A three-valued framework for the induction of general logic programs

Lionel MARTIN, Christel VRAIN [1]

L.I.F.O. université d'Orléans
B.P. 6759 - 45067 Orléans Cedex 2
France
email: {cv,martin}@chambord.univ-orleans.fr
tel: 38 41 72 89
fax: 38 41 71 37

Abstract

We present in this paper a framework to learn general logic programs, i.e., sets of rules which may contain negation in their bodies. It is based on a three-valued logic ({*true, false, undefined* }) that enables to model both the notion of unknown information and the notion of undefined answer of the Prolog interpreter. In this framework, the usual notions of completeness and consistency give rise to three acceptability criteria. Most empirical learning systems are based on the notion of extensional coverage, that is generally not sufficient to ensure that the learned program will really prove the examples. In this paper, the problems due to recursion are handled through the notion of recursive dependencies. It enables to disregard background knowledge for the evaluation of the learned program. Moreover, it gives an estimation of the utility of a clause to really prove examples. It has been applied to single predicate learning in the system ICN.

1 Introduction

In the field of Inductive Logic Programming (I.L.P.), many works deal with the task of learning a definition of a concept from positive and negative examples of this concept and a knowledge base [15, 16, 10, 6, 11]. The predicates specified in the knowledge base are called the basic predicates, and the predicate that must be learned is called the target predicate. The goal is to learn a concept definition, which is complete, i.e., which proves that all the positive examples are *true* , and consistent, i.e., which proves that all the negative examples are *false*.

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

In this paper, we are interested in learning general logic programs without function symbols, i.e., sets of clauses $A \leftarrow L_1, \ldots, L_n$ where A is an atom and $L_i, 1 \leq i \leq n$ are literals (atoms or the negations of atoms). But, for sake of simplicity, in the introduction, we consider only definite programs (that contain no negation). The semantics $\mathcal{M}_{\mathcal{P}}$ of a definite program $\mathcal{P}$ is defined by the set of atoms that are *true* for $\mathcal{P}$, the atoms *false* are obtained by complementarity with the Herbrand base $\mathcal{H}_{\mathcal{P}}$. In the next sections, we extend the definitions 1 and 2 to handle general programs and therefore we introduce a three-valued framework, $\{true, false, undefined\}$.

Definition 1. *Let* BK *be the knowledge base expressed by a definite program and let $\mathcal{P}$ be a definite program that defines the target predicate q. $\mathcal{P}$* proves *that a ground atom e is* true *if $\mathcal{P} \cup \text{BK} \models e$, otherwise $\mathcal{P}$* proves *that e is* false.

In the field of I.L.P., two main approaches can be distinguished:

- interactive theory revision, illustrated by the systems MIS [20] and Clint [16]
- empirical learning, illustrated by FOIL [15] and Golem [10].

In the field of interactive theory revision, the systems check, each time a new clause is generated, whether the whole set of clauses remains complete and consistent w.r.t. the set of positive and negative examples that have already been processed. On the other hand, many approaches in the field of empirical learning are based on the notion of extensional coverage, defined as follows:

Definition 2. *Let $\mathcal{P}$ be a definite program that defines the target predicate, let E^+ and E^- be the sets of positive, respectively negative, examples of the target predicate and let* BK *be a definite program that defines the basic predicates.*
$\mathcal{P}$ extensionally covers *the ground atom e if there exists a ground instance of a clause of $\mathcal{P}$, $l \leftarrow l_1, \ldots, l_n$ such that $e = l$ and for all i, $l_i \in (E^+ \cup \mathcal{M}_{\text{BK}})$.*
$\mathcal{P}$ extensionally rejects *the ground atom e if for all ground instances of a clause of $\mathcal{P}$, $l \leftarrow l_1, \ldots, l_n$ such that $e = l$, there exists a literal l_i with $l_i \in E^- \cup \overline{\mathcal{M}_{\text{BK}}}$.*

When the specification of the examples is total ($E^- = \overline{E^+}$), the two statements "$\mathcal{P}$ extensionally rejects e" and "$\mathcal{P}$ does not extensionally cover e" are equivalent. It is no more true as soon as the specification is partial, as shown by the following example

Example 1.

BK:	$r(a,a) \leftarrow$	$E^+ =$	$\{q(a)\}$
	$r(c,b) \leftarrow$	$E^- =$	$\{q(c)\}$

The program $\mathcal{P}$ composed of the unique clause $q(X) \leftarrow r(X,Y), q(Y)$ does not extensionally cover $q(c)$ and does not extensionally reject it.

When learning a single concept, if the learned definition is not recursive then the notions of extensionally covered and proved true are equivalent. It becomes false as soon as we allow recursive definitions. For instance, the trivial definition of *grand-father*, *grand-father*$(X, Y) \leftarrow$ *grand-father*(X, Y) extensionally covers all the positive examples and rejects the negative ones, but the semantics of the program composed of BK and of this clause contains no information about *grand-father*. No satisfactory syntactic biases can solve this problem, as shown by the following example:

Example 2. Let us consider the target predicate q defined by $E^+ = \{q(a), q(b), q(c)\}$ and $E^- = \{q(d)\}$, and a knowledge base defining two basic predicates p and r.

Case 1	Case 2
$p(c) \leftarrow .$	$p(c) \leftarrow .$
$r(a, b) \leftarrow .$	$r(a, b) \leftarrow .$
$r(b, a) \leftarrow .$	$r(b, c) \leftarrow .$
$r(c, d) \leftarrow .$	$r(c, d) \leftarrow .$

Figure 1: Two definitions of BK

The program $\mathcal{P}$ composed of the two clauses

$$q(X) \leftarrow p(X)$$
$$q(X) \leftarrow r(X, Y), q(Y)$$

extensionally covers the positive examples of q and rejects the negative ones, both when p and r are defined by the clauses given in the first column of figure 1 or when they are defined by the clauses given in the second column. On the other hand, the program composed of $\mathcal{P}$ and of the clauses of column 1 proves only that $q(c)$ is *true*, whereas the program composed of the same learned program $\mathcal{P}$ and of the clauses of column 2 proves that $q(a)$, $q(b)$ and $q(c)$ are *true*.

To deal with these problems and to extend the representation language to general logic programs, we have developed a general framework, based on a three-valued logic ({*true, false, undefined* }). Few works, except as far as we know [1], use a three-valued logic since usually, either a classical two-valued logic, or a four-valued logic ({*true, false, unknown, inconsistent* }) [17, 11] is used.

To handle general programs, an underlying problem is the choice of the semantics of negation. In Logic Programming, two three-valued semantics are commonly used, Fitting semantics [3] and the well-founded semantics [19]. We present here Fitting semantics, but our framework can as well be applied in the well-founded semantics[2]. It differs from the other systems that learn general programs in I.L.P.: either negation is made explicit [11], or they use the classical negation by failure [15] or completion semantics [18], or they restrict the class of general programs that can be learned (for instance in [17] they learn only stratified programs).

[2] Computing the well-founded semantics of general programs is more costly than computing Fitting semantics.

This framework has been applied to the problem of single predicate learning and lead to the system ICN [8].

2 Semantics

2.1 Semantics of general logical programs

In this paper, we consider only general logic programs that contain no function symbols other than constants.

Let us first precise some basic definitions that we will use throughout this paper.

