Semantic Model for Induction of First Order Theories

Celine Rouveirol
Universite Paris Sud,
LRI, Building 490, 91405 Orsay, France
email: celine@lri.lri.fr.uucp

Abstract*

This paper presents an extension of the inverse resolution framework, which is a logical model of theory induction in first order logic. We first propose a definition of completeness for inductive procedures and exhibit some limitations of inverse resolution with respect to this definition. We then propose our model of *inversion of logical entailment* that extends inverse resolution with regard to these limitations. Two operators are formalized that implement this formal framework and we prove that they arc complete. In the last part of the paper, we propose two possible controls for these operators, to restrict the search for an acceptable theory. These two controls express some semantic properties of the theory that the system is searching for.

1 Introduction

There have been considerable research efforts concerning automated induction in the field of Machine Learning, under different subfields such as generalization (Michalsksi, 1983; Kodratoff & Ganascia, 1986; Buntine, 1987), theory refinement (Shapiro, 1982; Duval & Kodratoff, 1990) and concept formation (Sammut & Banerji, 1986; De Raedt & Bruynooghe, 1989). Recently, (Muggleton & Buntine, 1988) have proposed a logical framework for induction of logic programs in first order logic, Inversion of Resolution (IR).

1R was the first inductive framework to provide a strong logical basis that allows the study of the formal properties of inductive behaviors. Having some elementary processes that can be controlled and combined to perform complex inductions is one step towards a more well founded comparison of learning programs. Moreover, making the link between Machine Learning and Logic Programming allows us to study formal properties of learning systems, such as completeness, soundness, and a concept more particular to ML: what kind of programs can be induced from data and some background knowledge.

IR, however, was subject to many limitations (Rouveirol & Puget, 1990; Rouveirol 1991b). We propose in this paper a framework that addresses some *of* those limitations, Inversion of Logical Entailment (ILE), which is defined as follows. Given:

- a domain theory T
- an example E such that E does not belong to the set of logical consequences of T (denoted T ⊨/= E)

we look in ILE for a domain theory T\ such that *T* logically entails both E and T. In this paper, we will restrict ourselves to the case where T' is identical to T plus a single new clause.

ILE, as opposed to IR which is based on the notion of proof procedure, is a semantic model for induction, i.e., ii is based on the notion of logical models of the manipulated theories: the model of T is considered as a partial model of T, the theory sought.

The paper is organized as follows: we first recall limitations of IR that lead us to introduce ILE. In this framework, we introduce a definition of completeness for inductive procedures and two new elementary operators Truncation and Saturation and we then prove they are complete. Finally, we will consider the control aspects of the method. In particular, we will introduce two controls for Truncation, one bias imposes some limits on the language in which the resulting theory is expressed, and another takes into account semantic properties of concepts the system learns about.
We adopt here the logical notations, as described in (Lloyd, 1987). We are dealing with definite clauses, i.e., clauses with exactly one head literal A clause can be interpreted as a partial definition for a concept, where the head literal identifies the defined concept and the literals in the body of the clause represent the preconditions for an object to belong to this concept. A clause *C*\ is an instance of a concept represented by the clause C iff the body of C 9-subsumes (Plotkin, 1969) the body of Ci , that is, the body of Cj unifies with a subpart of the body of C.

2. Completeness of inductive procedures

2.1, Incompleteness of IR

We are interested in characterizing formal properties of inductive procedures such as ILE. Among the properties that arc usually studied in the logical framework, let us first concentrate on completeness. A very natural semantic definition for completeness of inductive procedures in ILE is the following:

Given a logic program T, a clause E such that T ⊨/= E, and IP, an inductive inference procedure, then IP is complete iff

IP generates all the logic programs T'= IP(T,E) such that $T \not\models T$ and $T \models E$.

