Identification in the limit and Constraint ILP (Work in Progress)

G. Richard

TRIVIA Research Team
Université des Antilles et de la Guyane
UFR Sciences - Campus de Fouillole 97159 - Pointe-à-Pitre (France)
Gilles.Richard@univ-ag.fr

Abstract. Concerning the full scope of machine learning, there are several formal models. Among these available models , the identification-in-the-limit model is probably the more suited to Inductive logic Programming. This is a model of interest not only because a large class of ILP processes fits with, but also because it is possible to establish a clean relationship with metric methods which are well-known in the pure logic programming field. In this paper, taking into account the ever increasing success of constraint programming, we explore the possibility of extension of this relationship to the constraint ILP setting. We first address the problem of defining distances between constraint interpretations. We present some distances and examine the behavior of some classes of programs with respect to theses distances. Furthermore, we examine, in a constraint framework, the relationship with the identification in the limit notion and we prove that, under some conditions, the learning process is consistent with it. So our paper is in-between constraint logic programming and machine learning.

Keywords: *learning model, inductive logic programming, constraint, distance*

1 Introduction

It is a central task in machine learning to chose an adequate learning model : doing that, questions such as "under what conditions is successful learning possible or impossible ?" or "under what conditions a given learning algorithm will be successful ?" make sense. A very powerful model is the *probably approximatively correct* (PAC) model of [17] derived from computational complexity theory. Unfortunately, a large majority of results concerning PAC learnability for *inductive logic programming* (ILP) are negative. So, one may leave away complexity notions and turn to an other model dealing with finite time convergence of a computational learning process. This model is known as the *identification-in-the-limit* model and is due to Gold ([5]). Despite some well-known negative results, there is a large class of ILP issues enjoying positive results with regard to the Gold model. Let us recall here the standard ILP setting and its translation within Gold formalism. In standard inductive logic programming, a concept C is a Herbrand interpretation (i.e. a subset of the Herbrand base $\mathcal{B}$) and the result of the learning process is a logic program P. Now, what does it means that we have learned C with P ? Informally at this step, this means that the "implicit" information deducible from (or coded by) P "covers" in some sense C. In the pure case, it is admitted that this implicit knowledge is the least Herbrand model of P denoted M_P. Generally, M_P (or P) is just an "approximation" of C. To learn a concept C, we are supposed to be given an infinite sequence $\{e_n \mid n \in \mathbb{N}\}$ of examples and the learning machine outputs a sequence $\{P_n \mid n \in \mathbb{N}\}$ of approximations of the target concept C, each P_n covering $S_n = \{e_0, e_1, \ldots, e_n\}$. The process fits the identification-in-the-limit notion if there is an n such that $M_{P_n} = C$. If a learning process fails to get this requirement then it is natural to search for an estimation of the "quality" of the learned hypothesis P with respect to the target concept C. We are led to compare objects of the same nature i.e. two subsets of $\mathcal{B}$: C and M_P. A solution is to evaluate a distance between the two subsets considered as elements of a metric space. It remains to design a metric capturing the intuition. When a metric d is defined over $2^{\mathcal{B}}$, the quality of the hypothesis P is measured with $d(C, M_P)$: so, a good learning algorithm has to output an hypothesis P such that $d(C, M_P)$ is minimal. However, the distance has to follow some minimal requirements to be really interesting. We would like that its definition fits with the previous notion of identification-in-the-limit. Assuming that $lim_{n\infty} S_n = C$ in the set theoretical sense, we hope that $lim_{n\infty} d(C, sem(P_n)) = 0$. As exhibited in the paper of [13], this is not a so natural behavior. The distance d has to be properly defined to fit with the previous requirement. As far as we know, [13] was the first one to effectively define a distance satisfying this requirement in the pure logic programming setting. This result shows, in some sense, that learning in the limit is an adequate model for ILP.

Our aim, in this paper, is to examine the possibility of extension of the works of [13] to the full setting of constraint inductive logic programming. In effect, it is now largely agreed that introduction of constraints is necessary for at least two reasons :

- To efficiently and effectively deal over specific data (numerical values, finite domain values, intervals, ...),
- To easily introduce negative examples like $([\![x \neq 0]\!], p(x))$ specifying with a single constrained atom an infinite set of counter-examples for property p.

Consequently, there is a lot of works addressing this problem : one may cite [7, 11, 2, 15, 10]. These works focus on the integration in classical inductive process of new formulas (the constraints) which will be treated with specific algorithms. In this paper, we are mainly concerned by the formal setting of ILP within a constraint framework, whatever the underlying machinery would be. Of course, in the case of constraint setting, the examples, the covering and the identification in the limit notions have to be revisited (section 3.2). Our paper is organized as follows : section 2 is devoted to a presentation of the constrained logic programming framework. This section is a little bit technical and some familiarity with constraint logic programming and semantics topics is assumed. Then, in section 3, we revisit the main notions of ILP in a constraint setting, specifically we give a suitable notion of *concept*, *example*, *covering* and *identification in the limit.* In section 4, we define a distance over the set of constrained interpretations and we examine some properties, especially its relationship with standard lattice-theoretic limit equalities for some classes of programs. A consistent behavior is a minimal requirement for such a distance. In section 5, we discuss the possibility to extend our distance to fit with identification-in-the-limit model. We conclude with section 6 and we give some hints concerning potential developments.

2 Constraint framework

The problem of defining a clean semantics for functional evaluations was a challenging problem in Logic Programming until the full framework of constraint logic programming was settled ([6]). This section recalls these works and is subdivided into 2 parts : the syntax and the semantics parts. We advise to report to [6, 9] for an extensive presentation.

2.1 Constraint language and constraint programs : syntax

Given an infinite set of variables $\mathcal{V}$, a finite set of function symbols Σ, a finite set of constraint predicate symbols Π_c supposed to contain a binary symbol "=" (these are the *built-in* or *interpreted* predicates), and a finite set of program predicate symbols Π_p (these are the *user-defined* or *uninterpreted* predicates), we define two first order languages:

1. the language $L(\mathcal{V}, \Sigma, \Pi_c)$ to express the constraints,
2. the language $L(\mathcal{V}, \Sigma, \Pi_p)$ used to write the program, whose set of atoms will be denoted $ATOM$. A literal is either an atom A or a negated atom $\neg A$. The set of literals is denoted LIT.