Definition 3. *An* atom *is an expression* $p(t_1, \ldots, t_n)$ *where* p *is a predicate and for all* i, t_i *is either a constant or a variable. An atom* $p(t_1, \ldots, t_n)$ *is* ground *if for all* i, t_i *is a constant. A* literal *is an atom or the negation of an atom. A* positive literal *is an atom and a* negative *literal is the negation of an atom. Let* L *be a literal then* $-L = \neg L$ *if* L *is a positive literal and* $-L = A$ *if* L *is a negative literal,* $L = \neg A$.
A general program *is a set of clauses* $A \leftarrow L_1, \ldots, L_n$ *where* A *is an atom and for all* i, L_i *is a literal.*

Notation Let $\mathcal{P}$ be a general logic program. $U_{\mathcal{P}}$ denotes the Herbrand universe of $\mathcal{P}$, i.e., the set of constants that appear in $\mathcal{P}$. $\mathcal{H}_{\mathcal{P}}$ denotes the Herbrand base of $\mathcal{P}$, i.e., the set of all the ground atoms $p(t_1, \ldots, t_n)$ where p is a predicate symbol that appears in $\mathcal{P}$ and for all i, $t_i \in U_{\mathcal{P}}$. If $S \subseteq \mathcal{H}_{\mathcal{P}}$, then $\overline{S}$ represents the complementary set of S in $\mathcal{H}_{\mathcal{P}}$. We denote $Inst_{\mathcal{P}}$ the set of ground instances of clauses of $\mathcal{P}$.

The semantics of a definite program $\mathcal{P}$ is usually defined by a partition of $\mathcal{H}_{\mathcal{P}}$ in two sets: the set of ground atoms that are *true*, i.e., that are logical consequences of $\mathcal{P}$ and the other ones considered as $false$. Such a semantics is qualified as bivalued, since each ground atom is assigned a truth value, *true* or $false$. It may be impossible to find such a partition for a general program. Let us consider for instance the program $\mathcal{P}_1$ composed of the clause:

$$a \leftarrow \neg a$$

a is not a logical consequence of $\mathcal{P}_1$, and a cannot be false (else a would also be *true*). $\mathcal{P}_1$ has no bivalued semantics[3]. Therefore, three-valued semantics have been defined for general programs.

[3] There exist bivalued semantics for general programs, as for instance the stable semantics [4], but they are defined only for some subsets of general programs.

Definition 4. *Let $\mathcal{P}$ be a logical program and S a set of ground atoms, $\neg S$ denotes the set $\{\neg A | A \in S\}$; a set of ground literals I can be decomposed into $I = I^+ \cup \neg I^-$ where $I^+ = \{A \in \mathcal{H}_\mathcal{P} | A \in I\}$ and $I^- = \{A \in \mathcal{H}_\mathcal{P} | \neg A \in I\}$. A set of ground literals I is a* three-valued interpretation *if $I^+ \cap I^- = \emptyset$; a three-valued interpretation is* total *if $I^+ \cup I^- = \mathcal{H}_\mathcal{P}$, otherwise it is* partial.

The semantics $\mathcal{M}_\mathcal{P}$ of a general logic program $\mathcal{P}$ is defined by 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 $\mathcal{P}$,
- $\mathcal{M}_\mathcal{P}^u = \overline{\mathcal{M}_\mathcal{P}^+ \cup \mathcal{M}_\mathcal{P}^-}$, the set of ground atoms $Undefined$ for $\mathcal{P}$.

$\mathcal{M}_\mathcal{P}$ denotes the semantics of $\mathcal{P}$. It is the set of ground **literals** true for $\mathcal{P}$, $\mathcal{M}_\mathcal{P} = \mathcal{M}_\mathcal{P}^+ \cup \neg\, \mathcal{M}_\mathcal{P}^-$.

Several semantics have been defined for general programs. We give here a characterization of Fitting semantics [3] which is the three-valued extension of Clark's completion semantics, commonly used in ILP. In [8], we give a formalization of the work presented in this paper, based on the well-founded semantics [19], that is widely used in deductive databases.

Fitting semantics [3] Let us consider a three-valued interpretation I and a general program $\mathcal{P}$. A ground atom p is true w.r.t. I in Fitting semantics, if there is a clause $p \leftarrow l_1, \ldots, l_n$ of $Inst_\mathcal{P}$ such that every literal l_i, $1 \leq i \leq n$ is true in I. Let us call $T_\mathcal{P}(I)$ the set of ground literals true w.r.t. I,

$$T_\mathcal{P}(I) = \{\ p \in \mathcal{H}_\mathcal{P} | \exists\ p \leftarrow l_1, \ldots l_n \in Inst_\mathcal{P} \text{ with } \forall i,\ i = 1..n,\ l_i \in I\ \}$$

In the same way, a ground atom p is false w.r.t. I in Fitting semantics if for all clauses of $Inst_\mathcal{P}$, $p \leftarrow l_1, \ldots, l_n$ there exists a literal l_i false in I. Let us call $N_\mathcal{P}(I)$ the set of ground literals false w.r.t. I,

$$N_\mathcal{P}(I) = \{\ p \in \mathcal{H}_\mathcal{P} | \text{ for all clauses } p \leftarrow l_1, \ldots l_n \text{ of } Inst_\mathcal{P}, \exists l_i \text{ such that } -l_i \in I\}$$

The set of **literals** true w.r.t. I is therefore: $\mathcal{F}_\mathcal{P}(I) = T_\mathcal{P}(I) \cup \neg N_\mathcal{P}(I)$.

The operators $T_\mathcal{P}$, $N_\mathcal{P}$ and $\mathcal{F}_\mathcal{P}$ are monotonous; $\mathcal{F}_\mathcal{P}$ has a least fixpoint that represents Fitting semantics of $\mathcal{P}$. It is written $\mathcal{M}_\mathcal{P}^{Fit}$ and is therefore obtained by computing the series $(\mathcal{M}_{\mathcal{P},n})_{n \geq 0}$ defined by $\mathcal{M}_{\mathcal{P},0} = \emptyset$ and $\mathcal{M}_{\mathcal{P},n+1} = \mathcal{F}_\mathcal{P}(\mathcal{M}_{\mathcal{P},n})$.

Example 3.

$$\mathcal{P}_2 : \quad \begin{array}{ll} p \leftarrow \neg a. & \mathcal{F}_{\mathcal{P}_2}(\emptyset) = \{a\} = \mathcal{M}_{\mathcal{P},0} \\ b \leftarrow \neg p. & \mathcal{F}_{\mathcal{P}_2}(\mathcal{M}_{\mathcal{P},0}) = \{a, \neg p\} = \mathcal{M}_{\mathcal{P},1} \\ a \leftarrow . & \mathcal{F}_{\mathcal{P}_2}(\mathcal{M}_{\mathcal{P},1}) = \{a, b, \neg p\} = \mathcal{M}_{\mathcal{P},2} \\ c \leftarrow c. & \mathcal{F}_{\mathcal{P}_2}(\mathcal{M}_{\mathcal{P},2}) = \mathcal{M}_{\mathcal{P},2} = \mathcal{M}_{\mathcal{P}_2}^{Fit} \end{array}$$

This notion of fixpoint gives a method to evaluate Fitting semantics of a ground program that contains no function symbols other than constants.

Evaluation of Fitting semantics Let $\mathcal{P}$ be a ground general logic program. Fitting semantics of $\mathcal{P}$ is computed by iterating the following operations. Initially,

$\mathcal{C} = Inst_{\mathcal{P}}$ and $I = \emptyset$.

(1) delete in the body of the clauses of $\mathcal{C}$ the literals *true* in I

(2) remove from $\mathcal{C}$ each clause containing, in its body, a literal *false* in I

(3) if there exists a clause $e \leftarrow . \in \mathcal{C}$, add e to I and remove from $\mathcal{C}$ all the clauses having as head e,

(4) if, during the operation (2), all clauses having as head e are deleted, add $\neg e$ to I

until $\mathcal{C}$ can be no more changed and then $\mathcal{M}_{\mathcal{P}} = I$.

