Inductive learning of normal clauses

Christel VRAIN [1], Lionel MARTIN
L.I.F.O.
Rue Léonard de Vinci
B.P. 6759
45067 Orléans Cedex 2
France
email: cv@univ-orleans.fr martin@univ-orleans.fr

submitted to ECML94

Summary: In this paper, we are interested in the induction of normal clauses. Several semantics have been proposed for normal programs, we consider here the well-founded semantics, based on a three-valued logics. We have generalized to this semantic the classical constraint: all the positive examples must be covered by the learned program and all the negative ones must be rejected. In the case of inductive learning, this constraint seems too strong and we have defined a weaker criteria: we require that a positive (resp. negative) example is not considered as $False$ (resp. $True$) by the learned program. This study has been applied to the system FOIL. Some positive examples that were covered by the program during the learning process were not in the well-founded semantics. and we define biases to solve this problem.

Keywords: Inductive logic programming, well-founded semantics, biases.

[1] corresponding member of the Inference and Learning group, L.R.I., university of Paris South

1 Introduction

In this paper we are interested in the induction of normal programs. Most systems, as for instance the system GOLEM [5], based on the R.L.G.G. (relative least general generalization) [8, 9] learns only definite clauses. The system Clint mostly deals with definite clauses but it has been extended to learn normal stratified programs [12]. To our knowledge, only the systems FOIL and FOCL [11, 7] can learn any normal programs but the method is purely syntactic and they do not really address the problem of the semantics of the learned program and of its correction relative to the initial specifications.

Learning normal clauses is much more complex first of all because of the semantics of normal programs. In the case of a definite program $\mathcal{P}$, everyone agrees that its semantics is the set of the Herbrand logical consequences of $\mathcal{P}$, defined also as the least fixpoint of the classical operator $T_{\mathcal{P}}$. Several semantics have been proposed for normal programs. Some are operational as for instance the SLDNF resolution [4], others are more theoretical, as the stable model semantics [3]. We give in the next section a brief overview of these semantics.

We have chosen to study this problem on the system FOIL for the following reasons:

- The underlying algorithm is very simple. The idea is to build a clause that covers many positive examples and rejects all the negative ones and to repeat the process until all the positive examples are covered. When a new literal is added to the body of a clause, we have to compute the new assignments of variables that satisfy the body of the clause. This operation is called "computation of new tuples".
- The domain theory is expressed by ground unit clauses.
- The predicate to learn is also defined by extension. Its definition can be complete or partial. This point is interesting since it enables to study two kinds of learning: reformulation of knowledge and induction of knowledge. In the first case, we have to rewrite knowledge initially described by ground facts into non-unit clauses. The interest of such an operation is the compression of information. It is then essential that the learned programm be correct, i.e. covers all the positive examples and rejects all the negative ones. In the second case, we have to induce a new program that can give a value $True$ or $False$ to unknown information.

It is important to notice that in this paper we do not consider the information-based heuristic that guides the search in FOIL. We are interested in the comparison between the semantics of a program that could have been built by FOIL because it covers (in the sense of FOIL and it will be explained in section 3.1) the positive examples and rejects the negative ones and its expected specifications.

The first point of our study is that FOIL uses an operational semantics of negation that has few interesting theoretical properties. We have used the well-founded semantics - and we explain why in the next section - and therefore we have had to change the way the new tuples were built.

We have defined some conditions that the target program must satisfy. In the following, we call E_Q^+ and E_Q^- respectively the sets of positive and negative examples of the target predicate Q and $\mathcal{P}$ the learned program, composed of the initial theory and of the new clauses that define the predicate Q. The semantics $\mathcal{M}_\mathcal{P}$ of the program is divided in three sets:

- $\mathcal{M}_\mathcal{P}^+$, the set of ground atoms $True$ for $\mathcal{P}$,
- $\mathcal{M}_\mathcal{P}^-$, the set of ground atoms $False$ for $\mathcal{P}$,
- $\mathcal{M}_\mathcal{P}^u$, the set of ground atoms $Undefined$ for $\mathcal{P}$ [2].

The "acceptability condition" that seems the most natural is:

$$E_Q^+ \subseteq \mathcal{M}_\mathcal{P}^+ \text{ and } E_Q^- \subseteq \mathcal{M}_\mathcal{P}^-$$

it means that the positive examples are $True$ in $\mathcal{P}$ and the negative ones are $False$ in $\mathcal{P}$.

In the first kind of learning, we have shown that it is possible to define biases so that the program $\mathcal{P}$ satisfies this condition. On the other hand, in the case of inductive learning, this constraint seems too strong and we have definied a new criterion:

$$E_Q^+ \cap \mathcal{M}_\mathcal{P}^- = \emptyset \text{ and } E_Q^- \cap \mathcal{M}_\mathcal{P}^+ = \emptyset$$

It means that we require that a positive (resp. negative) example is not considered as $False$ (resp. $True$) by the learned program, but we allow that the program consider some examples as $Undefined$.

Finally, in the case of inductive learning, it seems that FOIL treats incorrectly unknown information. It leads to programs that satisfy none of the above conditions. We have modified the step called "computation of new tuples", in order to obtain acceptable programs.