IR is not complete with respect to this definition of completeness. It is a well known result in Logic Programming that resolution is refutation complete, that is, given an unsatisfiablc set of clauses, resolution always derives the empty clause. But resolution for first order logic clauses is not complete in the sense of consequent finding when dealing with definite clauses. The following subsumption lemma gives a more precise formulation of this statement:

$$\forall C,\ T \vdash_{SLD} C \Rightarrow \forall \theta\, \exists D,\ T \models D \text{ and } C \supseteq D\theta.$$

In other words, given a clause C belonging to a domain theory T, all the clauses C_s, such that C G-subsumcs C_s are logical consequences of T, although it is not possible to derive those clauses by using resolution. It is therefore not sufficient to invert resolution to have a complete inference procedure. We have shown in (Rouveirol & Puget, 1990) the practical consequences of incompleteness of IR (Truncation in CIGOL is limited to inversion of substitutions and therefore cannot handle the dropping conditions rules). Now that we have a semantic model for induction, we arc no longer restricted to inversion of a proof procedure, we have extended CIGOL's original operators to overcome the above limitations.

3. Saturation

3.1. Incompleteness of generalization

Generalization algorithms such as Elaboration in MARVIN (Sammut & Banerji, 1986), Absorption of CIGOL (Muggleton & Buntine, 1988) and IRES (Rouveirol & Puget, 1989) proceed as follows. Given

- an example clause C_e : $T_e \leftarrow Lc_e$ and
- a clause C_r : $T_r \leftarrow Lc_r$ such that Lc_r θ-subsumes Lc_e ($\exists\, \theta$ such that $Lc_e \supseteq Lc_r\theta$),

the generalization of C_e given C_r is the clause:

C_g: $T_e \leftarrow Rem \wedge T_r\ \theta$,

with $Lc_e = Rem \wedge Lc_r\theta$.

In other words, the generalization amounts to replacing by the head of C_r, substituted by 0, the literals 0-subsumed by the body of C_r in the body of C_e. Therefore, a problem arises when there exists in a domain theory partially overlapping concepts. In this case, there are several competing generalizations for the literals in the body of the initial clause.

Just to take a very simple proposilional example, let us suppose our domain theory is formed of the two following clauses:

T_1: $A \leftarrow B \wedge C$.

T_2: $F \leftarrow C \wedge D$.

and that our example is:

E_1: $E \leftarrow B \wedge C \wedge D$.

There are two possible generalizations following the above algorithm, either

G_1: $E \leftarrow A \wedge D$.

or

G_2: $E \leftarrow B \wedge F$.

but it is not possible, by using the above algorithm to get the most specific generalization of this clause and to perform both generalizations at the same time. We have presented in (Rouveirol & Puget, 1990) a solution to this generalization problem in the form of a new operator called Saturation that performs all the possible and concurrent Absorptions in parallel. We will now develop a formalization of this Saturation operator, relating it to our ILE framework and then to resolution itself.

3.2. Elementary Saturation

Let us first introduce an intermediate step called Elementary Saturation. Basically, elementary saturation docs not replace the description of an instance of a concept by the concept identifier in the body of the generalized clause, but rather *adds* to the body of the clause the information that an object is an instance of a concept.

Definition: Let there be a clause C_e : $T_e \leftarrow Lc_e$ that we will call the initial or *example clause,* and a clause C_s $T_s \leftarrow Lc_s$, *saturant clause* such that Lc_s 0-subsumes Lc_e ($\exists\, \theta$ such that $Lc_e \supseteq Lc_s\theta$). We call *elementary saturation* of C_e by C_s, the clause:

C_h: $T_e \leftarrow Lc_e \wedge T_s\ \theta$

Given the same domain theory and example as defined in section 3.1, one possible elementary saturation is

G_3: $E \leftarrow B \wedge C \wedge A \wedge D$.

and this elementary saturation does not prevent us from performing the other elementary saturation that would lead us to the most specific clause:

G_4: $E \leftarrow B \wedge C \wedge A \wedge D \wedge F$.