In the following sections, $\overline{x}$ will denote a list of variables and the same convention will apply to terms and literals. Given an expression e, $Var(e)$ will denote the set of free variables appearing in e.

A $CLP(\mathcal{A})$ language is defined in the context of a particular structure which determines the meaning of the function and relation symbols appearing in the language. More precisely, such a structure is constituted of:

1. An interpretation $\mathcal{A}$ of $L(\mathcal{V}, \Sigma, \Pi_c)$, with domain $\mid \mathcal{A} \mid$ over which the computation is to be performed. The binary predicate "=" is interpreted as identity in $\mid \mathcal{A} \mid$.

2. A specific class $\mathcal{L}$ of formulæ in $L(\mathcal{V}, \Sigma, \Pi_c)$ starting from the set of atomic (or primitive) constraints, and containing two special constraints $true$ and $false$ supposed to be identically $true$ and $false$. $\mathcal{L}$ is generally considered as closed under variable renaming, conjunction, existential quantification and negation.

The notions of *valuation*, *satisfaction* and *truth* are the usual ones: $\mathcal{A}\ , v \models c$ expresses the fact that the valuation v satisfies the constraint c in the interpretation $\mathcal{A}$. $\mathcal{A} \models c$ denotes the fact that c is true in $\mathcal{A}$. $\exists c$ ($\forall c$) denotes the full existential (universal) closure of c. $\mathcal{A} \models \exists c$ means that the constraint c is satisfiable or consistent in $\mathcal{A}$. In practice, some minimal properties are supposed about the constraints and their interpretation. It is insured that their satisfiability is decidable in $\mathcal{A}$ i.e. that there exists a decidable finite first-order theory $\mathcal{T}$ such that:

1. $\mathcal{A}$ is a model of $\mathcal{T}$,
2. for any constraint c, either $\mathcal{T} \models \exists c$ or $\mathcal{T} \models \neg\exists c$ (i.e. each model of $\mathcal{T}$ is a model of $\exists c$ or each model of $\mathcal{T}$ is a model of $\neg\exists c$). This second property is called *satisfaction completeness*.

A *normal constraint clause* is a formula $A :\!-\ \mathrm{c}, \mathrm{B}_1, ..., \mathrm{B}_\mathrm{n}$ (also denoted $A :\!-\ \mathrm{c}, \beta$ for sake of conciseness) where A is an atom and the $B_i's$ belong to LIT and c belongs to $\mathcal{L}$. A is the *head* and $B_1, ..., B_n$ the *body*. When the body is empty, $A \leftarrow c$ is a *constraint fact*. A *normal constraint logic program* is a set of normal constraint clauses. Formally, a *constraint goal* is merely a clause of the form $:\!-\ \mathrm{c}, \beta$ (we usually omit the $\leftarrow$ symbol). As usual, we assume that all literals in goals or program clauses contain no function symbols : a clause $p(x+1, y) :\!-\ (\mathrm{x} > \mathrm{y}), \mathrm{p}(\mathrm{x}, \mathrm{y}+1)$ is considered as a shorthand for $p(z, y) :\!-\ (\mathrm{z} = \mathrm{x}+1, \mathrm{z}' = \mathrm{y}+1, \mathrm{x} > \mathrm{y}), \mathrm{p}(\mathrm{x}, \mathrm{z}')$. This canonical form allows to simplify the presentation without any loss of generality. In the next subsection, we introduce the declarative semantics $\mathcal{L}(P)$ for a CLP program P.

2.2 Constraint language and constraint programs : semantics

A *constraint atom* is a couple (c, A) where c is a satisfiable constraint such that $Var(c) \subseteq Var(A)$. The set of constraint atoms is denoted $\mathcal{B}$ (consequently $(true, A) \in \mathcal{B}$ for each $A \in ATOM$). It is convenient to introduce an extended domain which will play for CLP the role of the Herbrand universe in standard logic programming.

An n+1-tuple $< p, d_1, ..., d_n >, p \in \Pi_p, d_i \in \mid \mathcal{A} \mid$ will be denoted $p(d_1, ..., d_n)$ and called an *$\mathcal{A}$-atom*. All these objects constitute the *$\mathcal{A}$- base*, denoted $\mathcal{B}_\mathcal{A}$. A valuation v could be extended to the set of atoms, $ATOM$, using the previous $\mathcal{A}$-base: for each $A = p(t_1, ..., t_n) \in ATOM$, $v(A) = p(v(t_1), ..., v(t_n)) \in \mathcal{B}_\mathcal{A}$ and $v(\neg A) = \neg v(A)$. By extension, if $\beta = B_1, ..., B_n$, then $v(\beta) = v(B_1), ..., v(B_n)$.

For a given constraint atom (c, A), the set of its *ground instances* is defined by:

$$[(c, A)]_{\mathcal{A}} = \{v(A) \in \mathcal{B}_{\mathcal{A}} \mid v\colon V \to \mid \mathcal{A} \mid, \text{ and } \mathcal{A}, v \models c\}.$$

Clearly, this notion of *groundness* is an abuse of terminology. Using this notion, a clause $p(\overline{x})$:– c, α is viewed as a finite representation of its ground instances $v(p(\overline{x}))$:– $\mathrm{v}(\alpha)$ for each valuation v satisfying c. If $I \subset \mathcal{B}$, then $[I]_{\mathcal{A}}$ is its set of ground instances. Note that $[\mathcal{B}]_{\mathcal{A}}$ is included in the $\mathcal{A}$-base, $\mathcal{B}_{\mathcal{A}}$. When this is clear from the context, the subscript $\mathcal{A}$ is omitted.