2 Semantics of normal clauses

The semantics of normal programs can be studied according several point of view: a declarative semantics that is independant of the way it will be implemented, a theoretical, operational semantics that defines how the semantics should be implemented and the current semantics that corresponds to the actual implementation.

[2] In the case of definite programs, we always have $\mathcal{M}_\mathcal{P}^u = \emptyset$

In the field of Inductive Logic Programming, we usually consider the two last points of view, because we think that we must build an operational program that can run on a Prolog interpreter. But, it is then difficult to study the properties of the learning system, and it explains why we have prefered a declarative semantics.

In Logic Programming, we can distinguish two approaches:

- the first one consists in transforming a program $\mathcal{P}$ into a set of formulas $\mathcal{A}(\mathcal{P})$ and to study the logical consequences of $\mathcal{A}(\mathcal{P})$. It is illustrated by Clark's completion semantics [4] and Fitting's 3-valued extension [2]; its theoretical operational semantics is computed by SLD-resolution.

- the second one consists in determining for a program $\mathcal{P}$ a canonical model $\mathcal{M}_{\mathcal{P}}$. It is illustrated by the stable model semantics [3], that has been extended to the three-valued stable semantics [10]. It then coincides with the well-founded semantics [14], that is computed by SLS-resolution.

In Learning, one tends to consider the SLD-resolution semantics and therefore Clark's completion semantics. Few considers a three-valued semantics, although it is very useful to account for undefined answers. The main difference between Fitting's approach and the well-founded one can be intutitively stated as follows: in the SLDNF approach, a ground atom is false if there is a finitely failed SLDNF tree and in the well-founded semantics, it is false if there is a failed research tree which can be infinite. Therefore we have the following relations:

$$\mathcal{M}_{\mathcal{P}}^{Fit} \subseteq \mathcal{M}_{\mathcal{P}}^{WF}$$

where $\mathcal{M}_{\mathcal{P}}^{Fit}$ denotes Fitting's semantics and $\mathcal{M}_{\mathcal{P}}^{WF}$ denotes its well-founded semantics. We have chosen the well-founded semantics, because it maximizes the sets of $True$ and $False$ atoms.

3 Overview of the system FOIL

In this section, we present briefly the system FOIL. We do not describe all its characteristics; we are mostly interested in the formalization of the notions underlying this system, as for instance the definition of an example of a clause or the notion of coverage.

The aim of FOIL is to learn an intensional definition of a predicate (or relation) Q from positive and negative examples of it.

As inputs, we give a finite set $\mathcal{D}$ and some predicates R_i, $1 \leq \mathrm{i} \leq \mathrm{j}$, defined in extension on the domain $\mathcal{D}$.

The extension of a k-ary predicate R_i is specified by two lists, written $E_{R_i}^+$ and $E_{R_i}^-$, of k-tuples. The list $E_{R_i}^+$ (resp. $E_{R_i}^-$) is composed of k-tuples, labelled

+ (resp. -) $(t_1, ..., t_k)$ such that $R_i(t_1, ..., t_k)$ is considered as true (resp. false) in the expected interpretation.

If $E^+_{R_i} \cup E^-_{R_i} = \mathcal{D}^k$, we say that the definition of R_i is total or that R_i is completely defined in extension.

The predicate to learn Q is defined in the same way in extension. As usual in learning, we also call the elements of E^+_Q (resp. E^-_Q) the positive (resp. negative) examples of Q. The sets E^+_Q and E^-_Q define the expected interpretation, also called the specification, of the predicate Q.

The learning system search for a set of non-unit clauses $\{\mathcal{C}_i\}$ that covers all the positive examples and rejects all the negative ones. The clauses $\mathcal{C}_i$: $L \leftarrow L_1, ..., L_p$ must satisfy the following conditions:
- L is an atom of predicate Q: $L = Q(X_1, \ldots, X_n)$,
- L_i is
  - either a literal (i.e. an atom or the negation of an atom) of predicate R_j or Q, [3]
  - or a relation $X_r = X_s$ or $X_r \neq X_s$ between two variables X_r and X_s that already occur in L or in a L_j, $1 \leq$j$\leq$ i-1.

Notations: In the following we consider a first order language composed of the basic predicates R_i, the predicate Q and the symbol of constants that appear in $\mathcal{D}$. The extensional definitions of R_i and Q are seen as the intended Herbrand interpretation. We use the following notations:
$E^+_R = \cup_1^j E^+_{R_i}$ and $E^+ = E^+_Q \cup E^+_R$
$E^-_R = \cup_1^j E^-_{R_i}$ and $E^- = E^-_Q \cup E^-_R$
$E_R = E^+_R \cup E^-_R$ and $E_Q = E^+_Q \cup E^-_Q$

3.1 Definition of an example

- for a predicate:
  - A positive example for a n-ary predicate $Q(X_1, ..., X_n)$ is a n-tuple $(t_1, ..., t_n)$ such that $(t_1, \ldots, t_n) \in E^+_Q$.
  - A negative example for a n-ary predicate $Q(X_1, ..., X_n)$ is a n-tuple $(t_1, ..., t_n)$ such that $(t_1, \ldots, t_n) \in E^-_Q$.