Theorem n°1: Elementary Saturation is a *sound* process.

It is indeed interesting to notice that Elementary Saturation which has been recognized as an inverse resolution step (Muggleton, 1990) can be expressed in terms of sound processes, skolcmization and resolution. Skolcmizing a clause means that all the universally quantified variables are replaced by constants that do not appear elsewhere (in the domain theory). Skolemization of the example is logically sound due to the following theorem, proved in (Rouveirol, 1991a).

Theorem n°2 (soundness of skolemization): Let T be a definite logic program and L the language of T . Let $F(X_1, \ldots x_n)$ be a n-ary formula of L, where $X_1, \ldots X_n$ are the variables of F.

$$T \models \forall X_1, \ldots X_n\ F(X_1, \ldots X_n)$$
$$\text{if and iff} \qquad T \models F(c_1, \ldots c_n)$$

where $c_1, \ldots c_n$, are constants that do not appear in T.

3.3. Saturation

When there is no information about how control the use of elementary saturation, we have chosen to be exhaustive. Elementary Saturation with this exhaustive control is refcred to as Saturation.

Definition: *Saturation* operator.
Given a definite program T and a definite clause C_e, the saturation of C_e given T is the transitive closure of the elementary saturation operator. More precisely:

- $Cs_0 = C_e$: $T_e \leftarrow Lc_e$
- $Cs_i = Cs_{i-1} \vee (\bigvee_{j=0}^{nj} \neg(Ts_{ji}\theta_{ji}))$

where the Ts_{ji} are the heads of all the clauses Cs_{ji}: $Ts_{ji} \leftarrow Lcs_{ji}$, such that $Lc_{i-1} \supseteq Lcs_{ji}\theta_{ji}$.
etc[1]....

Definition *(Inversion completeness):* An inference procedure IP is *inversion complete,* given a definite logic program P and a definite clause C_e: T_e <— Lc_e, such that P |=/=C_e, iff IP produces all the clauses C_s : T_s <— Lc_s such that:

$$T_e = T_s \text{ and } T, C_s \models C_e.$$

Theorem n°2 : Saturation is inversion complete.

We will not produce here the proof of inversion completeness of Saturation that can be found in (Rouveirol, 1991a) but we rather give the Saturation algorithm which follows the structure of the proof. This algorithm exploits the relationship between elementary saturation and resolution: inverting logical entailment amounts to performing exhaustive deduction on the skolemized negation of the body of the example clause.

Given the clause C_e: $T_e \leftarrow Lc_e$ and a domain theory T, proceed to steps 1, 2 and 3:

1. Skolemization of C_e in $C_e\theta_S$ (we keeps track of θ_S for deskolemization). For each literal Lc_ei in the body of C_e, a clause C_Ski is created, with C_ski: $Lc_{ei}\theta_S \leftarrow$.
2. Deductive phase: we apply all possible resolutions between the set of clauses {Cski} and the clauses of T.
3. If atoms have been deduced after step 2, they are transformed into unit clauses and added to {Csk_i}. The process then iterates on step 2. If no literals have been deduced after step 2, the (Cski) are deskolemizcd and added to Lc_e to form the body of the saturated clause, C_s.

Figl: Algorithm of Saturation

The result of Saturation is a clause that subsumes all the possible generalizations of the initial clause. The set of all the possible generalizations of the initial clause contains all the clauses that are obtainable from the initial clause by any sequence of inversion of resolution steps.

Saturation can be seen as a formalization of the use of deduction while generalizing, which is an idea that was presented in (Kodratoff & Ganascia, 1986) although not in such a systematic way. Similar steps to Saturation arc used in various systems such as MSG (Buntine, 1987), KBG (Bisson, 1990), CLINT (De Raedt & Bruynooghc, 1989).