Concerning normal CLP programs, the formal semantics is described in terms of partial constraint interpretations. A *partial constraint interpretation* is a couple $I =< I^+, I^- >$ of sets of constraint atoms satisfying a consistency condition: $[I^+]_{\mathcal{A}} \cap [I^-]_{\mathcal{A}} = \emptyset$. I^+ denotes the set of $true$ constraint atoms and I^- denotes the set of $false$ constraint atoms. The set of partial constraint interpretations denoted $\mathcal{I}$ is a complete inf-semi-lattice[1] for set inclusion over $true$ and $false$ constraint atoms. This means that every non empty directed subset of $\mathcal{I}$ has a least upper bound denoted lub. It is not a lattice as the union of two interpretations may not be an interpretation due to the consistency condition. As usual, the logical meaning of a normal program P is defined by the way of the Clark Completion, which is a set of formulas obtained by adding $\mathcal{T}$ to a formula P^* which is the conjunction of the following formulas:

$$\forall \overline{x}\, p(\overline{x}) \Leftrightarrow \bigvee_{i=1}^{n} (\exists \overline{y_i}\; c_i \wedge \beta_i)$$

for each predicate symbol p defined in P with a set of clauses $\{p(\overline{x}) \text{ :– } \mathrm{c_i}, \beta_\mathrm{i}\}_{1 \leq \mathrm{i} \leq \mathrm{n}}$, and where $\overline{y_i} = Var(c_i \wedge \beta_i) \setminus \overline{x}$. For each predicate symbol p appearing in P but which doesn't appear in the head of a clause, the formula $\forall \overline{x}\, \neg p(\overline{x})$ is added to the conjunction. Fitting [3] proposes to interpret these formulas in a logic with 3 truth-values t, u and f, with usual three-valued interpretations of quantifiers and connectives except for $p \Leftrightarrow q$ whose truth value is t iff p and q have the same truth value, f otherwise. This is Lukasiewicz's interpretation of logical equivalence. We recall here the truth-tables for the connectives $\neg$, $\wedge$, $\vee$ and $\Leftrightarrow$:

$\neg$	t	u	f
	f	u	t

$\wedge$	t	u	f
t	t	u	f
u	u	u	f
f	f	f	f

$\vee$	t	u	f
t	t	t	t
u	t	u	u
f	t	u	f

$\Leftrightarrow$	t	u	f
t	t	f	f
u	f	t	f
f	f	f	t

The logical meaning of a normal CLP program P is thus the following partial interpretation:

$$\mathcal{L}(P) = (\mathcal{L}^+(P), \mathcal{L}^-(P))$$

where

[1] As usual, a standard set operator $\top \in \{\cup, \cap, \ldots\}$ is defined componentwise over the Cartesian product $\mathcal{B} \times \mathcal{B}$ with $(X_1 \times X_2) \top (Y_1 \times Y_2) \equiv (X_1 \top Y_1) \times (X_2 \top Y_2)$

- $\mathcal{L}^+(P) = \{(c, A) \in \mathcal{B} \mid P^* \cup \mathcal{T} \models_3 c \Rightarrow A\}$
- $\mathcal{L}^-(P) = \{(c, A) \in \mathcal{B} \mid P^* \cup \mathcal{T} \models_3 c \Rightarrow \neg A\}$

Usually, a fixed point semantics ([3], [8]) is given, allowing one to fill the gap between the declarative meaning of P and the information operationally extracted from it (its operational semantics). Using the groundness notion defined in the beginning of this section, a *partial ground interpretation* is a pair $I = < I^+, I^- >$ such that $I^+, I^- \subseteq [\mathcal{B}]$ and $I^+ \cap I^- = \emptyset$ (consistency condition). A ground atom A is $true$ (resp. $false$) in I iff $A \in I^+$ (resp. $A \in I^-$). A constraint c is $true$ in I iff c is $true$ in $\mathcal{A}$. It is nice to denote $I(A)$ the truth value of an atom A w.r.t. I : so $I(A) = t$ (resp. f) iff $A \in I^+$ (resp. I^-), u otherwise.

The set of partial ground interpretations forms a semi-lattice for set inclusion on true and false atoms : the bottom element is $I = < \emptyset, \emptyset >$, often denoted $\emptyset$, where all things are undefined. Given a first order formula F, built over $\Pi_c \cup \Pi_p$, F is true in I, denoted $I \models_3 F$ iff it is $true$ under the strong three-valued interpretation of the logical symbols as presented in the above truth-tables. Using these notions, it becomes possible to give a ground fixed point semantics [3, 8, 16] to a CLP program.

Definition 1. *Given P a normal CLP program, the immediate consequence operator Φ_P is defined by :*

$$\Phi^+{}_P(I) = \{A \in [\mathcal{B}] \mid \textit{there exists a clause } p(\overline{x}) \text{ :- c}, \alpha \textit{ in } \mathrm{P} \textit{ and}$$
$$\textit{a valuation } v \textit{ such that } v(p(\overline{x})) = A \textit{ and } I, v \models_3 c \wedge \alpha\}$$
$$\Phi^-{}_P(I) = \{A \in [\mathcal{B}] \mid \textit{for any clause } p(\overline{x}) \text{ :- c}, \alpha \textit{ in } \mathrm{P} \textit{ and}$$
$$\textit{for any valuation } v \textit{ such that } v(p(\overline{x})) = A \textit{ then } I, v \models_3 \neg(c \wedge \alpha)\}$$

Starting from the fact that Φ_P is monotonic (but not continuous) over the lattice of ground interpretations and following the Knaster-Tarski theorem, it admits a least fixed point which is the least three-valued $\mathcal{A}$-model of $P^* \cup \mathcal{T}$. The ordinal powers of Φ_P are defined as follows :

Definition 2. *[3]*

$$\Phi_P \uparrow 0 = \emptyset,$$
$$\Phi_P \uparrow n + 1 = \Phi_P(\Phi_P \uparrow n) \textit{ if } n \in \mathbb{N},$$
$$\Phi_P \uparrow \alpha = lub\{\Phi_P \uparrow \beta \mid \beta < \alpha\} \textit{ if } \alpha \textit{ is a limit ordinal.}$$

The following result ([16]) states the relationship between these ordinal powers and the logical semantics previously defined.

Theorem 1. *Given P a CLP program and F a formula, then $P^* \cup \mathcal{T} \models F$ iff $\forall F$ is true in $\Phi_P \uparrow n$ for some integer n.*

As a corollary ([3], [8]), we have : $[\mathcal{L}(P)] = \Phi_P \uparrow \omega$. This exactly means that $[\mathcal{L}(P)]$ is an adequate definition of the ground things we can hope to cover by a constraint logic program. Let us now turn to constraint inductive logic programming.