  When the extensional definition of the target predicate Q or of the basic predicates R_i are incomplete we can use sets of tuples different from E^+ and E^- in order to compute a program the semantics of which corresponds to the intended interpretation. Therefore we write C^+ (resp. C^-) the set of tuples that will be considered as true (resp. as false) during the

[3] in the case of FOIL, in the literal L_i, at least a variable occurring in L or in a literal L_j, j < i, must appear.

computation. When the extensional definitions of Q and R_i are total, we have $C^+ = E^+$ and $C^- = E^-$.

- for a clause:

  Let $\mathcal{C}(X_1, \ldots, X_k)$: $L \leftarrow L_1, \ldots, L_p$, a clause in which appear the variables X_1, ..., X_k.

  - A **positive example for the clause** $\mathcal{C}(X_1, \ldots, X_k)$ is an assignment α: $X_1, \ldots, X_k \rightarrow \mathcal{D}$ satisfying the following conditions:
    - $\alpha.L \in C^+$
    - for all i, $1 \leq$ i $\leq$ p,
      - if L_i is a positive literal, $\alpha.L_i \in C^+$
      - if L_i is a negative literal, $L_i = \neg L_i^{'}$, then $\alpha.L_i^{'} \in C^-$
  - A **negative example for the clause** $\mathcal{C}(X_1, \ldots, X_k)$ is an assignment α: $X_1, \ldots, X_k \rightarrow \mathcal{D}$ satisfying the following conditions:
    - $\alpha.L \in C^-$
    - for all i, $1 \leq$ i $\leq$ n,
      - if L_i is a positive literal, $\alpha.L_i \in \overline{C^-}$.
      - if L_i is a negative literal, $L_i = \neg L_i^{'}$ then $\alpha.L_i^{'} \in \overline{C^+}$.

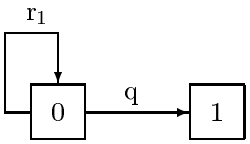
  The assignment α for the variables $X_1, \ldots, X_k$ is represented by the k-tuple $(\alpha(X_1), \ldots, \alpha(X_k))$

- Coverage

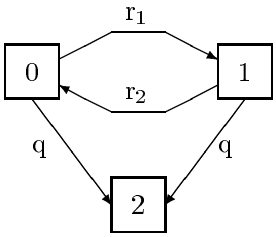
  A **n-tuple** $(c_1, \ldots, c_n)$ is **covered** by a program $\mathcal{P}$ relatively to C^+ and C^- if there exists an instantiated clause of $\mathcal{P}$: L $\leftarrow L_1, \ldots, L_k$ with L = $Q(c_1, \ldots, c_n)$ and for all i, $1 \leq$ i $\leq$ k,

  - if L_i is a positive literal, then $\alpha.L_i \in C^+$
  - if L_i is a negative literal, $L_i = \neg L_i^{'}$, then $\alpha.L_i^{'} \in C^-$

The system FOIL constructs step by step clauses that cover all the positive examples and reject all the negative ones. The important point here is that there may exist examples that are covered by a program $\mathcal{P}$ but that do not belong to the actual semantics of $\mathcal{P}$. Let us for instance consider the two following cases:



Example 1



Example 2

$E_q^+ = \{(0, 1)\}$ $\qquad\qquad$ $E_q^+ = \{(0, 2)\ ,\ (1, 2)\}$

The following programs cover all the positive examples respectively for the first and the second example

q(X,Y) $\leftarrow$ r_1(X, Z), q(Z, Y)
$r_1(0, 0) \leftarrow$.

q(X,Y) $\leftarrow$ r_1(X, Z), q(Z, Y)
q(X,Y) $\leftarrow$ r_2(X, Z), q(Z, Y)
$r_1(0, 1) \leftarrow$.
$r_2(1, 0) \leftarrow$.

These programs are definite and their semantics are respectively given by the sets $\{r_1(0, 0)\}$ and $\{r_1(0, 1)\ ,\ r_2(1, 0)\}$ that do not contain E_q^+.

In these two examples, the set E_q^+ satisfies the property of being unfounded [14] for the program $\mathcal{P}$ (for all element $q(c_1, \ldots, c_n)$ of E_q^+, and for all ground instance of a clause of $\mathcal{P}$, $q(c_1, \ldots, c_n) \leftarrow L_1, \ldots, L_n$, there exists a positive literal L_i belonging to E_q^+). It is then easy to find a model in which all the elements of E_q^+ are true or false or undefined.

3.2 Algorithm

The algorithm of FOIL is based both on a "divide-and-conquer" method and on a "covering" method. To produce the set of clauses $\{\mathcal{C}_j\}$ of $\mathcal{P}$, two loops are needed: an outer loop that constructs clauses one by one (this loop is left when all the positive examples are covered), and an inner one that constructs a clause A $\leftarrow L_1, \ldots, L_n$, by adding literals L_j until all the negative examples are rejected.