2.2 Semantics of the learned program

The goal of learning is to induce a general logic program $\mathcal{P}$ defining a predicate q from:

- ▷ an intended interpretation $E = E^+ \cup \neg E^-$ where E^+ is a set of positive examples and E^- a set of negative examples, $(E^+ \cap E^- = \emptyset)$,
- ▷ a set BASE of basic predicates, defined either **intensionally** by a general program BK in which the predicate q does not appear, or **extensionally** by a three-valued interpretation $\mathcal{M}_{\mathrm{BK}} = \mathcal{M}^+_{\mathrm{BK}} \cup \neg\mathcal{M}^-_{\mathrm{BK}}$,

When the basic predicates are intensionally defined, the first step of empirical systems, as the system Golem [10] or the system ICN that we have developed [8], consists in computing the semantics $\mathcal{M}_{\mathrm{BK}}$ of BK so as to use only this semantics during the learning process.

Conversely, given an interpretation I_{BK} on the predicates of BASE, it is always possible to associate a program BK, expressed with predicates of BASE, such that the semantics of BK, $\mathcal{M}_{\mathrm{BK}}$, coincides with I_{BK}: let us consider for instance the program composed of the unit clauses $a \leftarrow$ for all $a \in I^+_{\mathrm{BK}}$, and of the clauses $a \leftarrow a$ for all $a \in \overline{I^+_{\mathrm{BK}} \cup I^-_{\mathrm{BK}}}$. Its semantics is I_{BK} (the reader may be convinced by running the algorithm given in the previous section). Of course, such a program is not very interesting.

To compute the semantics of a program $\mathcal{P}$ defining the target predicate q, w.r.t. a model $\mathcal{M}_{\mathrm{BK}}$ of the knowledge base, we introduce three new operators $\widetilde{T}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}$, $\widetilde{N}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}$ and $\widetilde{F}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}$. Let us write $\mathcal{H}_q$ the set of ground atoms built with the predicate q. If $I = I^+ \cup \neg I^-$ is a three-valued interpretation, we define:

- $\widetilde{T}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}(I) = \{e \in \mathcal{H}_q \mid \exists r \in Inst_{\mathcal{P}}$ such that $head(r) = e$ and $body(r) \subseteq (I \cup \mathcal{M}_{\mathrm{BK}})\}$
- $\widetilde{N}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}(I) = \{e \in \mathcal{H}_q \mid \forall r \in Inst_{\mathcal{P}}$ if $head(r) = e$ then $\exists\, l \in body(r)$ satisfying $-l \in (I \cup \mathcal{M}_{\mathrm{BK}})\}$
- $\widetilde{\mathcal{F}}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}(I) = \widetilde{T}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}(I) \cup \neg\widetilde{N}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}(I)$

Theorem 1. *Let us consider a set of predicates* BASE *and a general program* BK *expressed with predicates of* BASE.
Let us consider a predicate q, $q \notin$ BASE *and a general program, $\mathcal{P}$, satisfying:*
– *the head of each clause of $\mathcal{P}$ is built with the predicate q,*
– *the body of each clause of $\mathcal{P}$ is expressed with predicates of* BASE $\cup \{q\}$.
Let us define the sequence $[\widetilde{M_i}]_{i\geq 0}$ by $\widetilde{M}_0 = \emptyset$ and $\widetilde{M}_i = \widetilde{\mathcal{F}}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}(\widetilde{M}_{i-1})$.

The sequence $[\widetilde{M_i}]_{i\geq 0}$ converges and its limit, written $\mathcal{M}_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}})$, satisfies $\mathcal{M}_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}}) \cup \mathcal{M}_{\mathrm{BK}} = \mathcal{M}_{\mathcal{P}\cup\mathrm{BK}}$

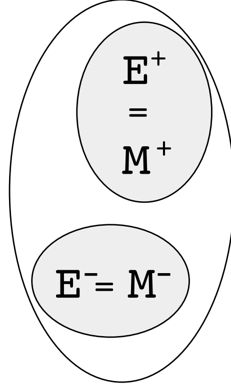
The proof of this theorem is given in annex A. This theorem gives a way to compute the semantics of $\mathcal{P} \cup$ BK, by computing the semantics of $\mathcal{P}$ w.r.t. the interpretation $\mathcal{M}_{\mathrm{BK}}$ (which is computed once for all).

3 Validation of the learned programs

Let us first recall that our learning problem is to find a general program $\mathcal{P}$ defining a target predicate q from the intended interpretation for q, $E = E^+ \cup \neg E^-$, and a knowledge base BK defining a set BASE of basic predicates.

3.1 Acceptability criteria

We extend in our framework - general program and three-valued interpretation - the classical notion of completeness and consistency. Three criteria of acceptability have been defined.

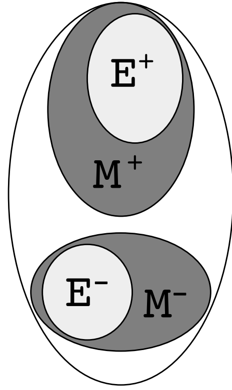


The criterion Ⓐ deals essentially with the situation of a total definition of the knowledge base and of the predicate to learn but it can also be satisfied when they are only partially defined. It expresses that the semantics of the program coincides exactly with the intended interpretation.

Criterion Ⓐ : Reformulation
The learned program satisfies the reformulation criterion if it satisfies the following condition :

$$E^+ = \mathcal{M}^+_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}}) \text{ and } E^- = \mathcal{M}^-_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}})$$

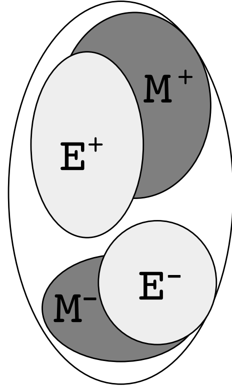
The two following criteria concern more precisely the case of inductive learning. The criterion Ⓑ expresses that the learned program must at least contain the intended interpretation. This constraint can be difficult to reach, when the predicates are partially defined, then we define a weaker criterion, Ⓒ, that requires that the semantics of the learned program is not contradictory with the intended interpretation.



Criterion Ⓑ : Strong induction
The learned program satisfies the strong induction criterion if it satisfies the following condition:

$$E^+ \subseteq \mathcal{M}^+_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}}) \text{ and } E^- \subseteq \mathcal{M}^-_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}}).$$

i.e., each positive example is true for $\mathcal{P}$ and each negative example is false for $\mathcal{P}$.



Criterion Ⓒ : Weak induction
The learned program satisfies the weak induction criterion if it satisfies the following condition:

$$E^+ \subseteq \overline{\mathcal{M}^-_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}})} \text{ and } E^- \subseteq \overline{\mathcal{M}^+_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}})}$$

i.e., none of the negative example is true for $\mathcal{P}$ and none of the positive example is false for $\mathcal{P}$. This condition must be associated to a bias of the learning process that prevents the building of clauses like $q(X_1, \ldots, X_n) \leftarrow q(X_1, \ldots, X_n)$ that would satisfy the weak criterion, since every ground atom of predicate q would be undefined according to Fitting semantics.

In [22], we have expressed a condition so that the learned program satisfies the criterion Ⓒ. It was based on a study of the positive recursive calls between the positive examples. It has been extended to the study of all the recursive calls between positive and negative examples, so that the criterion Ⓑ holds.

3.2 Extensional coverage versus extensional rejection

We extend the definitions of covered and rejected by extension and the definitions of proved true and proved false in our framework.