3 Constraint ILP setting

As stressed in our introduction, there are now a lot of attempts to integrate constraints in standard ILP process. So it is necessary to reformulate the ILP problem using a constrained framework. This is the object of this section.

3.1 Concepts, examples

The target concept C is now described as a partial constrained interpretation $< C^+, C^- >$. Let us take a simple example in the real numbers domain. Suppose we want to learn invertibility of real number : in a logic setting, we have to produce a definition for a unary predicate $p =_{def} has-an-inverse/1$. Necessarily, and this is the advantage of dealing with constraints, the ILP machine should accept not only things of the form $p(r)$ with a ground value r, but also some pieces of informations e as $([\![3 < x < 10]\!],\ p(x))$. It should be noted that e.g. Progol ([12]) can use constraints as examples and would accept as positive example $p(x) : -x > 3$ as well as negative one like : $-p(x), x < -2$ (using Progol syntax). In our syntax, these two examples would be rewritten as $(x > 3, p(x))$ with t truth value and $(x < -2, p(x))$ with f truth value. Now what is the meaning of "e is an example for C" ? It is not so simple as in the standard case and it is not realistic to assume a simple set relationship $e \in C$. Back to our toy example, we can consider $\{([\![x > 0]\!], p(x)), ([\![x < 0]\!], p(x))\}$ as a good definition of our concept. Then we come back to a standard notion of *covering* that we recall. Given two constraint atoms (c, A) and (d, A), we have a natural partial pre-ordering over $\mathcal{B}$ by putting :

$$(c, A) \sqsubseteq (d, A) \text{ iff } \mathcal{T} \models (c \Rightarrow d).$$

Note that, since $\mathcal{T}$ is assumed to be a complete theory, $(c, A) \sqsubseteq (d, A)$ is equivalent to $[(c, A)]_{\mathcal{A}} \subseteq [(d, A)]_{\mathcal{A}}$ (see section 2.1). This pre-ordering is the *covering pre-ordering* and could be easily extended to subsets of $\mathcal{B}$ in the *strong covering*:

Definition 3. *Given I and J, two subsets of $\mathcal{B}$: $I \sqsubseteq J$ iff $\forall\ (c, A) \in I, \exists\ (d, A) \in J,\ (c, A) \sqsubseteq (d, A)$.*

This covering extends to partial interpretations by :

Definition 4. *Given I and J two partial interpretations : $I \sqsubseteq J$ iff $I^+ \sqsubseteq J^+$ and $I^- \sqsubseteq J^-$.*

An example e would be a constraint atom covered by the target concept : this definition allows to consider $([\![3 < x < 10]\!], p(x))$ as an example for $\{([\![x >

$0]\!], p(x)), ([\![x < 0]\!], p(x))\}$. Unfortunately, this is not sufficient for $([\![-3 < x < 10, x \neq 0]\!], p(x))$ which is also a good candidate to be an example. So, it becomes clear that we need the *finite covering* notion :

Definition 5. *Given I and J, two subsets of $\mathcal{B}$: $I \sqsubseteq_f J$ iff*
$\forall\ (c, A) \in I, \exists\ (d_1, A), \cdots, (d_n, A) \in J$ such that $\mathcal{T} \models (c \Rightarrow \bigvee_{i=1}^{n} d_i)$.

The finite covering notion extends to partial interpretations in the same manner as strong covering. Now we can restate the notion of example :

Definition 6. *e is an example of a given concept C if and only if $e \sqsubseteq_f C$. e is a positive (resp. negative) example iff $e \sqsubseteq_f C^+$ (resp. $e \sqsubseteq_f C^-$).*

With that definition, we capture our toy example. This definition is equivalent to the next one : e is a positive (resp. negative) example of a given concept C if and only if $[e]_{\mathcal{A}} \subset [C^+]_{\mathcal{A}}$ (resp. $[C^-]_{\mathcal{A}}$) Now, to learn a unknown concept C, the ILP machine accesses to :

i) a finite proper interpretation $S = S^+ \cup S^-$ (the training set in Instance Based Learning terminology) such that $S \sqsubseteq_f C$: S^+ are the positive examples i.e. the things known as being true, S^- the negative ones i.e. the things known as being false. Of course, S is supposed to satisfy the consistency requirement of a partial interpretation. In some case, ILP machinery does not accept negative examples and so necessarily S^- is empty.

ii) a basic program usually denoted B (as background knowledge) representing a basic knowledge we have concerning the concept to learn : B is a constraint logic program insufficient to explain our observables S.

It is not realistic to hope that the ILP machines exactly outputs a constrained logic program H such that $\mathcal{L}(H \cup B) = C$. This would almost be a syntactic equality. We have only to insure a weaker but sufficient property : $\mathcal{L}(H \cup B) \sqsubseteq_f B$ and $C \sqsubseteq_f \mathcal{L}(H \cup B)$. This property is equivalent to :

$$[\mathcal{L}(H \cup B)]_{\mathcal{A}}\ (= \Phi_{H \cup B} \uparrow \omega) = [C]_{\mathcal{A}}$$

In that case, we say that H is *correct* with regard to C. We may notice that, when dealing with equality constraints over Herbrand universe, using the Clark equality theory as underlying theory, we simply get the standard unconstrained framework.