Let POS be the subset of E^+, containing the positive examples that are not covered by the clauses previously produced (initially POS = E_Q^+); let T_i^+ (resp. T_i^-) be the set of positive (resp. negative) examples of the clause A $\leftarrow L_1, \ldots,$ L_i under construction; the algorithm can be summarized as follows:

POS = E_Q^+
While POS $\neq \emptyset$, repeat *Create a new clause*
  T_0^+ = POS , T_0^- = E_Q^-
  While $T_i^- \neq \emptyset$, repeat *Add a new literal*
    Find a literal L_i
    Compute T_{i+1}^+ and T_{i+1}^-,
    i = i+1
    EW
  POS = POS - T_i^+ / { $X_1, \ldots X_n$ }
  EW

For more details see [11]

3.3 Computation of T_i^+ and T_i^-

When entering into the inner loop, the clause is $Q(X_1, \ldots, X_n) \leftarrow .$; the positive examples of this clause are the elements of E_Q^+ which are not covered by the previous clauses, i.e. $T_0^+ = \{$assignment $\alpha \mid \alpha.Q(X_1, \ldots, X_n) = Q(c_1, \ldots, c_n)$ and $(c_1, \ldots, c_n) \in$ POS $\}$. We note $T_0^+ =$ POS. In the same way, we note $T_0^- = E_Q^-$.

We give a generic way to compute T_i from T_{i-1}, depending on a set C, and we will discuss in the following two choices of C so that the learned program $\mathcal{P}$ satisfies the condition given in section 1.

- if L_i is a positive literal $P(X_{i_1}, \ldots, X_{i_r})$:
  - $T_i^+ = \{$ assignment α' $\mid$ α' coincides with an assignment $\alpha \in T_{i-1}^+$ on the variables $X_1, \ldots, X_m$, and such that $\alpha'.P(X_{i_1}, \ldots, X_{i_r}) \in C^+ \}$
  - $T_i^- = \{$ assignment $\alpha' \mid \alpha'$ coincides with an assignment $\alpha \in T_{i-1}^-$ on the variables $X_1, \ldots, X_m$, and such that $\alpha'.P(X_{i_1}, \ldots, X_{i_r}) \in \overline{C^-} \}$
- if L_i is a negative literal $\neg P(X_{i_1}, \ldots, X_{i_r})$:
  - $T_i^+ = \{$ assignment $\alpha' \mid \alpha'$ coincides with an assignment $\alpha \in T_{i-1}^+$ on the variables $X_1, \ldots, X_m$, and such that $\alpha'.P(X_{i_1}, \ldots, X_{i_r}) \in C^- \}$
  - $T_i^- = \{$ assignment $\alpha' \mid \alpha'$ coincides with an assignment $\alpha \in T_{i-1}^-$ on the variables $X_1, \ldots, X_m$, and such that $\alpha'.P(X_{i_1}, \ldots, X_{i_r}) \in \overline{C^+} \}$

The computation of T_i^- and T_i^+ is different in FOIL. In the case of a positive literal, C^+ and $\overline{C^-}$ are replaced by E^+ and in the case of a negative literal, C^- and $\overline{C^+}$ are replaced by E^- .

Moreover, when a negative literal $\neg P(X_{i_1}, \ldots, X_{i_r})$ is added: the set T_i^+ (resp. T_i^-) is the subset of T_{i-1}^+ (resp. T_{i-1}^-) containing the assignment α such that for each α' that coincides with α and making , $\alpha' P(X_{i_1}, \ldots, X_{i_r})$ ground, $\alpha' P(X_{i_1}, \ldots, X_{i_r}) \in C^-$; the underlying "semantics" is the semantics induced by the (operational) *negation by failure* rule without detection of floundering, which has few interesting theoretical properties.

Notations: In the following the program $\mathcal{P}$ is composed of the unit clauses that define the elements of $E_{R_i}^+$ for all the relations R_i and the learned clauses for the predicate Q.

4 Complete definition of Q and R_i

4.1 Well founded ordering

In FOIL [11], a bias has been defined to prevent the system from creating programs such

$$Q(X_1, \ldots, X_n) \leftarrow Q(X_1, \ldots, X_n).$$

which satisfies the constraints of coverage (all the positive examples are covered and no negative one is covered).

We have formalized it in [15] and we say that the learned program $\mathcal{P}$ verifies the **stratification bias** if we can find an index k such that for each ground instance of a clause of $\mathcal{P}$: $Q(c_1, \ldots, c_n) \leftarrow L_1, \ldots, L_m$ such that:

- for each positive literal $L_i = \mathrm{p}(c_1', \ldots, c_m')$, $(c_1', \ldots, c_m') \in E_p^+$,
- for each negative literal $L_i = \neg\ \mathrm{p}(c_1', \ldots, c_m')$, $(c_1', \ldots, c_m') \in E_p^-$,

then for each literal $L_i = Q(c_1', \ldots, c_n')$ or $L_i = \neg\ Q(c_1', \ldots, c_n')$, $c_k > c_k'$.

This bias depends on well-founded orders on the elements of the domain $\mathcal{D}$. These orders are computed when beginning a session of FOIL.