4. Truncation

4.1. Definition

Truncation is a purely inductive operator, that is, it generalizes clauses without using background knowledge. According to (Michalski, 1983), there are two main purely inductive generalization rules, the dropping conditions rule and the turning term into variable rule. Truncation can produce, given an input clause C the set of all the clauses that are 0-subsumed by C. It is an extension of the Truncation operator of CIGOL, because it can model both the turning term into variable and the dropping conditions rules, whereas Truncation in CIGOL only takes into account the former. Given an initial clause E, we consider that one Truncation step cither drops one literal in the body of E, or replaces all occurrences of a given term by the same new variable in E or else splits different occurrences of the same term into different variables.

4.2. Coupling Saturation and Truncation

Coupling Saturation and Truncation amounts to performing generalization in the presence of background knowledge. It can, depending on the chosen control for Truncation, reproduce the behavior of generalization algorithms such as Structural Matching (Kodratoff & Ganascia, 1986), and GOLEM (Muggleton & Feng, 1990).

Coupling Saturation and Truncation allows us to achieve inversion of logical entailment, but with a restriction to the completeness defined in section 2.2. Given a clause E and a domain theory T, Saturation provides the clause E_s. Then, Truncation can build from E_s all the clauses E_{stj} such that E_{sli} 0-subsumes E_s, that means, all the possible generalizations of E *which have the same head as E.* Therefore, by adding to the initial theory T any of the E_{sli}, we get all the logic programs which differ from the initial clause by a clause which has the same head as the unrecognized example. To remove this restriction, that is, to induce E_{sti} clauses that do not necessarily have the same head as the example clause, we would have to apply some backward chaining mechanism on the head of the clause. In this way, we would achieve real completeness, but controlling the combination of an descending method such as (Wirth, 1989) and our ascending method is still a research problem.

5. Implementation: the ITOU system

ITOU takes as input a domain theory and the example, it starts by performing a *subsumption test:* docs the theory logically entail the example? The subsumption test is based on the theorem n°l. This subsumption test is very similar to Saturation: the body of the example is skolemized, resolution is then performed, if the skolemized head of the example is derived, the example is subsumed by the theory, otherwise not. This subsumption test and Saturation suffer from the same limitations: if the theory contains loops, the deduction graph grows exponentially. In this case, the depth of the deduction graph is bounded, and once this bound is reached, the example is by default considered unexplained.

If the example is not subsumed by the theory, ITOU first saturated the example (indeed, it uses the deduction graph of the subsumption test). The system then tries to generate a new theory T' such that T subsumes both the initial theory and the unrecognized example. There are of course plenty of theories that satisfy these two conditions. The simplest one is the initial theory plus the unrecognized example only. However, it is clear that we are looking for greater predictiveness for the new theory than this allows. Therefore, ITOU integrates the example into the domain theory, by expressing it in terms of concepts already defined in the domain theory (it is the role of Saturation) and then it will generalize it, dropping unnecessary information, by applying Truncation. The coupling Saturation-Truncation, as explained in section 4, allows to perform generalization in the presence of the domain theory.

[1] We notice that Saturation looks very much like the operator used to build the Least Herbrand model (Muggleton, 1990).

Saturation is deterministic (it produces only one clause whose body contains the bodies of all possible generalizations of an initial clause, given a domain theory), the only control point is within Truncation. Strictly, Truncation generalizes one single clause (the saturated failure example), but to restrict the possible choices for applying Truncation, it has been implemented as a n clauses generalization algorithm: the saturated failure example is generalized against some similar clauses already existing in the incomplete domain theory. Truncation up to now is based on a constrained least general generalization algorithm described in the next section. Each computed generalization is proposed to the user for validation. Each validated generalization is added to the domain theory.

6. Control for induction in ITOU

The Truncation has to be carefully controlled for two reasons. First of all, it is the only inductive step of the method, so information loss has to be under control. The second reason is more practical: the space of reachable clauses from one given clause is very large, if we consider only the general formulation for Truncation.

