer stage, the system chooses the predicate to be learned. Preference is given to those target predicates whose definition is incomplete (that is, not all positive examples have been covered) and appear in the body of previously learned clauses. For instance, if the first generated clause is the following:

$odd(X) \leftarrow succ(Y, X), even(Y)$

at the second step the system tries to learn the definition of *even(Y)*. In other words, base clauses are the last to be learned.

The generalization model represents another difference between ATRE and the other multiple predicate learning systems. MPL adopts two different generalization models during its search: θ-subsumption,

while learning a single clause, and logical entailment, while learning the whole theory. Therefore, two distinct checks are performed by the system for each learned clause: a *local* consistency/completeness check based on θ-subsumption (*extensional* coverage) and a *global* check based on logical entailment (*intensional* coverage). A similar approach is also adopted by MULT_ICN. However, as pointed out by Martin and Vrain [45], the extensional coverage test can lead to the generation of non-terminating, but extensionally valid, theories, such as the following:

$even(A) \leftarrow zero(A)$
$odd(A) \leftarrow succ(A, B), even(B)$
$even(A) \leftarrow succ(B, A), odd(B)$

which defines odd and even numbers on the basis of their successors (and not predecessors). To avoid this problem, MULT_ICN introduces an acceptability criterion for a clause and lets the user choose the right rate of acceptability. MPL solves the problem differently, by deleting globally incorrect and irrelevant clauses added to the theory. Therefore, when the clause

$odd(A) \leftarrow succ(A, B), even(B)$

is generated as the best clause, it is first added to the theory and soon after removed, because it is globally irrelevant (no positive examples of odd numbers are covered). However, in this approach it is not clear how to select the clause to remove, when more than one is globally irrelevant or inconsistent. Moreover, care should be taken not to get into infinite loops by first deleting clauses and then adding them again.

Similarly to Progol, MPL uses a depth-bounded interpreter to check whether an induced theory logically entails an example. This depth-bound allows the system to check both consistency and completeness properties in the case of infinite recursive definitions (e.g., $p \leftarrow p$). Although not explicitly stated, a similar depth-bounded approach should be adopted in NMPL, where logical entailment is used to check the properties of completeness and consistency of a clause.

Systems that adopt the extensional (or θ-subsumption based) coverage tests have an additional problem in learning recursive clauses when examples are sparse. This problem is less evident in ATRE, which can learn the correct definition of odd and even numbers, also when the example set is incomplete [42].

Finally, problems caused by the non-monotonicity property of the normal ILP setting have not been considered in some multiple predicate learning systems, such as MULT_ICN and NMPL. As explained in Section 6, this is a crucial aspect of the multiple predicate learning problem. Progol, for instance, cannot be properly considered a multiple predicate learning system since it can induce theories that are globally inconsistent. MPL solves the problem by means of the clause deletion technique, although undoing previous work may considerably increase the learning time. ATRE adopts the layering technique and invents a new predicate, when the addition of a clause to the theory interferes with another clause already added to the theory. As shown in Section 7.1, the application of the layering technique does not prevent ATRE from discovering equally correct theories, which could be later simplified by keeping the minimum Herbrand model (*theory restructuring*) [62, 63].

8.2. Further work

The current implementation of ATRE is not optimal. One of the reasons is that every time a clause is added to the theory, the specialization hierarchies are reconstructed for a new set of seeds, which may intersect the set of seeds explored in the previous step. In other words, it is possible that the system explores the same specialization hierarchies several times, since it has no memory of the work done in previous steps. Currently, we are optimizing the separate-and-parallel-conquer search strategy to stop it

exploring the specialization hierarchies repeatedly during the learning process. This approach is based on complex caching techniques, whose effectiveness is clear when concepts to learn are neither recursively definable nor mutually dependent.

Another important aspect is the abundance of candidate consistent clauses that the ATRE's *parallel_conquer_by_beam_search* procedure can generate. Currently, only the best clause with respect to the user's preference criterion is selected, but, even in this case, we observed that often many 'equivalent' clauses exist, given a preference criterion. This means that ATRE makes a blind choice among many equally good clauses. Whether better selection strategies exist and how sets of equivalent clauses can be used in recursive theory learning is still an open question.

Finally, the well-known issues of sparse and noisy training sets make the problem of learning recursive theories even harder. The integration of abductive mechanisms in inductive learning algorithms can be a solution to the problems of data sparseness and class noise [22], while the presence of noise in the background knowledge requires probabilistic tests, similar to the extension of the θ-subsumption test proposed by [21].

9. Web site

A stand-alone release of the ATRE 2.0 system together with the data sets used in the experiments can be downloaded from http://www.di.uniba.it/∼malerba/software/atre/.