4.2 Results

When the predicate Q and the initial theory are completely defined in extension, we have the two following results. Their proofs are given in [15].

- if a definite program $\mathcal{P}$ satisfies the stratification bias, then the set E^+ is the least Herbrand model of $\mathcal{P}$.
- if the normal program $\mathcal{P}$ satisfies the stratification bias, then the set $E^+ \cup \neg\ E^-$ is the well founded model of $\mathcal{P}$.

In this case, we have shown that the "natural" model of $\mathcal{P}$ is 2-valued and that $\mathcal{M}_\mathcal{P}^+ = E^+$ and $\mathcal{M}_\mathcal{P}^- = E^-$: the semantics of $\mathcal{P}$ is exactly the intended interpretation and we cannot expect more satisfying conditions; the reformulation of knowledge in FOIL is then correct and complete.

5 Partial definition

We focus now on the case where the predicate Q is partially defined i.e. $E_Q^+ \cup E_Q^- \subset \mathcal{D}^n$ (we note E_Q^u the set of unknown tuples, $E_Q^u = \mathcal{D}^n$ - $(E_Q^+ \cup E_Q^-)$).

We have chosen the (3-valued) well founded semantics to express the "natural" semantics of the learned program: the semantics of $\mathcal{P}$ is defined by two sets of ground atoms: the set $M_\mathcal{P}^+$ of the true ground atoms and the set $M_\mathcal{P}^-$ of the false ground atoms. Generally, $M_\mathcal{P}^+ \cup M_\mathcal{P}^-$ is a subset of the domain and we note $M_\mathcal{P}^u$ the set of undifined ground atoms.

5.1 Definition of an acceptable program

We can require that the semantics of the learned program verifies the following condition:

$$E^+ \subseteq M^+_{\mathcal{P}} \text{ and } E^- \subseteq M^-_{\mathcal{P}}$$

i.e. each positive example is a consequence of $\mathcal{P}$ and each negative example is false for $\mathcal{P}$.

Let us recall that FOIL does not take the unknown information into account. Let us consider the following example, where $\mathcal{D} = \{ 1, 2, 3 \}$.
The predicate to learn q, is partially defined and the predicates r and S and completely defined by:

$E^+_q = \{(1,2), (1,3)\}$, $E^+_r = \{(2,3), (1,2)\}$ and $E^+_s = \{(2,2), (3,1)\}$
$E^-_q = \{ (2,1) \}$ $E^-_r = \mathcal{D}^2$ - E^+_r $E^-_s = \mathcal{D}^2$ - E^+_s

We assume that the first clause learned by FOIL is q(X,Y) $\leftarrow$ s(Y,X).
which covers the positive example (1,3) and rejects the negative example (2,1).
Then FOIL try to find a new clause, in order to cover the positive example (1,2): if we assume that it chooses first the literal r(Y,Z) and then the literal q(X,Z), the sets T^+_i and T^-_i are:

$T^+_1 = (1,2,3)$ and $T^+_2 = (1,2,3)$
$T^-_1 = (2,1,2)$ $T^-_2 = \emptyset$

and the tuple (2,1,2) is rejected, because $(2,2) \notin E^+_q$.

Finally, the learned definition for q is:

q(X,Y) $\leftarrow$ s(Y,X).
q(X,Y) $\leftarrow$ r(Y,Z), q(X,Z).

but this program answers YES for the goal q(2,1), which was given in the set of negative examples.

To avoid the construction of such programs, we have two solutions, consisting in modifying E^+ and E^-, in order to have $E^u = \emptyset$: in the first one, we consider that every unknown tuple is a positive example (the set E^+ is replaced by $E^+ \cup E^u$) and in the second one, every unknown tuple is treated like a negative example (the set E^- is replaced by $E^- \cup E^u$). In this way, we can apply the results of the section "complete interpretation", and we obtain the expected property.

5.2 Definition of a weak acceptability criterion

The previous constraint is too strong, and is implicitly linked to the *Closed World Assumption* and a 2-valued semantics; we define now a weak acceptability criterion, adjusted to a 3-valued semantics:

$$E^+ \subseteq (M^+_{\mathcal{P}} \cup M^u_{\mathcal{P}}) \text{ and } E^- \subseteq (M^-_{\mathcal{P}} \cup M^u_{\mathcal{P}})$$
$$\text{i.e.}$$
$$E^+ \subseteq \overline{M^-_{\mathcal{P}}} \text{ and } M^+_{\mathcal{P}} \subseteq \overline{E^-}.$$

The program is required to satisfy: none of the negative examples can be proved by $\mathcal{P}$, and none of the positive examples can be false for $\mathcal{P}$.

To reach such a condition, we modify the computation of the new tuples, taking $C^+ = E^+$ and $C^- = E^-$ and we use a partial stratification bias, defined in next section.

We show in annex A that, when the program $\mathcal{P}$ is accepted by this bias, then it verifies the weak acceptability criterion.

