Generalizing Programs via Subsumption

Miguel A. Gutiérrez-Naranjo, José A. Alonso-Jiménez, and
Joaquín Borrego-Díaz

Dept. of Computer Science and Artificial Intelligence
University of Seville
{magutier,jalonso,jborrego}@us.es

Abstract. In this paper we present a class of operators for Machine Learning based on Logic Programming which represents a *characterization* of the subsumption relation in the following sense: The clause C_1 subsumes the clause C_2 iff C_1 can be reached from C_2 by applying these operators. We give a formalization of the closeness among clauses based on these operators and an algorithm to compute it as well as a bound for a quick estimation. We extend the operator to programs and we also get a characterization of the subsumption between programs. Finally, a weak metric is presented to compute the closeness among programs based on subsumption.

1 Introduction

In a Machine Learning system based on clausal logic, the main operation lies on applying an operator to one or more clauses with the hope that the new clauses give a better classification for the training set. This generalization must fit into some relation of order on clauses or sets of clauses. The usual orders are the subsumption order, denoted by $\succeq$, and the implication order $\models$.

Subsumption was presented by G. Plotkin [9]. In his study about the lattice structure induced by this relation on the set of clauses, he proved the existence of the *least general generalization* of two clauses under subsumption and defined the *least generalization under relative subsumption*. Both techniques are the basis of successful learning systems on real-life problems. Later, different classes of operators on clauses, the so-called refinement operators, were studied by Shapiro [11], Laird [5] and van der Laag and Nienhuys-Cheng [13] among others. In their works, the emphasis is put on the specialization operators, which are operators such that the obtained clause is implied or subsumed by the original clause, and the generalization operators are considered the *dual* of the first ones.

In this paper we present new results and algorithms about the generalization of clauses and logic programs via subsumption. We propose *new* generalization operators for clausal learning, the Learning Operators under Subsumption which represent a *characterization* by operators of the subsumption relation between

⋆ Work partially supported by project TIC 2000-1368-C03-0 (Ministry of Science and Technology, Spain) and the project TIC-137 of the *Plan Andaluz de Investigación.*

clauses in the following sense: If C_1 and C_2 are clauses, C_1 subsumes C_2 if and only if there exists a finite sequence (a *chain*) of LOS $\{\Delta_1/x_1\}, \ldots, \{\Delta_n/x_n\}$ such that $C_1 = C_2\{\Delta_1/x_1\} \ldots \{\Delta_n/x_n\}$. If C_1 subsumes C_2, we know that the set of chains of LOS from C_2 to C_1 is not empty, but in general the set has more than one element.

The existence of a non-empty set of chains gives us the idea for a formalization of *closeness* among clauses as the length of the shortest chain from C_2 to C_1, if C_1 subsumes C_2, and infinity otherwise.

This mapping, which will be denoted by *dc*, is the algebraic expression of the subsumption order: for every pair of clauses, C_1 and C_2, C_1 subsumes C_2 if and only if $dc(C_2, C_1)$ is finite. Since the subsumption order is not symmetric, the mapping *dc* is not either. Therefore *dc* is not a metric, but a *quasi-metric*.

Finally, *dc* is computable. We give in this paper an algorithm which calculates the quasi-distance between two clauses and present a bound which allows to estimate the closeness between clauses under the hypothesis of subsumption. This algorithm and estimation provides useful tools for the design of new learning systems which use quasi metrics to compute closeness.

In the second part of the paper, we extend the study to programs. We define a new class of operators, the *composed LOS*, which act on the set of programs and they also represent a characterization of the subsumption relation between programs. Analogously to the clausal case, the minimum of the length of the chains of operators between two programs is the basis of a weak metric to quantify the closeness between programs. This weak metric has been experimentally checked and can be added to existing systems or used to design new ones.

2 Preliminaries

From now on, we will consider some fixed first-order language $\mathcal{L}$ with at least one function symbol. Var, $Term$ and Lit are, respectively, the sets of variables, terms and literals of $\mathcal{L}$. A *clause* is a finite set of literals, a *program* is a non-empty finite set of non-empty clauses, $\mathbb{C}$ is the set of all clauses and $\mathbb{P}$ is the set of all programs. A definite program is a program where each clause contains one positive and zero or more negative literals. As usual, T_P will denote the immediate consequence operator of the program P.