3.2 *Identification in the limit* revisited

We may now revisit the notion of [5] to fit with our framework. So we suppose to be given an infinite increasing sequence S_i, $i \in I\!N^*$ of training sets such that for each i, $S_i \sqsubseteq_f C$ where C is the unknown target concept. Of course, each S_i is supposed to be constituted with a finite set of constraint atoms, and more precisely $\#S_1 = 1$ and $\#S_{n+1} = \#S_n + 1$. Each example is supposed to be given with its truth-value (t or f) with regard to C. This sequence of training examples is provided to an ILP machine A which is supposed to produce for each input S_i (and B as background knowledge) a constraint logic program $H_i = A(S_i, B)$ such that $[S_i]_{\mathcal{A}} = [\mathcal{L}(H_i \cup B)]_{\mathcal{A}}$, i.e. H_i is correct with respect to S_i. The very meaning we are interested in is $[C]_{\mathcal{A}}$. So, we can hope to get an hypothesis with the same ground semantics and the following definition is well-suited to our aim :

Definition 7. *The target concept C is identified in the limit by the ILP algorithm A if and only if*

$$lim_{n\infty}[A(S_i, B) \cup B]_{\mathcal{A}} = lim_{n\infty}[H_i \cup B]_{\mathcal{A}} = [C]_{\mathcal{A}}$$

This is not the standard definition where one needs to exactly obtain $[C]_{\mathcal{A}}$ for a given finite n. There is no ambiguity concerning the lim operator since we handle partial ground interpretations and lim has to be understood as the least upper bound (with regard to interpretation inclusion) of an increasing family of interpretations : $lim_{n\infty}$ is a more familiar notation than $lub_{n<\omega}$. If the previous equality holds, then we can get H by building step by step programs H_i over finite training sets. Of course, if the process is infinite, then, in finite time, we shall only get an approximation H of C and it is important to measure how $fine$ is H with respect to C. Thus distances are generally nice tools for such a measure. So our main task now is to build distances over the set of constraint interpretations.

4 A distance over partial ground interpretations

One may follow a standard approach and work over the $\mathcal{A}$-base $\mathcal{B}_{\mathcal{A}}$. So inspired by the work of [4], we can put :

Definition 8. *A level function is a mapping $||$ from $\mathcal{B}_{\mathcal{A}}$ to $I\!N$. Given an $\mathcal{A}$-atom A, $\mid A \mid$ is the level of A.*

Now, we can define the following real-valued mapping :

Definition 9. *Given I and J, two partial ground interpretations,*

$$d(I, J) =_{def} \text{ if } I = J \text{ then } 0 \text{ else } \frac{1}{2^n}$$

where $n = min\{\mid A \mid \mid\ I(A) \neq J(A)\}$.

In fact n is the first level where I and J disagree. We may notice that this underlying notion of disagreement is rather smooth because undefined values are also considered. For instance, if $I(A) = t$ and $J(A) = u$, I and J disagree over A in the previous sense. One can see [14] where a careful examination of different notions of disagreement is done. Following the lines of [4], we prove :

Proposition 1. *d is a distance over $2^{\mathcal{B}_\mathcal{A}} \times 2^{\mathcal{B}_\mathcal{A}}$ and the metric space $(2^{\mathcal{B}_\mathcal{A}}, d)$ is complete.*

4.1 Relationship with the lattice-theoretic semantics

Before to go on, let us examine the relationship between a level issued distance d and the limit equality of section 2.2 : $[\mathcal{L}(P)] = \Phi_P \uparrow \omega$. In the simple case of [13], it has been proved :

Proposition 2. *([13]) Given a level mapping $||$ such that for every n the set of atoms A such that $| A |= n$ is finite, then $lim_{n\infty} d(\Phi_P \uparrow n, [\mathcal{L}(P)]) = 0$ where d is the distance induced from $||$.*

Roughly speaking, the distance induced by a level function well-behaves with regard to lattice theoretic limit. In the case of [13], we are over the Herbrand base and $[\mathcal{L}(P)]$ is just the least Herband model of P. The proof of the previous property uses as basic assumption the fact that the Herband base is denumerable and we can divide it with an infinite number of finite slices : each slice being constituted with level equivalent atoms. In our case, the $\mathcal{A}$-base $\mathcal{B}_\mathcal{A}$ is in general not denumerable (think for instance to the case $\mathcal{A} = I\!R$) and we can not hope to get such a denumerable decomposition in finite subsets. That is why we consider a special class of programs for which the property will hold.

Definition 10. *Given a level mapping $||$, a program P is strictly recurrent w.r.t. $||$ if and only if for each $\mathcal{A}$-atom A in $\Phi_P \uparrow m \setminus \Phi_P \uparrow n$ $(m > n)$, $| A |>| B |$ for each $\mathcal{A}$-atom B in $\Phi_P \uparrow n$.*

Thus we can prove :

Proposition 3. *Given a level function $||$, if P is strictly recurrent w.r.t. $||$, then we have $lim_{n\infty} d(\Phi_P \uparrow n, [\mathcal{L}(P)]) = 0$ where d is the distance induced from $||$.*

Proof. : We consider the increasing sequence $\Phi_P \uparrow 0 = \emptyset, \Phi_P \uparrow 1, \ldots, \Phi_P \uparrow n \ldots$. If this sequence is stationary at a given step n i.e. $\Phi_P \uparrow n = \Phi_P \uparrow n+1$, so we are done since in that case all the $\Phi_P \uparrow i$'s are now equal and the sequence $d(\Phi_P \uparrow i, [\mathcal{L}(P)])$ is stationary and equal to 0.
The only interesting case is when $\Phi_P \uparrow 0 \subset \Phi_P \uparrow 1 \subset \ldots \subset \Phi_P \uparrow n \ldots$ This implies, for each n there exists an A_n such that $A_n \in [\mathcal{L}(P)] \setminus \Phi_P \uparrow n$ and $d([\mathcal{L}(P)], \Phi_P \uparrow n) = \frac{1}{2^{|A_n|}}$.
Now, given an atom A_n of the sequence, we can consider the least integer $m > n$ such that $A_n \in \Phi_P \uparrow m$. By definition, $A_m \notin \Phi_P \uparrow m$ and thus $A_m \in \Phi_P \uparrow l$ for a given $l > m$ since $[\mathcal{L}(P)] = lim_{n\infty} \Phi_P \uparrow n$. Since P is strictly recurrent w.r.t $||$, we have $| A_m |>| A_n |$ and so $d([\mathcal{L}(P)], \Phi_P \uparrow m) = \frac{1}{2^{|A_m|}} < \frac{1}{2^{|A_n|}} = d([\mathcal{L}(P)], \Phi_P \uparrow n)$. This means we have a sub-sequence of $\{A_n \mid n \in I\!N\}$, $\{A_{\alpha_0}, \ldots, A_{\alpha_n}, \ldots\}$ such that $| A_{\alpha_0} |<| A_{\alpha_1} |< \ldots <| A_{\alpha_n} |< \ldots$ Hence the sequence $\{\frac{1}{2^{|A_{\alpha_n}|}} \mid n \in I\!N\}$ converges to 0 and we are done. □

We may notice that we do not need the level function to be defined over the whole set $\mathcal{B}_{\mathcal{A}}$ but only over $[\mathcal{L}(P)]$. One can ask for the existence of strictly recurrent programs w.r.t. a level function. There is a natural level mapping for which P is strictly recurrent.