6 Partial stratification

To achieve the constraint ($E^+ \subseteq \overline{M^-_{\mathcal{P}}}$ and $M^+_{\mathcal{P}} \subseteq \overline{E^-}$), it is sufficient (for more details, see annex A) to find a well-founded ordering of E^+_Q. Therefore, we define a partial stratification bias, which finds - when there exists one - a well founded ordering of E^+_Q, by creating the graph AND/OR (called G) of the positive recursive calls.

6.1 Graph of the positive recursive calls

This graph is constructed as follows:

- the nodes of the graph are the elements of E^+_Q,
- for each ground instance of a clause of $\mathcal{P}$: Q(c_1, ... , c_n) $\leftarrow$ L_1, ... ,L_p such that:
  - (c_1, ... ,c_n) $\in E^+_Q$,
  - if $L_j = R_i(c_1\text{'}, \ldots, c_m\text{'})$, then ($c_1$', ... ,$c_m$') $\in E^+_{R_i}$
  - if $L_j = \neg R_i(c_1\text{'}, \ldots, c_m\text{'})$, then ($c_1$', ... ,$c_m$') $\in E^-_{R_i}$

  we create a virtual node N, OR-son of the node (c_1, ... ,c_n) and having for AND-son, the tuples (c_1", ... ,c_n") such that there exists L_i = Q(c_1", ... ,c_n")

We say that G' is a **subgraph** of G, if G' is a graph satisfying:

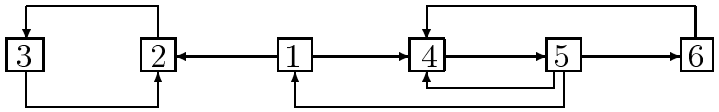
- the nodes of G' are the nodes of G, i.e. the elements of E^+_Q,
- if a node N has at least one virtual OR-son within G, then N has at least one virtual OR-son within G'

- the AND-sons of the virtual node N within G' are the AND-sons of the virtual node N within G.

We define the **partial stratification bias** as follow: let G be the graph of the positive recursions of $\mathcal{P}$; if there exists a subgraph G' of G such that G' contains no cycles, then $\mathcal{P}$ is accepted[4].

6.2 Example

Let $\mathcal{D}$ be the domain {1, 2, 3, 4, 5, 6}, and let us consider the relations r_1 defining a directional network on $\mathcal{D}$, and r_2 a property satisfied by the element 6. We wish to learn the predicate q defined as follow: q(X) is true iff r_1^*(X, 6), where r_1^*(X, 6) is the transitive closure of r_1:

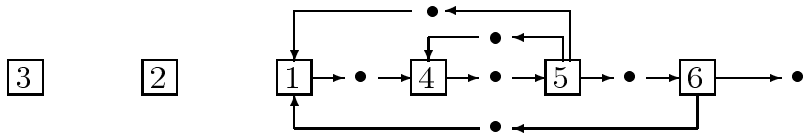


The sets E^+ are defined by:

$E^+_{r_1}$ = {(1,2), (3,2), (2,3), (1,4), (4,5), (5,1), (5,4), (5,6), (6,4)}
$E^+_{r_2}$ = {(6)}
E^+_q = {(1), (4), (5), (6)}

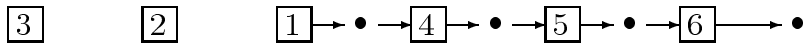
and we assume that the learned program is q(X) ← r_2(X).
q(X) ← r_1(X,Y), q(Y),

(The first bias presented in section 4.1 cannot be applied since for every ordering of $\mathcal{D}$, there exists no index of stratification). The graph of the positive recursions of $\mathcal{P}$ is:



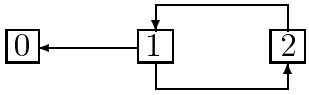
where the virtual nodes are represented by a "•".
The subgraph of G:

[4]The following naive algorithm gives a way to test the existence of a subgraph G' without cycle, by coloring the nodes of G: we first color the leaves of G; if the node M has an OR-son colored, then color N, and if the node M has all its AND-son colored, then color M, until we cannot color new nodes. If every real node of G is colored, then a subgraph without cycle exists



contains no cycles and then $\mathcal{P}$ is accepted. Here, the extensional definitions are total for all the predicates and $\mathcal{P}$ has a total well founded model (in this case, $\mathcal{P}$ satisfies also the "strong" acceptability criterion).

We give now an example for which $\mathcal{P}$ satisfies only the weak acceptability criterion: let r_1 be the relation defining a directional network on $\mathcal{D} = \{0, 1, 2\}$, and r_2 a property satisfied by 0. We wish to learn the predicate q defining by : q(X) iff there exists a way between X and 0, having an even number of arcs:



The sets E^+ are defined by

$E^+_{r_1} = \{(1,2), (2,1), (1,0)\}$	$E^-_{r_1} = \mathcal{D}^2$ - $E^+_{r_1}$
$E^+_{r_2} = \{(0)\}$	$E^-_{r_2} = \mathcal{D}$ - $E^+_{r_2}$
$E^+_q = \{(0), (2)\}$	$E^-_q = \mathcal{D}$ - E^+_q

and the program $\mathcal{P}$: $q(\mathrm{X}) \leftarrow r_2(\mathrm{X})$.
$q(\mathrm{X}) \leftarrow r_1(\mathrm{X,Y}), \neg q(\mathrm{Y})$

is accepted, because the graph of the positive recursions contain no arcs, and then no cycles. The only ground litteral of predicat q in the well founded model is $q(0)$, therefore E^+_Q is not included in $\mathcal{M}^+_{\mathcal{P}}$ and only the weak acceptability criteria is satisfied.