A *substitution* is a mapping $\theta : S \to Term$ where S is a finite set of variables such that $(\forall x \in S)[x \neq \theta(x)]$. We will use the usual notation $\theta = \{x/t : x \in S\}$, where $t = \theta(x)$, $Dom(\theta)$ for the set S and $Ran(\theta) = \cup\{Var(t) : x/t \in \theta\}$. A pair x/t is called a *binding*. If A is a set, then $|A|$ is the cardinal of A and $\mathcal{P}A$ its power set. We will denote by $|\theta|$ the number of bindings of the substitution θ. The clause C *subsumes* the clause D, $C \succeq D$, iff there exists a substitution θ such that $C\theta \subseteq D$. A *position* is a non-empty finite sequence of positive integers. Let $\mathbb{N}^+$ denote the set of all positions. If $t = f(t_1, \ldots, t_n)$ is an atom or a term, t_i is the term at position i in t and the term at position $i\hat{}u$ in t is s if s is at position u in t_i. Two positions u and v are *independent* if u is not a prefix of v and vice versa. A set of positions P is *independent* if it is a pairwise independent

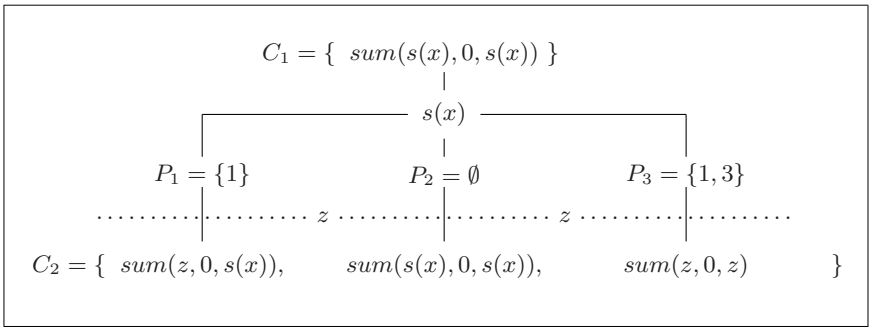

Fig. 1. Example of generalization

set of positions. The set of all positions of the term t in L will be denoted by $Pos(L, t)$. If t is a term (resp. an atom), we will denote by $t[u \leftarrow s]$ the term (resp. the atom) obtained by grafting the term s in t at position u and, if L is a literal, we will write $L[P \leftarrow s]$ for the literal obtained by grafting the term s in L at the independent set of positions P.

3 The Operators

In the generalization process, when a program P is too specific, we replace it by P' with the hope that P' covers the examples better than P. The step from P to P' is usually done by applying an operator to some clause C of P. These operators can be defined as mappings from $\mathbb{C}$ to $\mathbb{C}$, where $\mathbb{C}$ is the set of clauses of the language. Before giving the definition of the operator, we will give some intuition with an example.

Consider the one–literal clause $C_1 = \{L\}$ with $L = sum(s(x), 0, s(x))$. In order to generalize it with respect to the subsumption order, we have to obtain a new clause C_2 such that there exists a substitution θ verifying $C_2\theta \subseteq C_1$. For that, we firstly choose a term t in L, say $t = s(x)$, then we choose several subsets of $Pos(L, t)$, e.g. $P_1 = \{1\}, P_2 = \emptyset, P_3 = \{1, 3\}$ and a variable not occurring in L, say z, and finally we build the clause $C_2 = \{L[P_i \leftarrow z] \,|\, i = 1, 2, 3\} = \{sum(z, 0, s(x)), sum(s(x), 0, s(x)), sum(z, 0, z)\}$. Obviously $\theta = \{z/s(x)\}$ satisfies $C_2\theta \subseteq C_1$ (see Fig. 1). If the clause has several literals, for example, $C_1 = \{L_1, L_2, L_3\}$, with $L_1 = num(s(x))$, $L_2 = less\_than(0, s(x))$ and $L_3 = less\_than(s(x), s(s(x)))$, the operation is done with all literals simultaneously. First, the same term is chosen in every literal of C_1, say $t = s(x)$. Then, for each literal $L_i \in C_1$, some subsets of $Pos(L_i, t)$ are chosen, e.g.,$P_1^* = \{\emptyset, \{1\}\} \subseteq \mathcal{P}Pos(L_1, t)$, $P_2^* = \emptyset \subseteq \mathcal{P}Pos(L_2, t)$ and $P_3^* = \{\{1, 2{\cdot}1\}, \{1\}\} \subseteq \mathcal{P}Pos(L_3, t)$ After taking a variable which does not occur in C_1, say z, we build the sets $L_1 \xrightarrow{P_1^*} \{num(s(x)), num(z)\}$ $L_2 \xrightarrow{P_2^*} \emptyset$ and $L_3 \xrightarrow{P_3^*} \{less\_than(z, s(z)), less\_than(z, s(s(x)))\}$. C_2 is the union of these sets, i.e., $C_2 = \{num(s(x)), num(z), less\_than(z, s(z)),\ less\_than(z, s(s(x)))\}$

and $C_2\{z/s(x)\} \subseteq C_1$. In our general description, we will begin with substitutions and grafts.