Definition 5. *Let $\mathcal{M}_{\mathrm{BK}}$ be the semantics of the knowledge base defining the predicates of* BASE, *let q be the target predicate, let $\mathcal{P}$ be a program defining q and let e be a ground atom built with the predicate q.*

- $\mathcal{P}$ *proves* that e is *true* (resp. *false*), if $e \in \mathcal{M}^+_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}})$ (resp. $e \in \mathcal{M}^-_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}})$),
- e is extensionally covered by $\mathcal{P}$ if there exists a clause of $Inst_{\mathcal{P}}$, $e \leftarrow l_1, \ldots, l_n$ such that for all i, $1 \leq i \leq n$, $l_i \in E \cup \mathcal{M}_{\mathrm{BK}}$,

- e is extensionally rejected by $\mathcal{P}$ if for each clause of $Inst_{\mathcal{P}}$, $e \leftarrow l_1, \ldots, l_n$ there exists a literal l_i, $1 \leq i \leq n$, $-l_i \in E \cup \mathcal{M}_{\mathrm{BK}}$.

Most empirical systems [15, 10] build a program that extensionally covers all the positive examples and that extensionally covers no negative examples, using the general algorithm given in figure 2.

- *While the set of positive examples is not empty*
  - *Create a new clause which extensionally covers some positive examples, and which extensionally covers no negative ones,*
  - *Remove the covered examples from the set of positive examples.*

Figure 2: General algorithm based on the notion of coverage

In our three-valued framework, it may happen that a ground atom is neither extensionally covered by a program, nor extensionally rejected, as shown in the following example.

Example 5. Let us consider the domain $U = \{0, 1, 2, 3, 4\}$, the target predicate *even* partially defined by $\{even(0), even(2), \neg even(1), \neg even(3)\}$ ($even(4)$ is unknown) and the basic predicate *succ* completely defined by $\mathcal{M}^+_{\mathrm{BK}} = \{succ(0,1), succ(1,2), succ(2,3), succ(3,4)\}$ et $\mathcal{M}^-_{\mathrm{BK}} = \overline{\mathcal{M}^+_{\mathrm{BK}}}$. The two programs

$$\begin{aligned} \mathcal{P}_1 &= even(X) \leftarrow succ(X,Y), \neg even(Y), \\ \mathcal{P}_2 &= even(X) \leftarrow succ(Y,X), \neg even(Y), \\ &\quad\; even(0) \leftarrow . \end{aligned}$$

extensionally cover the positive examples. $\mathcal{P}_1$ does not extensionally reject $even(3)$ whereas $\mathcal{P}_2$ rejects all the negative examples. Fitting semantics of $\mathcal{P}_1$ is $\mathcal{M}_{\mathcal{P}_1}(\mathcal{M}_{\mathrm{BK}}) = \{\neg even(0), \neg even(2), \neg even(4), even(1), even(3)\}$ and $\mathcal{P}_1$ does not satisfy (B), whereas the semantics of $\mathcal{P}_2$ is $\mathcal{M}_{\mathcal{P}_2}(\mathcal{M}_{\mathrm{BK}}) = \{even(0), even(2), even(4), \neg even(1), \neg even(3)\}$.

Nevertheless, the fact that a program extensionally covers the positive examples and rejects the negative ones is not sufficient to ensure that the program satisfies the criterion (B). For instance, the program $q(X_1, \ldots, X_n) \leftarrow q(X_1, \ldots, X_n)$ satisfies the condition of extensional coverage but Fitting semantics of this program is empty. To deal with this problem, we introduce the notion of recursive dependencies.

4 Recursive dependencies

4.1 Motivation and definition

Example 6. Let us suppose that we want to learn the predicate *even* defined on the domain $U = \{0, 1, 2\}$, and that we know two basic predicates, *zero* et *succ*.

$\mathcal{M}^+_{\mathrm{BK}} = \{zero(0), succ(0,1), succ(1,2)\}$

$\mathcal{M}_{\mathrm{BK}}^{-} = \mathcal{H}_{\mathrm{BK}} - \mathcal{M}_{\mathrm{BK}}^{+}$
$E = \{even(0), \neg even(1), even(2)\}$

The following program extensionally covers the positive example and rejects the negative ones.

$$C_1 : \quad even(X) \leftarrow succ(Y,X), \neg even(Y).$$
$$C_2 : \quad even(X) \leftarrow succ(X,Y), \neg even(Y).$$

(The clause C_1 does not extensionally cover $even(0)$ and the second clause C_2 does not cover $even(2)$, therefore none of these clauses is sufficient to ensure alone the constraints of coverage.)

However, Fitting semantics of this program is empty. Intuitively, to solve for instance the goal $even(0)$, we must solve the goal $\neg even(1)$ (clause C_2) and to know whether $even(1)$ is true, we must solve the goal $\neg even(0)$ (clause C_1) which leads to a loop or the goal $\neg even(2)$ (clause C_2) and then $\neg even(1)$ (clause C_1) which leads also to a loop.

To deal with this problem, we build the set of recursive dependencies, which synthesizes the set of links that exist between positive and negative examples.

In our previous example, the set of recursive dependencies is:

$$\begin{array}{lcl|lcl} even(2) & \leftarrow & \neg even(1) & even(0) & \leftarrow & \neg even(1) \\ even(1) & \leftarrow & \neg even(0) & even(1) & \leftarrow & \neg even(2) \end{array}$$

Figure 3: the set of recursive dependencies

The recursive dependency $even(0) \leftarrow \neg even(1)$. comes from the ground instance of $C_2 : even(0) \leftarrow succ(0,1), \neg even(1)$ It means that to prove that $even(0)$ is *true*, it is sufficient to prove that $\neg even(1)$ is *true*, i.e., that $even(1)$ is *false*.

The recursive dependency $even(1) \leftarrow \neg even(2)$ comes from the ground instance of $C_2 : even(1) \leftarrow succ(1,2), \neg even(2)$. It means that, since $succ(1,2)$ is true, to prove that $even(1)$ is *false* we have to prove that $\neg even(2)$ is *false*, i.e., that $even(2)$ is *true*.

Although $even(0) \leftarrow succ(1,0), \neg even(1)$ is a ground instance of C_1, it does not lead to a recursive dependency, since $succ(1,0)$ is *false* and therefore we could never prove $even(0)$ using this ground clause. In the same way, the clause $even(1) \leftarrow succ(1,1), \neg even(1)$ does not lead to a recursive dependency since $succ(1,1)$ is false and therefore $even(1)$ could never be proved *true* through this ground clause.

Algorithm: Let BK represent the knowledge base that defines the set BASE of basic predicates, let q be the target predicate and let $\mathcal{P}$ be a program that extensionally covers all the positive examples of q and extensionally rejects all the negative examples of q. Let us write $\mathcal{H}_q$ (resp. $\mathcal{H}_{\mathrm{BASE}}$) the subset of $\mathcal{H}_{\mathcal{P}}$, composed of the ground atoms built with the predicate q (resp. built with a predicate of

BASE), $\mathcal{H}_q^* = \mathcal{H}_q \cup \neg\mathcal{H}_q$, the set of ground literals built with the predicate q and $\mathcal{H}_{\text{BASE}}^* = \mathcal{H}_{\text{BASE}} \cup \neg\mathcal{H}_{\text{BASE}}$. The set $\mathcal{P}_{rec}$ of recursive dependencies of $\mathcal{P}$ is built as follows:

- for each clause of $Inst_{\mathcal{P}}$, $p \leftarrow l_1, \ldots, l_n$ ($p \in E^+$) which *extensionally covers* p, we add to $\mathcal{P}_{rec}$ the clause $p \leftarrow l_{i_1}, \ldots l_{i_k}$ where $l_{i_1}, \ldots l_{i_k}$ are the literals of $l_1, \ldots l_n$ that belong to $\mathcal{H}_q^*$,
- for each clause of $Inst_{\mathcal{P}}$, $p \leftarrow l_1, \ldots, l_n$ ($p \in E^-$) that *extensionally rejects* p and such that every literal l_i that belongs to $\mathcal{H}_{\text{BASE}}^*$ is true or undefined in $\mathcal{M}_{\text{BK}}$, i.e. no literals belonging to $\mathcal{H}_{\text{BASE}}^*$ enables to reject p, we add to $\mathcal{P}_{rec}$ the clause $p \leftarrow l_{i_1}, \ldots l_{i_k}$ where $l_{i_1}, \ldots l_{i_k}$ are the literals of $l_1, \ldots l_n$ that belongs to $\mathcal{H}_q^*$ and to $E^- \cup \neg E^+$, i.e., that enables to reject p.