6.3 Critical discussion

On the operational point of view, this bias has especially two defaults:

- on the one hand, it gives an acceptability criterion for the whole program $\mathcal{P}$: when the program is rejected, we come to a deadlock. We could define a more dynamic solution, which construct the subgraph, clause by clause. Then, if a clause is rejected, we can backtrack on the literals of this clause, until we find a new clause satisfying the constraint of coverage, and accepted by this bias,
- on the other hand, if Q is a n-ary predicate and $\mathcal{D}$ contains p elements, the graph contains about p^n nodes, and the control of such a graph can

become expensive. We can weaken this bias, using n graph (G_1, ..., G_n) with p nodes in place of a graph with p^n nodes, where the G_i are defined by: the nodes of each G_i are the element of $\mathcal{D}$ and for each ground instance of a clause satisfying (1) and (2): $Q(c_1, \ldots, c_n) \leftarrow L_1, \ldots, L_m$, we create a virtual node N_i within G_i, such that N_i is an OR-son of c_i, and for all $L_j = Q(c_1', \ldots, c_n')$, we add the AND-son c_i' to the node N_i (i=1...n). Then the condition of existence of a subgraph of G without cycle is replaced by the condition of existence of a subgraph without cycle in one of the G_i.

7 Conclusion

We have studied the semantics of the learned program and defined biases so that the program satisfies at least the weak acceptability criterion. This work is not only theoretical and we are currently developing a system in Quintus Prolog that implements these ideas.

In the case of inductive learning, to be sure that the positive examples do not belong to $\mathcal{M}_{\mathcal{P}}^-$, we must in the computation of T_i^+ consider only known information. If E^+ and E^- are small, it can then be difficult to find a program that covers all the positive examples. We study how to cope with this problem, either by relaxing this constraint or by giving dynamically values to unknown information.

References

[1] Ferrand G., Deransart P., 1992. Proof method of partial correctness and weak completeness for normal logic programs. Joint International Conference and Symposium on Logic Programming, Washington, October-November 1992.

[2] Fitting M., 1985. A Kripke-Kleene semantics for logic programs. Journal of Logic Programming, 2(4), pp. 295-312.

[3] Gelfond M., Lifschitz V., 1988. The stable model semantics for logic programming. proceedings of the fifth Logic Programming Symposium, Association for Logic Programming, R. Kowalski and K. Bowen (Eds.), MIT Press, Cambridge, pp. 1070-1080.

[4] J. W. Lloyd. Foundations of Logic Programming. *Springer Verlag*, 1987.

[5] Muggleton S., Feng C., 1992. Efficient Induction of Logic Programs. Inductive Logic programming. The A.P.I.C. Series N° 38, S. Muggleton (Ed.), Academic Press. pp. 281-298.

[6] Muggleton S., Buntime W., 1992. Machine Invention of First-Order Predicats by Inverting Resolution. Inductive Logic programming. The A.P.I.C. Series N° 38, S. Muggleton (Ed.), Academic Press. pp. 261-280.

[7] Pazzani M., Kibler D., 1992. The Utility of Knowledge in Inductive Learning. Machine Learning, Vol. 9, N°. 1, June 1992, Kluwer Academic Publishers, pp. 56-94.

[8] Plotkin G., 1970. A note on inductive generalization. Machine Intelligence, Vol. 5, Edinburgh University Press, Edinburgh.

[9] Plotkin G., 1971. A further note on inductive generalization. Machine Intelligence, Vol. 6, Edinburgh University Press, Edinburgh.

[10] Przymusinski T., 1990. Well-founded semantics coincides with three-valued stable semantics. Fundamenta Informaticae XIII, IOS Press, pp. 445-463.

[11] Quinlan J.R., 1990. Learning Logical Definitions from Relations. *Machine Learning Journal*, Vol. 5, Kluwer Academic Publishers, pp. 239-266.

[12] de Raedt L., 1992. Interactive theory revision, an inductive logic programming approach. Academic Press Limited.

[13] Rouveirol C., 1991. ITOU: Induction of First Order Theories, proceedings of the International Workshop on Inductive Logic Programming, Portugal, 2-4 March 1991, pp.127-158.

[14] Van Gelder A., Ross K.A., Schlipf J.S., 1991. The well-founded Semantics for General Logic Program. Journal of the ACM, Vol. 38, No. 3, July 1991, 620-650.

[15] Vrain C., Martin L., 1993. Induction de clauses normales: application au système FOIL. Rapport de recherche *LIFO*, 93-6.