Definition 1. *Let L be a literal and t a term. The set of positions P is called* compatible *with the pair $\langle L, t\rangle$ if $P \subseteq Pos(L, t)$.*

Let P^* be a set whose elements are sets of positions. Let L be a literal and t a term. P^* is called compatible *with the pair $\langle L, t\rangle$ if every element of P^* is compatible with $\langle L, t\rangle$*

For example, if $L = sum(s(x), 0, s(x))$ and $t = s(x)$, then $P_1 = \{1\}$, $P_2 = \emptyset$, $P_3 = \{1, 3\}$ are compatible with $\langle L, t\rangle$ but $P_4 = \{1{\cdot}1, 2\}$, $P_5 = \{1, 4{\cdot}3\}$ are not. If $P_1^* = \{P_1, P_2, P_3\}$ and $P_2^* = \{P_2, P_4\}$, then P_1^* is compatible with $\langle L, t\rangle$ and P_2^* is not.

The next mappings are basic in the definition of our operators. As we saw in the example, the key is to settle a set of sets of positions for each literal, all them occupied by the same term. This one is done by the following mappings.

Definition 2. *A mapping $\Delta : Lit \to \mathcal{PP}\mathbb{N}^+$ is an* assignment *if there exists a term t such that, for every literal L, $\Delta(L)$ is compatible with the pair $\langle L, t\rangle$.*

Note that the term t does *not* have to be unique, for example, consider the *identity* assignment $(\forall L \in Lit)[\Delta(L) = \{\emptyset\}]$, the *empty* assignment $(\forall L \in Lit)[\Delta(L) = \emptyset]$ or any mixture of both.

The assignments map a literal into a set of sets of positions. Each element of this set of positions will produce a literal, and the positions are the places where the new term is grafted. If $\Delta : Lit \to \mathcal{PP}\mathbb{N}^+$ is an assignment of positions and s is a term, we will denote by $L\{\Delta(L)/s\}$ the set of literals, one for each element $P \in \Delta(L)$, obtained by grafting s in L at P. Formally $L\{\Delta(L)/s\} = \{L[P \leftarrow s] \,|\, P \in \Delta(L)\}$ For example, if $L = sum(s(x), 0, s(x))$, z is a variable, P_1^* is taken from the above example and Δ is an assignment such that $\Delta(L) = P_1^*$ then $L\{\Delta(L)/z\} = \{L[P \leftarrow z] \,|\, P \in \Delta(L)\} = \{L[P \leftarrow z] \,|\, P \in P_1^*\} = \{L[P_1 \leftarrow z]\}, L[P_2 \leftarrow z], L[P_3 \leftarrow z]\} = \{sum(z, 0, s(x)), sum(s(x), 0, s(x)), sum(z, 0, z)\}$ We can now define our Learning Operators under Subsumption[1].

Definition 3. *Let Δ be an assignment and x a variable. The mapping*

$$\begin{aligned} \{\Delta/x\} : \mathbb{C} &\longrightarrow \mathbb{C} \\ C &\mapsto C\{\Delta/x\} = \textstyle\bigcup_{L \in C} L\{\Delta(L)/x\} \end{aligned}$$

is a Learning Operator under Subsumption (LOS) *if for all literal L, if $\Delta(L) \neq \emptyset$ then $x \notin Var(L)$.*

Turning back to a previous example, if $C = \{L_1, L_2, L_3\}$, with $L_1 = num(s(x))$, $L_2 = less\_than(0, s(x))$, $L_3 = less\_than(s(x), s(s(x)))$, and the assignment

$$\Delta(L) = \begin{cases} P_1^* = \{\,\emptyset, \{1\}\,\} & \text{if } L = L_1 \\ P_2^* = \emptyset & \text{if } L = L_2 \\ P_3^* = \{\,\{1, 2{\cdot}1\}, \{1\}\,\} & \text{if } L = L_3 \\ \emptyset & \text{otherwise} \end{cases}$$

[1] A preliminary version of these operators appeared in [2].

and considering z as the variable to be grafted, then $C\{\Delta/z\} = \{num(s(x)), num(z), less\_than(z, s(z)), less\_than(z, s(s(x)))\}$. These operators allow us to generalize a given clause and go up in the subsumption order on clauses as we see in the next theorem.

Proposition 1. *Let C be a clause and $\{\Delta/x\}$ a LOS. Then $C\{\Delta/x\} \succeq C$.*

The LOS define an operational definition of the subsumption relation. The last result states one way of the implication. The next one claims that all the learning based on subsumption of clauses can be carried out only by applying LOS.