Remark: In $\mathcal{P}_{rec}$, there are only literals of $\mathcal{H}_q^*$. The semantics $\mathcal{M}_{\text{BK}}$ is no more needed.

Theorem 2. *Let $E = E^+ \cup \neg E^-$ the intended interpretation of the target predicate q; let $\mathcal{P}$ be a general program that extensionally covers the positive examples E^+ and extensionally rejects the negative examples E^- w.r.t. the semantics of the basic predicates $\mathcal{M}_{\text{BK}}$; let $\mathcal{P}_{rec}$ be the set of recursive dependencies of $\mathcal{P}$. If $E \subseteq \mathcal{M}_{\mathcal{P}_{rec}}$ then $E \subseteq \mathcal{M}_{\mathcal{P}}(\mathcal{M}_{\text{BK}})$, i.e., if the positive (rep. negative) examples are proved* true *(resp.* false*) in $\mathcal{P}_{rec}$, then it is also satisfied by $\mathcal{P}$.*

This result gives a sufficient condition to ensure that the criterion Ⓑ is satisfied. It is proved in annex B.

4.2 Acceptability criterion for a clause

The theorem 2 enables us to check whether the learned program satisfies the criterion Ⓑ of acceptability. We would like to evaluate during the construction of the program how much positive examples (resp. negative) examples will be really proved.

Let us consider again the example 6 of section 4.1, let us assume that C_1 is the first clause that has been learned and let us study the program composed of this single clause. Its set of recursive dependencies is given by the first column of figure 3. Fitting semantics of C_{1rec} is $\{\neg even(0), even(1), \neg even(2)\}$. The clause C_1 seems unacceptable (it gives the opposite of the expected result) although it intuitively looks a good clause. In fact, as $even(0)$ is not extensionally covered by C_{1rec}, there is no recursive dependency $even(0) \leftarrow \ldots$ and therefore, when we compute the semantics of C_1, the truth value of $even(0)$ is *false*. If now we suppose that $even(0)$ is true (because it will be later extensionally covered by a second clause and then will possibly be proved), then $even(1)$ becomes *false* and $even(2)$ becomes *true*. The semantics of C_{1rec} w.r.t $even(0)$ is the expected result.

To determine whether a clause C can be added to the program $\mathcal{P}$ that is built, we compute the semantics of $(\mathcal{P} \cup C)_{rec}$ w.r.t. $Uncov$ where $Uncov$ is the set of positive examples that are not yet extensionally covered by $\mathcal{P} \cup C$.

If C_1 is accepted, a new clause must be built to extensionally cover $even(0)$. If the learned clause was C_2, it would be rejected since the semantics of $\{C_1 \cup C_2\}_{rec}$ w.r.t. $\emptyset$ (all the positive examples are extensionally covered) is empty. A good clause would be the clause $even(X) \leftarrow zero(X)$.

It may be difficult to find a clause C such that the semantics of $(\mathcal{P} \cup C)_{rec}$ w.r.t. $Uncov$ is equal to the intended interpretation E. Let us consider for instance the following example

Example 7. Let us suppose that we want to learn the predicate p defined on the domain $U = \{a, b, c, d\}$, and that we know three basic predicates, r, s and q defined by:

$\mathcal{M}^+_{\text{BK}} = \{r(a), s(c,a), s(d,d), q(d,a), q(c,c)\}$
$\mathcal{M}^-_{\text{BK}} = \mathcal{H}_{\text{BK}} - \mathcal{M}^+_{\text{BK}}$
$E = \{p(a), \neg p(b), p(c), p(d)\}$

Let us now consider the following complete and consistent program:

$$
\begin{array}{ll}
C_0 : & p(X) \leftarrow r(X). \\
C_1 : & p(X) \leftarrow s(X,Y), p(Y). \\
C_2 : & p(X) \leftarrow q(X,Y), p(Y).
\end{array}
$$

The set of recursive dependencies

$$
\begin{array}{lllllll}
\text{from } C_0 : & p(a) & \leftarrow & & & & \\
\text{from } C_1 : & p(c) & \leftarrow & p(a), & p(d) & \leftarrow & p(d) \\
\text{from } C_2 : & p(d) & \leftarrow & p(a), & p(c) & \leftarrow & p(c)
\end{array}
$$

When the clause C_0 is built, $p(a)$ is extensionally covered and proved. If C_1 is built after C_0, $p(c)$ and $p(d)$ are extensionally covered but only $p(c)$ is proved. If C_2 is built after C_0, $p(c)$ and $p(d)$ are extensionally covered but only $p(d)$ is proved. Whatever the order clauses are built, there exists always a positive example which is extensionally covered but not proved. Therefore we introduce an acceptability rate ϵ.

Definition 6. *Let ϵ be an* **acceptability rate**, $0 \leq \epsilon \leq 1$, *let q be the target predicate defined by the intended interpretation E and let $\mathcal{P}$ be a general program that extensionally covers a subset of E^+ and extensionally rejects all the negative examples of E^-, let C be a new clause that extensionally covers some elements of E^+ and extensionally rejects all the negative examples of E^-. We write $\mathcal{P}' = \mathcal{P} \cup C$. Let Cov (resp. $UnCov$) be the set of positive examples extensionally covered (resp. not covered) by $\mathcal{P}$'.*

The clause C is **acceptable** if the two following conditions are satisfied:

$(1)\frac{card(\mathcal{M}_{\mathcal{P}'_{rec}}(Uncov)\cap Cov)}{card(Cov)} \geq \epsilon$

$(2)\ E^- \subseteq \mathcal{M}^-_{\mathcal{P}'_{rec}}(Uncov)$.

Remark If $\epsilon = 1$ then when all the positive examples are extensionally covered, then all the positive examples are also proved. If $\epsilon \neq 1$, even when all the positive examples are extensionally covered, it may happen that some positive examples are not proved. To deal with this problem, a solution is to add clauses that are not recursive and that extensionally cover some of the remaining examples to the learned program.

5 Application: the system ICN

As already mentioned, we have applied this framework in the system ICN that empirically learns a definition of a target predicate q from the intended interpretation E and a knowledge base BK. We do not detail this system here, we only give the general algorithm suggested by this framework.

- *POS* $\leftarrow E^+$
- *While POS* $\neq \emptyset$ *do*
    - *Create a new acceptable clause which extensionally covers some elements of POS and extensionally rejects each element of* E^-
    - *remove the covered examples from POS.*
- *Compute* $\mathcal{M}_{\mathcal{P}_{rec}}$ *(*$\mathcal{P}$ *is the learned program)*
- *While* $E^+ - \mathcal{M}^+_{\mathcal{P}_{rec}} \neq \emptyset$ *do*
    - *Create a new acceptable and not recursive clause which extensionally covers some elements of* $E^+ - \mathcal{M}^+_{\mathcal{P}_{rec}}$ *and rejects each element of* E^-
    - *Compute* $\mathcal{M}_{\mathcal{P}_{rec}}$

Figure 4: the general algorithm of ICN

A clause is computed by refining the most general clause $q(X_1, \dots X_n) \leftarrow$, where q is the target predicate and $X_1, \dots X_n$ are distinct variables. To achieve this, we compute the gain of each possible literal [4] and literals with a good gain are memorized for an eventual backtrack. We choose the literal L with the best gain from the list previously computed. We repeat the process until all the negative examples are extensionally rejected. A precise description of the computation of a clause, which takes unknown information into account, is given in [22].