Definition 11. *Given a constraint program P and an $\mathcal{A}$-atom $A \in [\mathcal{L}(P)]$ we put*

$$ag(A, P) = min\{n \in I\!N \mid A \in \Phi_P \uparrow n\}$$

So we get a partial level function by putting $|A|_P = ag(A, P)$: $ag(A, P)$ is only defined over $[\mathcal{L}(P)]$. An important difference with previous works concerning level functions is the fact that our level is not issued from syntactic observations : we abstract over the underlying representation language and so we capture a little bit more of the semantics of the given object. We have the immediate following result :

Proposition 4. *P is strictly recurrent w.r.t. $||_P$.*

Proof. : for each $\mathcal{A}$-atom A in $\Phi_P \uparrow m \setminus \Phi_P \uparrow n$, $ag(A, P) > n$ but for each $B \in \Phi_P \uparrow n$, $ag(B, P) \leq n$. And we get the desired result. □

So, $||_P$ is a natural level mapping associated to a given program that renders it recurrent and we deduce :

Corollary 1. *For each constraint program P, there exists a distance over the set of ground partial interpretations such that $lim_{n\infty} d(\Phi_P \uparrow n, [\mathcal{L}(P)]) = 0$.*

At this step, we may notice that we have a result similar to [3] (but for different aims) where the distance for which a program well-behaves is program-dependent. In [13], the distance is defined independently of programs. It is not sure that such a possibility exists for constraint programs.

4.2 Continuity of Φ_P

Now, one can ask about the continuity of the Φ_P operator associated to a given P with respect to the topology induced by a level-issued distance d. In fact, the previous equality does not insure the continuity of Φ_P. The continuity is defined as usual : given a fixed I,

$$\forall \epsilon \in I\!R, \exists \eta \in I\!R, d(I, J) < \eta \Rightarrow d(\Phi_P(I), \Phi_P(J)) < \epsilon$$

Inspired by [13], we need to extend here to the constraint case the definition of a recurrent program ([1]) :

Definition 12. *Given a level mapping $||$, a program P is recurrent w.r.t. $||$ if and only if for each ground clause $A \leftarrow \alpha$, the level of A is strictly greater than the level of the atoms in α.*

The level function is "increasing" from right to left along a single clause. We may notice that there is no relationship between recurrent and strictly recurrent properties : "to be recurrent" is a local property since "to be strictly recurrent" is a global one involving the full operational process. As in [13], we have thus the following property :

Proposition 5. *Given* $||$*, a level function and* P *a constraint recurrent program w.r.t.* $||$*,* I *and* J *two ground partial interpretations and* d *the distance induced by* $||$*. Then* $d(\Phi_P(I), \Phi_P(J)) \leq d(I, J)$*.*

Proof. : if $d(\Phi_P(I), \Phi_P(J)) = 0$, we are done. So the only interesting case is when $d(\Phi_P(I), \Phi_P(J)) = \frac{1}{2^{|A|}}$ where $A \in \Phi_P(I) \setminus \Phi_P(J)$ for instance. We have two possibilities :
Either $A \in \Phi_P(I)^+ \setminus \Phi_P(J)^+$: then there exists a ground clause $A \leftarrow \alpha$ such that α is t in I and is not t in J. This means there exists an atom $B \in \alpha$ such that $B \in I \setminus J$. Thus I and J disagree on B and so $d(I, J) \geq \frac{1}{2^{|B|}}$ which is greater than $\frac{1}{2^{|A|}} = d(\Phi_P(I), \Phi_P(J))$ since P is recurrent.
Or $A \in \Phi_P(I)^- \setminus \Phi_P(J)^-$: then for each ground clause $A \leftarrow \alpha$, α is f in I and this is not the case for J. So, among these clauses $A \leftarrow \alpha$, there exists one $A \leftarrow \alpha_0$ such that α_0 is not f in J. This means there is in α_0 an atom where I and J disagree. The above reasoning applies and we have the expected result. □

Corollary 2. *If* P *is a constraint recurrent program w.r.t.* $||$ *then* Φ_P *is a (uniformly) continuous mapping w.r.t. the distance induced by* $||$*.*

Proof. : this fact since a function satisfying the previous inequality is uniformly continuous. □

We have now identify some situations where a natural distance definition is possible and the given distance well-behaves, for some program classes, with standard lattice-theoretic limit equalities. With our definitions and restrictions, we are now in the situation of [13]. The next section examines if we can do more.

5 Relationship with identification in the limit

And now, what about the relationship between our distance d and the identification in the limit notion ? Back to the work of [13], the identification in the limit fits with a distance based over a level function such that *for a given level, there is only a finite number of atoms with this level* : so, it is possible to provide finite sets of examples to an ILP machine using an "increasing level" strategy and to cover the whole Herbrand base. [13] exhibits a candidate function for defining level of a given ground atom : the depth of its representing tree. Doing so, we insure that there are only a finite number of trees with a given depth as soon as the language is finite. For each atom, this level is finitely computable and we can apply the previous strategy. To be precise, with these assumptions, one can prove that, when providing to an ILP machine, a sequence $\{e_i \mid i \in I\!N\}$ of examples such that if $n < m$ then $\mid e_n \mid \leq \mid e_m \mid$, this machine outputs a sequence of programs P_n covering $S_n = \{e_i \mid i \leq n\}$ such that $lim_{n\infty} d(C, \mathcal{L}(P_n \cup B)) = 0$.