To reduce the search space of Truncation, the operator takes as input *a set of definite clauses:* only the literals that are not common to the n clauses after saturation are dropped. So the first control point in ITOU is in the choice of clauses (that may already belong to the domain theory or that may be other unrecognized examples) that will be generalized with the saturated example. In our current implementation, examples are represented as definite clauses, so by default, clauses that have the same head predicate will be generalized together. Extensions to more sophisticated clustering techniques that allows for learning of disjunctive concepts have to be studied.

As Truncation is a generalization algorithm that does not use background knowledge, the basis of the algorithm is the least general generalization algorithm (denoted lgg) (Plotkm, 1971). Basically, the lgg algorithm uses the notion of selection. A selection is a n-uple of literals having the same predicate symbol, each member of the n-uple belonging to one of the n clauses to generalize). The arguments of each member of the selection of each selection are recursively scanned in parallel, each occurrence of different terms occurring at the same place in members of the selections is replaced by a variable in the generalization of the selection. All the possible selections are considered and same terms or subterms within the same clauses are replaced by the same variables to ensure that the obtained generalization is the least general one.

We have modified the basic lgg algorithm because, sometimes, it may well be the case that the least general generalization is not the most suited to the learning situation. We have therefore introduced some constraints at different points in the algorithm. The constrained lgg algorithm first orders all the possible selections to consider. Moreover, each time the generalization of a selection is added to the current generalization, the system tests whether the new current generalization satisfies a set of constraints defined on the result of the generalization. If it is the case, the algorithm succeeds and return the new current generalization. If it is not the case, the algorithm considers further selections. If no constraints are given to the algorithm, it returns at the end the lgg of the n input clauses.

There have been a lot of biases for generalization, mostly syntactic, such as constraints on the form of generalization. Just to mention some of them, there are simplicity criteria evaluated with regard to the number of variables or to the number of literals, connectivity or utility criteria, that maximize the number of times where the same object (variable or term) appears in the generalization, the generalizations can be evaluated in terms of their position in the generalization lattice (least or most general generalization) or any domain dependent characteristic. We refer to (Kodratoff, 1988) for a survey of generalization techniques and biases. Sometimes, counter examples can also be used to restrict the space of possible generalizations.

We present here two biases, one called connexion, that restricts the language in which the generalizations are expressed. This bias has an interesting semantic property, and moreover it addresses some well known concerns in knowledge representation. The second one, that we call functional relation bias exploits some semantic properties of concepts we want to learn about.

6.1. Connexion

The connexion principle states that variables in the body of a clause must either appear in the head of the clause, or be linked to a variable of the head of the clause through a path of predicates. This ensures that all literals in the body of a clause are related to its head, or that all the conditions for an instance to belong to a concept are related to the concept itself. (Morik, 1989) lists some common errors in knowledge modeling that justify the use of connexion.

The input of the generalization algorithm is restricted to connex clauses, and the generalization of two connex clause has to be connex. The corresponding constraint in our generalization algorithm is that there must be at least one variable in the generalization of a new selection that appears in the current generalization. This allows us to limit the number of variables in the generalization. Similar constraints have been used in CLINT (De Raedt & Bruynooghe, 1989). Moreover, we have proved that a non connex formula has the same least Herbrand model as the corresponding connex clause (where all the unconnected literals have been dropped).

6.2. Functional relation

The functional relation principle is a semantic bias. For some predicates, we know that instantiation of some of its arguments uniquely sets instantiations for its other arguments. For example, an object has only one position, people have only one social security number, one father and one mother, etc... We propose here a definition of functional relation close to the one proposed in (Puget, 1989).

Definition: Let P be a logic program, let pred (X, Y) be a predicate where X and Y are two vectors of variables defined in P. Pred (X, Y) is *afunctional relation* of X if pred (X, Y) verifies :

$$T + \text{equality theory} \models (\forall X,Y,Z, pred(X,Y) \wedge pred(X,Z) \rightarrow Y=Z)$$

The notion of functional relation is very close of that of the functional determination (Russel, 1989). Functional

relations are indeed frequent: for example, in object oriented languages, a mono-valued slot defines a functional relation.