[4] A possible literal is an atom or its negation built with a basic predicate or with the target predicate and a variabilisation such that at least one variable of the literal already appears in the clause under construction. It can also be an expression $X = Y$ or $X \neq Y$. Moreover, when the literal is added to the clause, it must at least cover a positive example. The gain is computed from the number of ground instances of the clause which cover (resp. reject) positive examples (resp. negative examples) with and without the new literal.

ICN has been implemented in Sicstus Prolog on a Sun4. Some biases have been implemented as for instance the limitation of the length of the clause to avoid the building of infinite clauses. They are given in [8].

5.1 An example session

Let U be the set $\{-3, 1, 2, 5, \frac{2}{5}, \frac{5}{2}, \frac{1}{5}, \frac{-3}{2}, \frac{2}{-3}, \frac{\frac{-3}{2}}{\frac{1}{5}}, \frac{\frac{5}{2}}{2}, \frac{\frac{-3}{2}}{-3}\}$. We give two basic predicates, $natural(X)$ and $fraction(F, N, D)$ such that the semantics of BK contains the groud atoms

$$\{natural(1), natural(2), natural(5), fraction(\tfrac{2}{5}, 2, 5), fraction(\tfrac{5}{2}, 5, 2),$$
$$fraction(\tfrac{1}{5}, 1, 5), fraction(\tfrac{-3}{2}, -3, 2), fraction(\tfrac{2}{-3}, 2, -3),$$
$$fraction(\tfrac{\frac{-3}{2}}{\frac{1}{5}}, \tfrac{-3}{2}, \tfrac{1}{5}), fraction(\tfrac{\frac{5}{2}}{2}, \tfrac{5}{2}, 2), fraction(\tfrac{\frac{-3}{2}}{-3}, \tfrac{-3}{2}, -3)\}.$$

The predicate to learn, $pos(X)$ is defined by

$$E^+ = \{1, 2, 5, \tfrac{2}{5}, \tfrac{5}{2}, \tfrac{1}{5}, \tfrac{\frac{5}{2}}{2}, \tfrac{\frac{-3}{2}}{-3}\}$$
$$E^- = \{-3, \tfrac{-3}{2}, \tfrac{2}{-3}, \tfrac{\frac{-3}{2}}{\frac{1}{5}}\}.$$

ICN learns the following consistent and complete program $\mathcal{P}$:

$$pos(X) < -natural(X).$$
$$pos(X) < -fraction(X, Y, Z), pos(Y), pos(Z).$$
$$pos(X) < -fraction(X, Y, Z), \neg pos(Y), \neg pos(Z).$$

Let us note that ICN still learns this program if the semantics of BK contains $fraction(\frac{5}{2}, 1, \frac{2}{5})$ and $fraction(\frac{2}{5}, 1, \frac{5}{2})$, but if the semantics of BK contains $fraction(\frac{-3}{2}, 1, \frac{2}{-3})$ and $fraction(\frac{2}{-3}, 1, \frac{-3}{2})$, the previous program is neither consistent nor complete since $pos(\frac{-3}{2})$, $pos(\frac{2}{-3})$, $pos(\frac{\frac{-3}{2}}{\frac{1}{5}})$ and $pos(\frac{\frac{-3}{2}}{-3})$ are then undefined in the Fitting semantics of $\mathcal{P} \cup$ BK.

6 Conclusion

The framework for learning general programs presented in this paper is based on a three-valued logic ({*true, false, undefined* }). It enables to model both the notion of unknown information ($E^+ \cup E^- \subset \mathcal{H}_q$ and $\mathcal{M}^+_{\mathrm{BK}} \cup \mathcal{M}^-_{\mathrm{BK}} \subset \mathcal{H}_{\mathrm{BASE}}$) and the notion of undefined answer of the Prolog interpreter. The usual notion of completeness and consistency has given rise to three acceptability criteria. The problems due to recursion are handled through the notion of recursive dependencies, which enables to disregard the knowledge base. Moreover, it gives an estimation of the utility of a clause to prove examples.

This framework has been applied to single predicate learning [8].

Similar notions have been applied for multiple predicate learning [9] in the case of definite programs. The underlying logic is the classical two-valued logic and the

notion of recursive dependency is then restricted to positive recursive dependency, i.e., recursive dependency between positive examples.

Finally we would like to study how dependency links could be applied in a framework like MIS [20] or Clint [16]. The problem is different, since examples are processed one after the other in an incremental way. Such systems rely on the notion of proofs and the notion of recursive dependencies that we have developed could simplify these proofs (MIS or Clint has to care with the problem of proof termination, which is avoided by the bottom-up computation of the semantics of our set of recursive dependencies).

References

[1] S. Bell, S.Weber, 1993. On the close logical relationship between FOIL and the frameworks of Helft and Plotkin. Proceedings of the Inductive Logic Programming Workshop, Bled, 1993.

[2] R. M. Cameron-Jones, J.R. Quinlan, 1993. Avoiding Pitfalls When Learning Recursive Theories. Proceedings of the Thirteen International Joint Conference on Artificial Intelligence, Chambéry, France, August 28 - September 3, 1993, Vol. 2, pp. 1050-1055.

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

[4] 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.

[5] Helft N., 1987. Inductive Generalization: A Logical Framework, Progress in Machine Learning, proceedings of EWSL 87, I. Bratko, N. Lavrac (Eds.), Sigma Press, Bled, Yugoslavia, pp. 149-157.

[6] Lavrač N., Džeroski S., 1992. Inductive Learning of Relations from Noisy Examples. Inductive Logic programming. The A.P.I.C. Series N° 38, S. Muggleton (Ed.), Academic Press, pp.495-516.

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

[8] Martin L., Vrain C., 1995. Apprentissage empirique d'un concept: le système ICN, Research report LIFO 95-3 University of Orléans.

[9] Martin L., Vrain C., 1995. MULT_ICN: an empirical multiple predicate learner. In these proceedings - 5th Inductive Logic Programming workshop - (also, research report LIFO 95-4, University of Orléans)

[10] 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.

[11] K. Morik, S. Wrobel, J.U. Kietz, W. Emde, 1993. Knowledge Acquisition and Machine learning: Theory, Methods and Applications. Academic Press, London.

[12] 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.

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

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

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

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

[17] de Raedt L., Lavrac N., Dzeroski S., 1993. Multiple Predicate Learning. Proceedings of the Thirteen International Joint Conference on Artificial Intelligence, Chambéry, France, August 28 - September 3, 1993, Vol. 2, pp. 1037-1043.

[18] de Raedt L., Bruynooghe M.,1993. A Theory of Clausal Discovery. Proceedings of the Thirteen International Joint Conference on Artificial Intelligence, Chambéry, France, August 28 - September 3, 1993, Vol. 2, pp. 1058-1063.

[19] 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.

[20] Shapiro E. Y., 1982. Algorithmic Program Debugging. ACM Distinguished Dissertation, MIT press.

[21] Vrain C., Martin L., 1994. Induction de clauses normales : sémantique du programme appris. Actes des neuvièmes journées francophones sur l'Apprentissage, JAVA94, Strasbourg 23-25 Mars 1994, pp. J-1,J-14.