In the constraint framework, the notion of tree representation is not so clear because of the evaluation of the function symbols leading to identify $1 + 1$ and 2 ! So, as previously noticed, we can not hope such a nice situation. Let us examine a possibility to get a kind of identification-in-the-limit agreement. The main notion we are lacking in $\mathcal{B}_{\mathcal{A}}$ is denumerability : so back to $\mathcal{B}$. Recall that $\mathcal{B}$ is the full set of constraint atoms and is necessarily denumerable as a product of denumerable sets : the set of formula and the set of atoms. As claimed in the beginning of section 2, constraints atoms are a more natural choice of observables than the standard success set as defined in [9]. That is why the very meaning of a constraint logic program is actually a non-ground meaning. So what about a distance over non-ground semantics i.e. over the set $\mathcal{I} \subset 2^{\mathcal{B}} \times 2^{\mathcal{B}}$ (to follow the notation of section 2) ? Starting from our level-issued distance d, we induce a real-valued mapping d' from $\mathcal{I} \times \mathcal{I}$ by putting $d'(I, J) = d([I]_{\mathcal{A}}, [J]_{\mathcal{A}})$.

Proposition 6. *d' is a quasi-distance over the set $\mathcal{I}$ of partial constrained interpretations.*

Proof. : The only property of a full distance which is not satisfied is the fact that $d'(I, J) = 0$ does not imply $I = J$. And this is a consequence of the fact that $[I]_{\mathcal{A}} = [J]_{\mathcal{A}}$ does not imply $I = J$ but only the two following relations : $I \sqsubseteq J$ and $J \sqsubseteq I$. □

We assume now a level-mapping $\mid \; \mid$ and the corresponding distances d and d' to be given. A concept C is just a consistent subset of $2^{\mathcal{B}} \times 2^{\mathcal{B}}$ i.e. an element of $\mathcal{I}$. Let be now a sequence $\{e_i \mid i \in I\!N\}$ of examples for C given with their corresponding truth value t or f. Each e_i is a pair (c_i, A_i) such that $e_i \sqsubseteq_f C$. Let us assume that the sequence satisfies the following property :

$$i < j \Rightarrow \; \forall a \in [e_i]_{\mathcal{A}}, \; \forall a' \in [e_j]_{\mathcal{A}}, \; \mid a \mid < \mid a' \mid \tag{1}$$

This is a kind of increasing-level sequence. We consider now the following sequence of partial constraint interpretations $S_n =< S_n^+, S_n^- >$ defined as :

$S_0 =$ if e_0 is t then $< \{e_0\}, \emptyset >$ else $< \emptyset, \{e_0\} >$

$S_{n+1} =$ if e_{n+1} is t then $< S_n^+ \cup \{e_{n+1}\}, S_n^- >$ else $< S_n^+, S_n^- \cup \{e_{n+1}\} >$

Since C is denumerable, it is realistic to cover C with such a sequence. The previous sequence enjoys the following properties :

$S_n \subset S_{n+1}$ i.e. $S_n^+ \subset S_{n+1}^+$ and $S_n^- \subset S_{n+1}^-$, $\#S_{n+1} = \#S_n + 1$,

$\forall a \in [e_{n+1}]_{\mathcal{A}}, \forall e \in S_n, \forall a' \in [e]_{\mathcal{A}}, \mid a' \mid < \mid a \mid$.

$lim_{n\infty} S_n \sqsubseteq_f C$ since each element e_i satisfies $e_i \sqsubseteq_f C$

We can then prove the following result :

Proposition 7. *Suppose $C \sqsubseteq_f lim_{n\infty} S_n$. Let us consider an ILP machine A providing for S_n a covering program $P_n = A(S_n, B)$ i.e. such that $[\mathcal{L}(P_n \cup B)]_{\mathcal{A}} = [S_n]_{\mathcal{A}}$ where B is the background knowledge. Then we have :*

$$lim_{n\infty} d'(C, \mathcal{L}(P_n \cup B)) = 0.$$

Proof. : if the previous equality does not hold, there exists k such that $\forall n \in I\!N, \exists n_0 > n$ such that $d'(C, \mathcal{L}(P_{n_0} \cup B)) > \frac{1}{2^k}$ i.e. $d([C]_{\mathcal{A}}, [\mathcal{L}(P_{n_0} \cup B)]_{\mathcal{A}}) > \frac{1}{2^k}$ i.e. $d([C]_{\mathcal{A}}, [S_{n_0}]_{\mathcal{A}}) > \frac{1}{2^k}$ since $[\mathcal{L}(P_{n_0} \cup B)]_{\mathcal{A}} = [S_{n_0}]_{\mathcal{A}}$ by hypothesis concerning our machine A. So we have for each n, a constraint atom $(c_n, A_n) \in C \setminus S_{n_0}$ such that there exists a valuation v_n, satisfying c_n and such that $v_n(A_n) \notin S_{n_0}$ and $\mid v_n(A_n) \mid < k$. Let us remind that, since $C \sqsubseteq_f lim_{n\infty} S_n$, there exists a m such that $(c_n, A_n) \sqsubseteq_f S_m$. By definition of $\sqsubseteq_f$, we have a finite number of constraint atoms $\{(d_l, A_n),\ l \in [1,p]\}$ in S_m such that :

$$\mathcal{T} \models (c_n \Rightarrow \bigvee_{l=1}^{p} d_l).$$

We deduce there exists $l \in [1,p]$ such that v_n validates d_l so $v_n(A_n) \in [S_m]_{\mathcal{A}}$. Among other things, this implies $m > n_0$ since $\{S_i \mid i \in I\!N\}$ is an increasing sequence and so $\{[S_i]_{\mathcal{A}} \mid i \in I\!N\}$. Consequently, we have build a ground atom $v_n(A_n)$ such that $v_n(A) \in [S_m]_{\mathcal{A}}$ but $v_n(A_n) \notin [S_n]_{\mathcal{A}}$. This is a property of the S_i's that necessarily $\mid v_n(A_n) \mid > n$. Since this construction is valid for each n, it suffices to chose $n > k$ to get a contradiction with a ground atom $v_n(A_n)$ satisfying in the same time $\mid v_n(A_n) \mid > n > k$ and $\mid v_n(A_n) \mid < k$. So we have the desired result. □