(Puget, 1989) showed how such an information can be used to remove negated literals in the body of a clause describing the negation of pred. Let us see here how it can be used in drive the generalization process. Let us suppose we are generalizing two clauses:

C_1: pred(X,Y) ← q_1(X,Y,Z)

C_2: pred(X',Y') ← q_2(X',Y',Z')

and that we know that pred is a functional relation of X. The generalization of C1 and C2 is the clause C_g

C_g: pred(A,B) ← q_3(A,B,C)

where q_3(A,B,C) is the generalization of q_1(X,Y,Z) and q_2(X',Y',Z').

q_3 (A,B,C) must be such that pred (A, B) is a functional relation of A. This means that given any instantiation of A, there must be only one possible instantiation for B whatever the relationships between variables of C and A and variables of C and B. To be able to compute such a q_3 (A, B, C), we need of course to know about predicates in q_1 (X, Y, z) and q_2 (X', Y', Z') that are themselves functional relations. The most general q_3s contain only predicates which arc functional relations.

Let us see on a small example how knowledge about functional relationship can guide the generalization process, plus (X, Y, z) is a functional relation of any two of its arguments. For example, once X and Y are instantiated, there is only one possible instantiation for Z. We want to generalize two clauses that describe saturated addition examples (see Rouveirol, 1990) for details)':

I2fs: plus(Y,Y,U) ←
zero(ζ) ∧ succ(ζ,X) ∧ succ(X,Y) ∧
succ(Y,Z) ∧ succ(Z,U) ∧ **plus(Y,X,Z)**.

which is the flattened saturated representation of 2 plus 2 equals 4. The literal in bold has been added by saturating the clause representing 2 plus 2 equals 4 with the clause representing 2 plus 1 equals 3.

I3fs : plus(Y',Z',V') ←
zero(ζ') ∧ succ(ζ',X') ∧ succ(X',Y') ∧
succ(Y',Z') ∧ succ(Z',U') ∧
plus(Y',X',Z') ∧ plus(Y',Y',U').

which is the saturated flattened representation of 2 plus 3 equals 5, obtained by saturation of the initial clause representing 2 plus 3 equals 5 with the clause representing 2 plus 1 equals 3 and I2fs.

A candidate generalization is

plus(X,U,V) <-
plus(X,Y,Z) A succ(Y,U).

Let us check whether this generalization verifies the functional relation of addition. Let us take any of the three functional relation constraints given for addition, for example

given plus(X,Y,Z), if Y and Z are instantiated, there is only one possible instantiation for X.

We also need the knowledge that succ(X,Y) is a functional relation for any one of its arguments.

plus(X,Y,Z) means that the result of X plus Y equals Z, and succ(X,Y) means that Y is the arithmetic successor of X.

Checking this functional relationship on the candidate generalization amounts to instantiating U and V. We then have to check how these instantiations propagates through the body of the clause. There is no instantiated variable in plus (X, Y, z), so no other variables become instantiated. For succ (Y, U), U is instantiated; as succ is a functional relation of any of its arguments, Y becomes instantiated. But this is not sufficient for X and Z to become instantiated, as only on variable in plus (X, Y, Z) is instantiated. X is not instantiated after propagations of all the instantiations, and therefore the functional relation is not fulfilled by our candidate generalization.

Another candidate generalization is:

plus(X,U,V) ← plus(X,Y,Z) ∧
succ(Y,U) ∧ succ(Z,V) .

Checking the same functional relation constraint, we start as in the previous example, but for succ (z, v) , V is instantiated and thus so becomes Z. Therefore, Y and Z are instantiated arguments in plus(X,Y,Z), and then X becomes instantiated. The functional relation is verified, this generalization is thus proposed to the user. This is one of the recursive definitions for addition.