[22] Vrain C., Martin L., 1994. Inductive learning of normal clauses. Machine Learning: ECML-94, Lecture Notes in Artificial Intelligence 784, F. Bergadano, L. De Raedt (Eds.), Springer Verlag, pp. 435-438.

A Proof of theorem 1

Theorem 1. *Let us consider a set of predicates* BASE *and a general program* BK *expressed with predicates of* BASE.
Let us consider a predicate q, $q \notin$ BASE *and a general program, $\mathcal{P}$, satisfying:*

– the head of each clause of $\mathcal{P}$ is built with the predicate q,
– the body of each clause of $\mathcal{P}$ is expressed with predicates of $\textsc{base} \cup \{q\}$.
Let us define the sequence $[\widetilde{M}_i]_{i\geq 0}$ *by* $\widetilde{M}_0 = \emptyset$ *and* $\widetilde{M}_i = \widetilde{\mathcal{F}}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}(\widetilde{M}_{i-1})$.
1. The sequence $[\widetilde{M}_i]_{i\geq 0}$ *converges.*
2. Its limit, written $\mathcal{M}_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}})$, *satisfies* $\mathcal{M}_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}}) \cup \mathcal{M}_{\mathrm{BK}} = \mathcal{M}_{\mathcal{P}\cup\mathrm{BK}}$

Proof of 1. Let us prove by induction that $\widetilde{M}_i \subseteq \widetilde{M}_{i+1}$.

The case $i = 0$ is obvious, let us suppose that this result holds for $i-1$:

– If e is an element of $\widetilde{M}_i^+$ ($= \widetilde{T}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}(\widetilde{M}_{i-1})$), then there exists in $Inst_{\mathcal{P}}$ a clause $e \leftarrow l_1, \ldots, l_n$ such that for all j, $l_j \in (\widetilde{M}_{i-1} \cup \mathcal{M}_{\mathrm{BK}})$. But, $\widetilde{M}_{i-1} \subseteq \widetilde{M}_i$ therefore for all j, $l_j \in (\widetilde{M}_i \cup \mathcal{M}_{\mathrm{BK}})$, i.e., $e \in \widetilde{M}_{i+1}^+$.

– If e is an element of $\widetilde{M}_i^- = \widetilde{N}_{\mathcal{P},\mathcal{M}_{\mathrm{BK}}}(\widetilde{M}_{i-1})$, then for all clauses of $Inst_{\mathcal{P}}$, $e \leftarrow l_1, \ldots, l_n$, there exists j, $1 \leq j \leq n$, $-l_j \in (\widetilde{M}_{i-1} \cup \mathcal{M}_{\mathrm{BK}})$. But, $\widetilde{M}_{i-1} \subseteq \widetilde{M}_i$ therefore for all clauses of $Inst_{\mathcal{P}}$, $e \leftarrow l_1, \ldots, l_n$, there exists j, $1 \leq j \leq n$, $-l_j \in (\widetilde{M}_i \cup \mathcal{M}_{\mathrm{BK}})$, i.e., $e \in \widetilde{M}_{i+1}^-$.

– The set $\mathcal{H}_q$ is finite, therefore there exists k such that $\widetilde{M}_k = \widetilde{M}_{k-1}$ and for all $j \geq k$, $\widetilde{M}_j = \widetilde{M}_{k-1}$. We denote $\mathcal{M}_{\mathcal{P}}(\mathcal{M}_{\mathrm{BK}})$ this limit.

Proof of 2. In the following, if $\mathcal{P}$ is a logical program, we write $[M_{\mathcal{P},i}]_{i\geq 0}$ the sequence defined by:

– $M_{\mathcal{P},0} = \emptyset$ and
– $M_{\mathcal{P},i} = T_{\mathcal{P}}(M_{\mathcal{P},i-1}) \cup \neg.N_{\mathcal{P}}(M_{\mathcal{P},i-1})$

where the operators T and N are the classical operators of Fitting semantics.

2.1. $\mathcal{M}_{\mathrm{BK}} \subseteq \mathcal{M}_{\mathcal{P}\cup\mathrm{BK}}$
– $M_{\mathrm{BK},0} \subseteq M_{\mathcal{P}\cup\mathrm{BK},0}$, obvious.
– If $M_{\mathrm{BK},i} \subseteq M_{\mathcal{P}\cup\mathrm{BK},i}$, then :

– for all $e \in M_{\mathrm{BK},i+1}^+$, there exists a clause of $Inst_{\mathrm{BK}}$, $e \leftarrow l_1, \ldots, l_n$ satisfying for all j, $l_j \in M_{\mathrm{BK},i}$. But, $M_{\mathrm{BK},i} \subseteq M_{\mathcal{P}\cup\mathrm{BK},i}$, therefore $e \in M_{\mathcal{P}\cup\mathrm{BK},i+1}^+$.

– for all $e \in M_{\mathrm{BK},i+1}^-$ and for all clauses of $Inst_{\mathrm{BK}}$, $e \leftarrow l_1, \ldots, l_n$, there exists j, $-l_j \in M_{\mathrm{BK},i}$. But, $M_{\mathrm{BK},i} \subseteq M_{\mathcal{P}\cup\mathrm{BK},i}$, therefore $e \in M_{\mathcal{P}\cup\mathrm{BK},i+1}^-$.

The set of ground literals is finite, therefore, there exists N such that $\mathcal{M}_{\mathrm{BK}} = M_{\mathrm{BK},N}$. Therefore, $\mathcal{M}_{\mathrm{BK}} \subseteq M_{\mathcal{P}\cup\mathrm{BK},N}$, and consequently, $\mathcal{M}_{\mathrm{BK}} \subseteq \mathcal{M}_{\mathcal{P}\cup\mathrm{BK}}$. Moreover, for all j, $\mathcal{M}_{\mathrm{BK}} \subseteq M_{\mathcal{P}\cup\mathrm{BK},N+j}$.

2.2. $\mathcal{M}_{\mathcal{P}}(\textsc{bk}) \subseteq \mathcal{M}_{\mathcal{P}\cup\mathrm{BK}}$.

We show by induction that for all i, $\widetilde{M}_i \subseteq M_{\mathcal{P}\cup\mathrm{BK},N+i}$. The case $i = 0$ is obvious and let us suppose that the result holds for i.

– Let $e \in \widetilde{M}_{i+1}^+$. There exists in $Inst_{\mathcal{P}}$ a clause $e \leftarrow l_1, \ldots, l_n$ such that for all j, $l_j \in \widetilde{M}_i \cup \mathcal{M}_{\mathrm{BK}}$. But, $\widetilde{M}_i \subseteq M_{\mathcal{P}\cup\mathrm{BK},N+i}$ and $\mathcal{M}_{\mathrm{BK}} \subseteq M_{\mathcal{P}\cup\mathrm{BK},N+i}$, therefore $e \in M_{\mathcal{P}\cup\mathrm{BK},N+i+1}^+$.

– Let $e \in \widetilde{M}^-_{i+1}$. For all clauses, $e \leftarrow l_1, \ldots, l_n$, of $Inst_{\mathcal{P}}$, there exists j $-l_j \in \widetilde{M}_i \cup \mathcal{M}_{\text{BK}}$. But, $\widetilde{M}_i \subseteq M_{\mathcal{P}\cup\text{BK},N+i}$ and $\mathcal{M}_{\text{BK}} \subseteq M_{\mathcal{P}\cup\text{BK},N+i}$, therefore $e \in M^-_{\mathcal{P}\cup\text{BK},N+i+1}$.