Annex A

In order to prove that when the learned program $\mathcal{P}$ satisfies the partial stratification bias then it verifies the weak acceptability criterion, we use the proof method proposed by [1], based on the following definitions : let S and C be two sets of ground atoms,

- the program $\mathcal{P}$ is *bottom-up closed* w.r.t. S and C if for all ground instances $A \leftarrow L_1, \ldots, L_n$ of a clause of $\mathcal{P}$, the following holds :
  - for all positive L_i, $L_i \in S$,
  - for all negative L_i, $L_i = \neg L_i$', L_i' $\in \overline{C}$,

  then $A \in S$,
- a program $\mathcal{P}$ is *top-down closed* w.r.t. S and C if there exists a function f defined on C into a well-ordered set such that for every atom B of C, there is a ground instance $A \leftarrow L_1, \ldots, L_n$ of a clause of $\mathcal{P}$ such that $B = A$ and
  - for all positive L_i, $L_i \in C$ and for all negative $L_i = \neg L_i$', L_i' $\in \overline{S}$,
  - for all positive L_i, $f(L_i) < f(A)$

Given a normal program $\mathcal{P}$ and two sets S and C, to ensure that $M^+_{\mathcal{P}} \subseteq$ S et C $\subseteq \overline{M^-_{\mathcal{P}}}$, it is sufficient to show that $\mathcal{P}$ is bottom-up closed and top-down closed w.r.t. S and C.

Let us recall that the program $\mathcal{P}$ is compound of the learned clauses defining the predicate Q, and a set of clauses defining the predicates R_i : to ensure that the well-founded semantics of this set is exactly the extensional definition E^+_R and E^-_R, we can, for example, create the unit clause $R_i(c_1, \ldots, c_k) \leftarrow$. for all the $(c_1, \ldots, c_k) \in E^+_{R_i}$, and the clause $R_i(c_1, \ldots, c_k) \leftarrow \neg R_i(c_1, \ldots, c_k)$ for all the $(c_1, \ldots, c_k)$ in $E^u_{R_i}$.

We prove now the bottom-up and top-down closure of $\mathcal{P}$, with $S = \overline{E^-}$ and $C = E^+$:

A Proof of the bottom-up closure :

Let $A \leftarrow L_1, \ldots, L_n$ be a ground instance of a clause of $\mathcal{P}$:

- either the head of the clause is $R_i(c_1, \ldots, c_k)$ and by definition, $(c_1, \ldots, c_k) \in \overline{E^-_{R_i}}$ and therefore $(c_1, \ldots, c_k) \in S$,
- or the predicate of the head of the clause is Q. If $A \notin S$, then $A \in E^-_Q$ i.e. A is a negative example, and with the constraints of coverage, A is rejected by every ground instance of $\mathcal{P}$. It means that there exists at least one literal L_j which rejects A :

– if L_j is a positive literal, $L_j = p(c_1, \ldots, c_k)$ $(c_1, \ldots, c_k) \notin (E_p^+ \cup E_p^u)$ and then $(c_1, \ldots, c_k) \notin S$,

– if L_j is a negative literal, $L_j = \neg p(c_1, \ldots, c_k)$, $(c_1, \ldots, c_k) \notin (E_p^- E_p^u)$ and then $(c_1, \ldots, c_k) \in E_p^+ \subset C$.

Finally, when $A \notin S$, there is no ground clause $A \leftarrow L_1, \ldots, L_n$ satisfying "for each positive L_j, $L_j \in S$ and for each negative $L_j = \neg L_j$', L_j' $\in \overline{C}$ ", then $\mathcal{P}$ is bottom-up closed.

B Proof of the top-down closure :

The partial stratification bias enables us to define a function f defined on E^+ into N that satisfies the following conditions:

- $f(R_i(c_1, \ldots, c_m)) = 0$ for each i and each tuple $(c_1, \ldots, c_m)$
- $f(\mathrm{Q}(c_1, \ldots, c_n)) > 0$, for each tuple $(c_1, \ldots, c_n)$
- $f(\mathrm{Q}(c_1, \ldots, c_n)) > f(\mathrm{Q}(c_1', \ldots, c_n'))$ when $(c_1', \ldots, c_n')$ is a son of $(c_1, \ldots, c_n)$ within the subgraph G'.

Let A be an atom of C

- if $A = R_i(c_1, \ldots, c_k)$ $((c_1, \ldots, c_k) \in E_{R_i}^+)$, there exists a unit clause $R_i(c_1, \ldots, c_k) \leftarrow$. in $\mathcal{P}$,
- if $A = Q(c_1, \ldots, c_n)$, the constraints of coverage imply that there exists a ground clause $Q(c_1, \ldots, c_n) \leftarrow L_1, \ldots, L_m$ for which

  - for each positive literal $L_i = p(c_{i_1}, \ldots, c_{i_s})$, $(c_{i_1}, \ldots, c_{i_s}) \in E_p^+ \subset C$,

  - for each negative literal $L_i = \neg p(c_{i_1}, \ldots, c_{i_s})$, $(c_{i_1}, \ldots, c_{i_s}) \in E_p^- \subseteq \overline{S}$.

  Moreover, the partial biais of straitification enables us to choose it so as for each positive literal L_i, $f(L_i) < f(A)$.