Just to give an idea about the power of these two biases, on this example for synthesizing addition from examples (Rouveirol, 1990), our basic generalization algorithm computed all the subparts of the least general generalization (Plotkin, 1969) of two clauses. There were approximately 1100 candidate generalizations without using any bias, 70 candidate generalizations after using the connexion bias, and only six of them were left after also applying the functional relation bias. The most general of these six was the recursive definition of addition.

7. Related works

We do not present here CIGOL, but a detailed comparison of our work and that of (Muggleton & Buntine, 1988) can be found in (Rouveirol, 1991a). GOLEM, its successor has some strong similarities with our. Our Saturation operator is similar to the v^n operator introduced in (Muggleton, 1990) but it is not restricted to handle strongly generative clauses, that means clauses where any variable appears at least twice, although wc can introduce afterwards this constraint to restrict our search for a good generalization in Truncation. To restrict the complexity of the task, GOLEM is restricted beforehand to theories formed of strongly generative clauses. Under this assumption, the background knowledge can be replaced by a finite set of ground atoms, i.e., a propositional case.

The v^n operator maintains a set of clauses that are obtainable by applying at most n most specific inverse resolution steps (a most specific inverse resolution step is similar to our elementary saturation). The difference is that our Saturation operator applies elementary saturation exhaustively: therefore it provides only one clause as a result that implicitly contains all the clauses maintained by the v^n operator. It is by applying Truncation that ITOU could build all the clauses built by v^n, but it will never, in practice, build them all. A completeness result for the v^n operator was reported for GOLEM, restricted to strongly generative clauses. Independently from (Rouveirol & Puget, 1990), (Muggleton, 1990) made a theoretical link between relative least general generalization (rlgg) that is, least general generalization in the presence of background

knowledge, and inverse resolution with the v^n operator. There is a strong similarity in both approaches: v^n is similar to Saturation, whereas GOLEM is similar to our generalization algorithm fro Truncation. But in ITOU, Saturation is used to complete the example description before performing Truncation, whereas it does not seem that v^n is used as part of GOLEM.

Another difference is that our approach relies on a small number of well chosen examples, whereas GOLEM builds efficient relative least general generalizations of a mass of examples. At last, to simplify the obtained rlgg, GOLEM uses biases to guide a dropping rule mechanism in a similar way. The MSG algorithm (Buntine, 1987) is also similar but addressed a different generality relation, generalized subsumption, whereas we deal with logical entailment with definite clauses.

8. Conclusion

We have proposed a new semantic model for induction of first order logic logic programs. In this framework, we have defined a notion of completeness and defined two operators that achieve a restricted completeness. These two operators are left deliberately general, so that they can be guided through an external declarative control that allow the system to converge faster towards the desired theory(ies).

There are many further directions for future research. The main one is the development of other operators within our framework. A specialization operator (up to now, we have just proposed operators that allow us to *increase* the number of logical consequents of logic programs), and therefore, we still cannot handle incorrect logic programs. Works of (Shapiro, 1983) (Wrobel, 1988) and (Bain, 1991) should be investigated. Introduction of new terms needs to be developed , in particular the notion of completeness has to be adapted to include the introduction of recursive predicates to allow to keep the size of the resulting theory finite. In parallel to introduction of new operators, declarative means of control should be developed. An interesting work in this respect is (Kirshenbaum & Sterling, 1991).

Acknowledgements: I would like to thank Yves Kodratoff, who was my thesis supervisor for the support and suggestions he has provided to this work, M. Bruynooghe and J.G. Ganascia, and the anonymous referees for their comments. I thank also J.F. Puget who has started with me this work on inverse of resolution, for his support.

References

Bain M. : "Experiments in Non Monotonic First ORder Induction", in *Proc. of the first Inductive Learning Programming Workshop, Muggleton (Ed),* 1991.