2.3. We show that $\mathcal{M}_{\mathcal{P}\cup\text{BK}} \subseteq \mathcal{M}_{\mathcal{P}}(\text{BK}) \cup \mathcal{M}_{\text{BK}}$. by proving that for all i, $M_{\mathcal{P}\cup\text{BK},i} \subseteq \widetilde{M}_i \cup M_{\text{BK},i}$.
The case $i = 0$ is obvious. Let us suppose that the result holds for i.

– Let $e \in M^+_{\mathcal{P}\cup\text{BK},i+1}$. There exists in $Inst_{\mathcal{P}\cup\text{BK}}$ a clause $e \leftarrow l_1, \ldots, l_n$ such that for all j, $l_j \in M_{\mathcal{P}\cup\text{BK},i}$. Therefore, by hypothesis of induction, for all j, $l_j \in \widetilde{M}_i \cup M_{\text{BK},i}$.
If the clause belongs to $Inst_{\text{BK}}$, then all the predicates l_j are built with a predicate of BASE and therefore belong to $M_{\text{BK},i}$ and $e \in M^+_{\text{BK},i+1}$.
If the clause belongs to $Inst_{\mathcal{P}}$, then $e \in \mathcal{H}_q$ and since $M_{\text{BK},i} \subseteq \mathcal{M}_{\text{BK}}$, then for all j, $l_j \in \widetilde{M}_i \cup \mathcal{M}_{\text{BK}}$. Therefore, $e \in \widetilde{M}^+_{i+1}$.
In both cases, $e \in \widetilde{M}^+_{i+1} \cup M^+_{\text{BK},i+1}$.

– Let $e \in M^-_{\mathcal{P}\cup\text{BK},i+1}$. For all clauses $e \leftarrow l_1, \ldots, l_n$ in $Inst_{\mathcal{P}\cup\text{BK}}$ there exists j, $-l_j \in M_{\mathcal{P}\cup\text{BK},i}$. Therefore, by hypothesis of induction, $-l_j \in \widetilde{M}_i \cup M_{\text{BK},i}$. (a)
If e is built with the predicate q, for all clauses $e \leftarrow l_1, \ldots, l_n$ in $Inst_{\mathcal{P}}$, there exists j, $-l_j \in \widetilde{M}_i \cup M_{\text{BK},i}(a)$ and since $M_{\text{BK},i} \subseteq \mathcal{M}_{\text{BK}}$, then $-l_j \in \widetilde{M}_i \cup \mathcal{M}_{\text{BK}}$. Therefore, $e \in \widetilde{M}^-_{i+1}$.
If e is built with a predicate of BASE, then for all clauses of $Inst_{\text{BK}}$, $e \leftarrow l_1, \ldots, l_n$ there exists j, $-l_j \in \widetilde{M}_i \cup M_{\text{BK},i}(a)$ and since all clauses of $Inst_{\text{BK}}$ are built with predicates of BASE, $-l_j \in M_{\text{BK},i}$. Therefore $e \in M^-_{\text{BK},i+1}$.
In both cases, $e \in \widetilde{M}^-_{i+1} \cup M^-_{\text{BK},i+1}$.

B Proof of theorem 2

Theorem 2. *Let $E = E^+ \cup \neg E^-$ be the intended interpretation of the target predicate q; let $\mathcal{P}$ be a general program that extensionally covers the positive examples E^+ and extensionally rejects the negative examples E^- w.r.t. the semantics of the basic predicates $\mathcal{M}_{\text{BK}}$; let $\mathcal{P}_{rec}$ be the set of recursive dependencies of $\mathcal{P}$. If $E \subseteq \mathcal{M}_{\mathcal{P}_{rec}}$ then $E \subseteq \mathcal{M}_{\mathcal{P}}(\mathcal{M}_{\text{BK}})$, i.e., if the positive (rep. negative) examples are proved* true *(resp.* false*) in $\mathcal{P}_{rec}$, then it is also satisfied by $\mathcal{P}$.*

Proof of theorem 2. To prove this result, we prove by induction that for all i, $E \cap M_{\mathcal{P}_{rec},i} \subseteq \widetilde{M}_i$. Then, since there exists two integers N and P such that $\mathcal{M}_{\mathcal{P}_{rec}} = M_{\mathcal{P}_{rec},N}$ and $\mathcal{M}_{\mathcal{P}}(\mathcal{M}_{\text{BK}}) = \widetilde{M}_P$, $E \cap M_{\mathcal{P}_{rec},max(N.P)} \subseteq \widetilde{M}_{max(N.P)}$, i.e., $E \cap \mathcal{M}_{\mathcal{P}_{rec}} \subseteq \mathcal{M}_{\mathcal{P}}(\mathcal{M}_{\text{BK}})$. The result is then obvious.

The case $i = 0$ is obvious. Let us suppose that the result holds for i and let e be an element of $E \cap M_{\mathcal{P}_{rec},i+1}$.

– If $e \in E^+$ then $e \in M^+_{\mathcal{P}_{rec},i+1}$. Therefore, there exists a clause C_{rec} of $\mathcal{P}_{rec}$, $e \leftarrow l_1, \ldots, l_n$ such that for i, $l_i \in M_{\mathcal{P}_{rec},i}$. Since the clause C_{rec} belongs to $\mathcal{P}_{rec}$ and since $e \in E^+$, there exists an instance C of a clause of $\mathcal{P}$, $e \leftarrow t_1, \ldots, t_p$ that extensionally covers e and that satisfies: for all literal t_j, $1 \leq j \leq p$, built with a predicate of q, there exists a literal l_i, $1 \leq i \leq p$ such that $l_i = t_j$. For all literal t_j of C, $t_j \in E \cup \mathcal{M}_{\text{BK}}$ (*C extensionally covers e*). If t_j is built with a predicate of BASE, $t_j \in \mathcal{M}_{\text{BK}}$. If t_j is a literal built with q, $t_j \in E$ and there exists a literal l_i, $1 \leq i \leq p$ such that $l_i = t_j$. We have $t_j(= l_i) \in M_{\mathcal{P}_{rec},i}$ and $t_j \in E$, therefore by hypothesis of induction, $t_j \widetilde{M}_i$. In all cases, $t_j \in \mathcal{M}_{\text{BK}} \cup \widetilde{M}_i$ and consequently, $e \in \widetilde{M}^+_{i+1}$.

– If $e = \neg e'$ with $e' \in E^-$ then $e' \in M^-_{\mathcal{P}_{rec},i+1}$. For each clause C of $Inst_{\mathcal{P}}$, $e' \leftarrow t_1, \ldots, t_p$, there exists a literal t_j with $-t_j \in \mathcal{M}_{\text{BK}} \cup E$. (*$C$ extensionally rejects e'*). If there exists a literal t_j, built with a predicate of BASE, with $-t_j \in \mathcal{M}_{\text{BK}} \cup E$, then $-t_j \in \mathcal{M}_{\text{BK}}$. Otherwise, (*Construction of $\mathcal{P}_{rec}$*), there exists a clause C_{rec} of $\mathcal{P}_{rec}$, $e' \leftarrow l_1, \ldots, l_n$, where $l_1, \ldots, l_n$ are the literals of $\{t_1, \ldots, t_p\}$ that belong to $E^- \cup \neg E^+$. Since $e' \in M^-_{\mathcal{P}_{rec},i+1}$, there exists a literal l_i that satisfies $-l_1 \in M_{\mathcal{P}_{rec},i}$. By hypotheses of induction $(-l_1 \in E \cap M_{\mathcal{P}_{rec},i})$, $-l_i \in \widetilde{M}_i$. In both cases, there exists a literal t_j of C with $-t_j \in \mathcal{M}_{\text{BK}} \cup \widetilde{M}_i$. Therefore $e' \in \widetilde{M}^-_{i+1}$, i.e., $e(= \neg e') \in \widetilde{M}_{i+1}$.