10. Acknowledgments

The author is grateful to Floriana Esposito and David Lorenzo for interesting discussions and suggestions concerning this work. Thanks to Margherita Berardi and Francesca A. Lisi for their assistance with some experiments. The author also wishes to thank Michelangelo Ceci and Lynn Rudd who devoted much time reading the manuscript.

References

[1] Aha, D. W., Lapointe, S., Ling, C. X., Matwin, S.: Learning recursive relations with randomly selected small training sets, *Proc. Eleventh International Conference on Machine Learning*, 1994.

[2] Apt, K. R.: Logic programming, in: *Handbook of Theoretical Computer Science* (J. van Leeuwen, Ed.), vol. B, Elsevier, Amsterdam, 1990, 493–574.

[3] Bergadano, F., Gunetti, D.: Learning clauses by tracing derivations, in: *Proc. of the Fourth International Workshop on Inductive Logic Programming* (S. Wrobel, Ed.), vol. 237 of *GMD-Studien*, 1994.

[4] Bergadano, F., Gunetti, D.: *Inductive Logic Programming: from machine learning to software engineering*, The MIT Press, Cambridge, MA, 1996.

[5] Boström, H.: Specialization of recursive predicates, in: *Machine Learning: ECML-95* (N. Lavrač, S. Wrobel, Eds.), vol. 912 of *LNAI*, Springer-Verlag, Berlin, 1995, 92–106.

[6] Boström, H.: Induction of Recursive Transfer Rules, in: *Learning Language in Logic* (J. Cussens, S. Džeroski, Eds.), vol. 1925 of *LNAI*, Springer-Verlag, Berlin, 2000, 237–246.

[7] Bratko, I.: Applications of machine learning: towards knowledge synthesis, *New Generation Computing*, **11**, 1993, 343–360.

[8] Buntine, W.: Generalised subsumption and its applications to induction and redundancy, *Artificial Intelligence*, **36**, 1988, 149–176.

[9] Cameron-Jones, R. M., Quinlan, J. R.: Avoiding pitfalls when learning recursive theories, *Proc. Twelfth International Joint Conference on Artificial Intelligence*, 1993.

[10] Ceri, S., Gottlob, G., L.Tanca: What you always wanted to know about Datalog (and never dared to ask), *IEEE Transactions on Knowledge and Data Engineering*, **1(1)**, 1989, 146–166.

[11] Cohen, W. W.: Learnability of restricted logic programs, *Proc. 3rd International Workshop on Inductive Logic Programming* (S. Muggleton, Ed.), 1993.

[12] De Raedt, L.: *Interactive Theory Revision*, Academic Press, London, 1992.

[13] De Raedt, L., Dehaspe, L.: Clausal discovery, *Machine Learning Journal*, **26(2/3)**, 1997, 99–146.

[14] De Raedt, L., Lavrač, N.: The many faces of inductive logic programming, in: *Methodologies for Intelligent Systems* (J. Komorowski, Z.W.Raś, Eds.), vol. 689 of *LNAI*, Springer-Verlag, 1993, 435–449.

[15] De Raedt, L., Lavrač, N.: Multiple predicate learning in two Inductive Logic Programming settings, *Journal on Pure and Applied Logic*, **4(2)**, 1996, 227–254.

[16] De Raedt, L., Lavrač, N., Džeroski, S.: Multiple predicate learning, *Proc. 13th International Joint Conference on Artificial Intelligence*, 1993.

[17] Džeroski, S., Bratko, I.: Handling Noise in Inductive Logic Programming, *Proc. Second International Workshop on Inductive Logic Programming*, Institute for New Generation Computing Technology, 1992.

[18] Džeroski, S., Lavrač, N.: *Relational Data Mining*, Springer-Verlag, Berlin, 2001.

[19] Esposito, F., D.Malerba, Lisi, F. A.: Induction of recursive theories in the normal ILP setting: issues and solutions, in: *Inductive Logic Programming* (J. Cussens, A. Frisch, Eds.), vol. 1866 of *LNAI*, Springer-Verlag, Berlin, 2000, 93–111.

[20] Esposito, F., Malerba, D., Lisi, F. A.: Machine Learning for Intelligent Processing of Printed Documents, *Journal of Intelligent Information Systems*, **14(2/3)**, 2000, 175–198.

[21] Esposito, F., Malerba, D., Marengo, V.: Inductive Learning from Numerical and Symbolic Data: An Integrated Framework, *Intelligent Data Analysis*, **5(6)**, 2001, 445–461.