We can slightly relax the property (1) into :

$$\forall i, \exists j \text{ such that } i > j \text{ and } \forall a \in [e_i]_{\mathcal{A}},\ \forall a' \in [e_j]_{\mathcal{A}},\ \mid a \mid < \mid a' \mid \qquad (2)$$

It is clear that the previous result is only a restricted translation of the result of [13] because of the conditions (1) or (2). These two conditions are naturally satisfied with Herbrand base and the depth of the terms as level mapping. Nevertheless, as soon as we investigate the scope of constraints, we are led to limitations due to the non denumerability of the given domain or the lack of denumerable decomposition in finite slices. So, to get a more powerful result, it would be probably necessary to put some more restrictions over the class of target programs or over the underlying domain $\mathcal{A}$ of constraints. Back to the notion of non-ground semantics, and considering a concept $C =< \{([\![x \neq 0]\!], p(x))\}, (\{[\![x = 0]\!], p(x))\} >$, it is clear that a learning machine giving the following program :

$$p(x) \ \text{:-}\ (\text{x} < 0)$$
$$p(x) \ \text{:-}\ (\text{x} > 0)$$

is rather satisfactory despite the fact that $\mathcal{L}(P)$ contains $([\![x = 3]\!], p(x))$ which is not in C but is only covered by C. We have also to recall that in the current version of Progol, CProgol4.4[2], non-ground constraint examples of the form $p(x) : -x > 3$ are allowed. So, from a constraint learning viewpoint, a non-ground semantics is of great interest.

6 Discussion and conclusion

Constraints are now unavoidable notions to be included in an ILP machinery. The number of works attempting to integrate constraints is an illustration of that claim. Despite this fact, we lack a formal setting establishing a relationship between semantics issues and a formal model of machine learning well-suited to ILP, namely the identification-in-the-limit model. In this paper, we provide such tools. The main idea, inspired by [3, 13], is to consider the underlying objects i.e. constraint atoms, as belonging to a metric space and to examine the behavior of an ILP process with respect to the given distance. So we define standard distances based over the notion of level mapping and we first show that, for some classes of programs, these distances well-behave with the lattice-theoretic semantic definitions. Then we restate the ILP framework in terms of constraint atoms and constraint programs and we give a sense to the notion of identification-in-the-limit. It is now reasonable to examine if the limit process underlying our metric definition is consistent with the notion of identifying the target concept in the limit. We show that, with some restrictions, we can get a similar result as [13]. In some sense, our work could be seen as an (incomplete) extension of the works of [13]. The main difference is that we deal with constraints and negation : so we loose some syntactic notions like depth of a tree, well-suited to pure Herbrand base. So it remains to explore the properties required from the constraint domain to insure a good adequation with identification-in-the-limit.

From a general viewpoint and as usual in the machine learning field, we are faced to the fact that C is unknown. So, a distance as $d(H, C)$, where H is the hypothesis outputed by an ILP machine to approximate a target concept C is not formally computable. But this distance measures, in some sense, the quality of H with respect to C. A solution would be to find computable upper bounds for such distances to insure a good learning process. In the PAC model ([17], such upper bounds are exhibited using the Vapnik-Chervonenkis dimension [18]) to bound sample complexity for instance. It remains to establish this kind of relationship between the VC dimension of some classes of program and the given distance. On another side, even if we get such bounds, it remains to design suitable algorithms minimizing these bounds and then insuring the quality of the hypothesis. As far as we know, these are open problems.

[2] available on **http://www.cs.york.ac.uk/mlg/progol.html** where a full and clear description is given.

References

1. M. Bezem. Characterizing termination of logic programs with level mappings. In MIT Press, editor, *Proc. of the North American Conf. on Logic Programming*, pages 69–80, 1989.
2. L. de Raedt and V. Van Laer. Inductive constraint logic. In *Proc. 6th Intern. Workshop on Algorithmic Learning Theory*, pages 80–94. Springer Verlag, 1995.
3. M. Fitting. A Kripke-Kleene Semantics for Logic Programs. *Journal of Logic Programming*, 4:295–312, 1985.
4. M. Fitting. Metric Methods, Three Examples and a Theorem. *Journal of Logic Programming*, 21:113–127, 1994.
5. I. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
6. J. Jaffar and J.L. Lassez. Constraint Logic Programming. In *Proceedings of Conference on Principles of Programming Languages*, pages 111–119, 1987.
7. T. Kawamura and K. Furukawa. Toward inductive generalization in constraint logic programs. In *Proc. of the IJCAI workshop on Inductive Logic Programming*, 1993.
8. K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4(3):289–308, 1987.
9. M.J. Maher. A Logic Programming View of CLP. In MIT Press, editor, *Proceedings of 10th ICLP*, pages 737–753, 1993.
10. L. Martin and C. Vrain. A three-valued framework for the induction of general logic programs. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 219–235, 1995.
11. F. Misoguchi and H. Ohwada. Constraint-directed generalization for learning spatial relation. In *Proc. of the second ILP worshop ILP'92*, 1992. (ICOT Technical report TM 1182).
12. S. Muggleton. Inverse entailment and Progol. *New Gen. Comput.*, 13::245–286, 1994.
13. S. H. Nienhuys-Cheng. Distances and Limits on Herbrand Interpretations. In *Proceedings of ILP'98*, volume 1446 of LNAI, pages 250–260, 1998. Springer Verlag.
14. G. Richard. From disagreement sets to divergence levels in ILP. In *Proc. of the 6th International Symposium on Mathematics and Artificial Intelligence (AMAI'2000)*, 5-7 janvier 2000. Fort Lauderdale (USA).
15. M. Sebag and C. Rouveirol. Constraint inductive logic programming. In Raedt de L., editor, *5th International Workshop on Inductive Logic Programming (ILP'95)*. Springer-Verlag, September 1995. Katholieke Universiteit Leuven Technical Report.
16. P. Stuckey. Constructive Negation for Constraint Logic Programming. In *Proceedings of 6th Symposium on LICS'91*, pages 359–373. IEEE Computer Society Press, 1991.
17. L. G. Valiant. A Theory of the Learnable. In *Communications of the ACM*, pages 1134 – 1142, 1984.
18. V. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Springer Verlag, 1982.