Bisson G.: "KBG: A Knowledge Base Generalizer", dans Proc. of the 7th Internat. Conf. on Machine Learning, B. Porter & R. Mooney (Eds), Morgan Kaufman, pp 9-16, 1990.

De Raedt, L. & Bruynooghe M: "Towards friendly Concept Learners", in Proc. of the 11th International Joint Conference on AI, pp 849-854, 1989.

Kirschembaum M & Sterling L.S.: "Refinement Strategies for Inductive Learning of Simple Prolog Programs", in Proc. of IJCAI91.

Kodratoff Y. & Ganascia J.G.G. : "Improving the generalization step in Learning", in *Machine Learning, An Artificial Intelligence Approach,* volume II, pp 215-244, Morgan Kaufman, 1986.

Kodratoff Y. : *Introduction to Machine Learning,* Pitman, 1988.

Lloyd J.W.: *Foundations of Logic Programming,* second extended edition, Springer Verlag, 1987.

Michalski, R. S. : "A Theory and Methodology of Inductive Learning", in *Machine Learning: An Artificial Intelligence Approach,* pp. 83-134, Michalski R.S., Carbonell J.G., Mitchell T.M. (eds),Tioga Publication Company, 1983.

Morik K. : "Sloppy Modelling", in Knowledge Representation and Organization in Machine Learning, K. Morik (ed), Lecture Notes in Artificial Intelligence 347, Springer Verlag, 1989.

Muggleton S. & Buntine W.: "Machine invention of first order predicates by inverting resolution", in *Proc. of fifth international Machine Learning Workshop,* pp 339-352, Morgan Kaufman, 1988.

Muggleton, S. : "Inductive Logic Programming", dans *Proc. of the first Conference on Algorithmic Theory,* Ohmsha Pub, Tokyo, 1990a.

Muggleton S., Feng C. : "Efficient Induction of Logic Programs", in *Proc. of the first Conference on Algorithmic Theory,* Ohmsha Pub, Tokyo, 1990.

Plotkin G. D. : "A note on inductive generalisation", in *Machine Intelligence 5,* B. Meltzer and D. Michie Eds, pp 153-163, Edimburgh University Press, 1969.

Puget J. F. : "Learning Concept Negation", in *Proc. of the fourth European Working Session on Learning,* ppl79-189, Pitman, 1989.

Sammut C. & Banerji R. B. : "Learning concepts by asking questions", dans *Machine Learning : an Artificial Intelligence Approach,* vol2, pp 167-192, Michalski R.S., Carbonell J.G. and Mitchell R.M. Eds, Morgan Kaufman, 1986

Rouveirol C. & Puget J.F. : "Beyond inversion of resolution", proceedings of the fifth International Conference on Machine Learning", Morgan Kaufman, 1990a.

Rouveirol C. : "Saturation: Potsponing Choices for Inverting Resolution", in *Proc. of the ninth European Conference on Artificial Intelligence,* (Luigia Carlucci Aiello Ed), pp 557-562, Pitman, 1990b.

Rouveirol C: "ITOU: Induction of First Order Theories", in *Proc. of the first Inductive Learning Programming Workshop, Muggleton (Ed),* 1991a.

Rouveirol C: "Completeness for inductive procedures", in *Proc. of the eight ML Workshop,* 1991b.

Russel S. J.: *The use of Knowledge in Analogy and Induction,* Research Notes in Artificial Intelligence, Pitman, 1989.

Shapiro E. Y.: *Algorithmic Program Debugging,* MIT Press, 1982.

Wirth R.: "Completing Logic Programs by Inverse Resolution", *proc. of the fourth European Working Session on Learning,* Pitman, pp 239-z50,1989.

Wrobel S.: "Automatic representation adjustement in an observational discovery system", in Proc. of the third European Working Session on Learning, pp 253-262, Pitman, 1988.