[22] Esposito, F., Semeraro, G., Fanizzi, N., Ferilli, S.: Multistrategy Theory Revision: Induction and Abduction in INTHELEX, *Machine Learning Journal*, **38(1/2)**, 2000, 133–156.

[23] Flener, P., Yilmaz, S.: Inductive synthesis of recursive logic programs: achievements and prospects, *Journal of Logic Programming, Special Issue on Synthesis, Transformation, and Analysis*, **41(2-3)**, 1999, 141–195.

[24] Fogel, L., Zaverucha, G.: Normal Programs and Multiple Predicate Learning, in: *Inductive Logic Programming* (D. Page, Ed.), vol. 1446 of *LNAI*, Springer-Verlag, 1998, 175–184.

[25] Gelfond, M., Lifschitz, V.: Logic programs with classical negation, *Proc. of the Seventh International Logic Programming Conference*, MIT Press, Cambridge, 1990.

[26] Giordana, A., Saitta, L., Baroglio, C.: Learning simple recursive theories, in: *Methodologies for Intelligent Systems* (J. Komorowski, Z. Raś, Eds.), vol. 689 of *LNAI*, Springer-Verlag, 1993, 425–434.

[27] Gottlob, G.: Subsumption and Implication, *Information Processing Letters*, **24(2)**, 1987, 109–111.

[28] Idestam-Almquist, P.: *Generalization of clauses*, Ph.D. Thesis, Department of Computer and Systems Sciences, Stockholm University and Royal Institute of Technology, Stockholm, Sweden, 1993.

[29] Idestam-Almquist, P.: Efficient induction of recursive definitions by structural analysis of saturations, in: *Advances in Inductive Logic Programming* (L. De Raedt, Ed.), IOS Press, Amsterdam, 1996, 192–205.

[30] Jorge, A., Brazdil, P.: Architecture for iterative learning of recursive definitions, in: *Advances in Inductive Logic Programming* (L. De Raedt, Ed.), IOS Press, Amsterdam, 1996, 206–218.

[31] Khardon, R.: Learning to take Actions, *Machine Learning Journal*, **35(1)**, 1999, 57–90.

[32] Khardon, R.: Learning Horn Expressions with LogAn-H, *Proc. of the Seventeenth International Conference on Machine Learning*, 2000.

[33] Lamma, E., Mello, P., Milano, M., Riguzzi, F.: Integrating Extensional and Intensional ILP Systems through Abduction, *Proc. of the Seventh International Workshop on Logic Program Synthesis and Transformation (LOPSTR'97)*, 1997.

[34] Lapointe, S., Matwin, S.: Sub-unification: A tool for efficient induction of recursive programs, *Proc. of the Ninth International Conference on Machine Learning*, Morgan Kaufmann, Aberdeeen, 1992.

[35] Lavrač, N., Džeroski, S.: *Inductive Logic Programming: techniques and applications*, Ellis Horwood, Chichester, 1994.

[36] Lavrač, N., Džeroski, S., Bratko, I.: Handling imperfect data in inductive logic programming, in: *Advances in Inductive Logic Programming* (L. De Raedt, Ed.), IOS Press, Amsterdam, 1996, 48–64.

[37] van Leeuwen, J.: Graph Algorithms, vol. A of *Handbook of Theoretical Computer Science*, Elsevier, Amsterdam, 1990, 525–631.

[38] Levi, G., Sirovich, F.: Generalized and-or graphs, *Artificial Intelligence*, **7**, 1976, 243–259.

[39] Lloyd, J. W.: *Foundations of Logic Programming*, Second edition, Springer-Verlag, Berlin, 1987.

[40] Lorenzo, D., Otero, R. P.: Learning to reason about actions, *Proc. of the Fourteenth European Conference on Artificial Intelligence* (W. Horn, Ed.), IOS Press, Amsterdam, 2000.

[41] Malerba, D., Esposito, F., Lanza, A., Lisi, F.: Machine learning for information extraction from topographic maps, in: *Geographic Data Mining and Knowledge Discovery* (H. J. Miller, J. Han, Eds.), Taylor and Francis, London, 2001, 291–314.

[42] Malerba, D., Esposito, F., Lisi, F.: Learning Recursive Theories with ATRE, *Proc. Thirteenth European Conference on Artificial Intelligence* (H. Prade, Ed.), John Wiley & Sons, Chichester, 1998.

[43] Malerba, D., Esposito, F., Semeraro, G., Caggese, S.: Learning simple recursive theories, in: *AI*IA 97: Advances in Artificial Intelligence* (M. Lenzerini, Ed.), vol. 1321 of *LNAI*, Springer-Verlag, 1997, 24–35.

[44] Malerba, D., Semeraro, G., Esposito, F.: A multistrategy approach to learning multiple dependent concepts, in: *Machine Learning and Statistics: The interface* (G. Nakhaeizadeh, C. Taylor, Eds.), John Wiley & Sons, New York, 1997, 87–106.

[45] Martin, L., Vrain, C.: A three-valued framework for the induction of general logic programs, in: *Advances in Inductive Logic Programming* (L. De Raedt, Ed.), IOS Press, Amsterdam, 1996, 219–235.

[46] Michalski, R.: A theory and methodology of inductive learning, in: *Machine Learning - An Artificial Intelligence Approach* (R. Michalski, J. Carbonell, T. Mitchell, Eds.), Tioga Publishing Co., Palo Alto, CA, 1983, 83–133.

[47] Mitchell, T.: *Machine Learning*, McGraw-Hill, 1997.

[48] Mofizur, C., Numao, M.: Top-down induction of recursive programs from small number of sparse examples, in: *Advances in Inductive Logic Programming* (L. De Raedt, Ed.), IOS Press, Amsterdam, 1996, 236–253.

[49] Muggleton, S.: *Inductive Logic Programming*, Academic Press, London, 1992.

[50] Muggleton, S.: Inverting Implication, *Proceedings of the 2nd International Workshop on Inductive Logic Programming* (S. Muggleton, K. Furukawa, Eds.), 1992.

[51] Muggleton, S.: Inverse Entailment and Progol, *New Generation Computing*, **13(3/4)**, 1995, 245–286.

[52] Muggleton, S., Bryant, C.: Theory completion using inverse entailment, in: *Inductive Logic Programming* (J. Cussens, A. Frisch, Eds.), vol. 1866 of *LNAI*, Springer-Verlag, 2000, 130–146.

[53] Nagy, G., Seth, S., Stoddard, S.: A prototype document image analysis system for technical journals, *IEEE Computer*, **25(7)**, 1992, 10–22.

[54] Nienhuys-Cheng, S.-W., de Wolf, R.: A complete method for program specialization based upon unfolding, *Proc. Twelfth European Conference on Artificial Intelligence*, 1996.

[55] Nienhuys-Cheng, S.-W., de Wolf, R.: The Subsumption theorem in inductive logic programming: Facts and fallacies, in: *Advances in Inductive Logic Programming* (L. De Raedt, Ed.), IOS Press, Amsterdam, 1996, 265–276.

[56] Nienhuys-Cheng, S.-W., de Wolf, R.: *Foundations of inductive logic programming*, Springer, Heidelberg, 1997.

[57] Pazzani, M., Kibler, D.: The utility of knowledge in inductive learning, *Machine Learning Journal*, **9**, 1992, 57–94.

[58] Plotkin, G.: A note on inductive generalization, vol. 5 of *Machine Intelligence*, Edinburgh University Press, Edinburgh, 1970, 153–163.

[59] Plotkin, G.: A further note on inductive generalization, vol. 6 of *Machine Intelligence*, Edinburgh University Press, Edinburgh, 1971, 101–124.

[60] Ramakrishnan, R., Srivastava, D., Sudarshan, S.: Rule ordering in bottom-up fixpoint evaluation of logic programs, *IEEE Transactions on Knowledge and Data Engineering*, **6**, 1994, 501–517.

[61] Rouveirol, C.: Flattening and saturation: Two representation changes for generalization, *Machine Learning Journal*, **14(2)**, 1994, 219–232.

[62] Sommer, E.: FENDER: an approach to theory restructuring (extended abstract), in: *Machine Learning: ECML-95* (N. Lavrač, S. Wrobel, Eds.), vol. 912 of *LNAI*, Springer-Verlag, 1994, 356–359.

[63] Sommer, E.: An approach to quantifying the quality of induced theories, *Proc. IJCAI Workshop on Machine Learning and Comprehensibility* (C. Nedellec, Ed.), 1995.

[64] Toussaint, J., Schmid, U., Wysotzki, F.: Using recursive control rules in planning, *Proc. of ICAI Session on Learning and Adapting in AI Planning*, Las Vegas, 2001.

[65] Van Laer, W., De Raedt, L., Džeroski, S.: On Multi-Class Problems and Discretization in Inductive Logic Programming, in: *Proceedings of the 10th International Symposium on Methodologies for Intelligent Systems (ISMIS97)* (Z. Raś, A. Skowron, Eds.), vol. 1325 of *LNAI*, Springer-Verlag, 1997, 277–286.

[66] Zelezny, F., Miksovsky, P., Stepankova, O., Zidek, J.: ILP for Automated Telephony, *Inductive Logic Programming (Work in Progress)* (J. Cussens, A. Frisch, Eds.), 2000.