Theorem 1. *Let C_1 and C_2 be two clauses such that $C_1 \succeq C_2$. Then there exists a finite sequence (a* chain*) $\{\Delta_1/x_1\}, \ldots, \{\Delta_n/x_n\}$ of LOS such that*

$$C_1 = C_2\{\Delta_1/x_1\} \ldots \{\Delta_n/x_n\}$$

For example, if we consider $C_1 = \{p(x_1, x_2)\}$ and $C_2 = \{p(x_2, f(x_1)), p(x_1, a)\}$ and the substitution $\theta = \{x_1/x_2, x_2/f(x_1)\}$. Then $C_1\theta \subseteq C_2$ holds and therefore $C_1 \succeq C_2$. Decomposing θ we can get $\sigma_1 = \{x_2/x_3\}$, $\sigma_2 = \{x_1/x_2\}$, $\sigma_3 = \{x_3/f(x_1)\}$ and $C_1\sigma_1\sigma_2\sigma_3 \subseteq C_2$ holds. Hence, considering the assignments

$$\begin{array}{lll} \Delta_1(p(x_2, f(x_1))) = \{\{2\}\} & \text{and} & \Delta_1(L) = \emptyset \text{ if } L \neq p(x_2, f(x_1)) \\ \Delta_2(p(x_2, x_3)) = \{\{1\}\} & \text{and} & \Delta_2(L) = \emptyset \text{ if } L \neq p(x_2, x_3) \\ \Delta_3(p(x_1, x_3)) = \{\{2\}\} & \text{and} & \Delta_3(L) = \emptyset \text{ if } L \neq p(x_1, x_3) \end{array}$$

we have $C_1 = C_2\{\Delta_1/x_3\}\{\Delta_2/x_1\}\{\Delta_3/x_2\}$. Note that if we take the assignment $\Delta(p(x_1, a)) = \{\{2\}\}$; $\Delta(L) = \emptyset$ if $L \neq p(x_1, a)$, then $C_1 = C_2\{\Delta/x_2\}$ also holds.

4 A Quasi-metric Based on Subsumption

The operational characterization of the subsumption relation given in the previous section gives us a natural way of formalizing the *closeness* among clauses. As we have seen, if $C_1 \succeq C_2$ then there exists *at least* one chain of LOS from C_2 to C_1 and we can consider the length of the shortest chain from C_2 to C_1. If C_1 does not subsume C_2, we will think that C_1 cannot be reached from C_2 by applying LOS, so both clauses are separated by an infinite distance.

Definition 4. *A* chain of LOS *of length n ($n \geq 0$) from the clause C_2 to the clause C_1 is a finite sequence of n LOS $\{\Delta_1/x_1\}, \{\Delta_2/x_2\}, \ldots, \{\Delta_n/x_n\}$ such that $C_1 = C_2\{\Delta_1/x_1\}\{\Delta_2/x_2\} \ldots \{\Delta_n/x_n\}$ The set of all the chains from C_2 to C_1 will be denoted by $\mathbf{L}(C_2, C_1)$ and $|\mathcal{C}|$ will denote the length of the chain $\mathcal{C}$. We define the mapping $dc : \mathbb{C} \times \mathbb{C} \to [0, +\infty]$ as follows:*

$$dc(C_2, C_1) = \begin{cases} min\{|\mathcal{C}| \,:\, \mathcal{C} \in \mathbf{L}(C_2, C_1)\} & \text{if } C_1 \succeq C_2 \\ +\infty & \text{otherwise} \end{cases}$$

The subsumption relation is not symmetric, so the mapping dc is not either. Instead of being a drawback, this property gives an algebraic characterization of the subsumption relation, since $C_1 \succeq C_2$ iff $dc(C_2, C_1) \neq +\infty$. Notice that a quasi–metric satisfies the conditions to be a metric, except for the condition of symmetry.

Definition 5. *A quasi–metric on a set X is a mapping d from $X \times X$ to the non–negative reals (possibly including $+\infty$) satisfying: (1) $(\forall x \in X)\, d(x,x) = 0$, (2) $(\forall x,y,z \in X)\, d(x,z) \leq d(x,y) + d(y,z)$ and (3) $(\forall x,y \in X)\, [d(x,y) = d(y,x) = 0 \Rightarrow x = y]$*

The next result states the computability of dc and provides an algorithm to compute it.

Theorem 2. *dc is a computable quasi–metric.*

Proof (Outline). Proving that dc is a quasi-metric is straightforward from the definition. The proof of the computability is split in several steps. Firstly, for each substitution θ we define the set of the splittings up:

$$Split(\theta) = \left\{ \sigma_1 \dots \sigma_n \,:\, \begin{array}{l} \sigma_i = \{x_i/t_i\} \quad x_i \notin Var(t_i) \\ (\forall z \in Dom(\theta))[z\theta = z\sigma_1 \dots \sigma_n] \end{array} \right\}$$

with $length(\sigma_1 \dots \sigma_n) = n$ and $weight(\theta) = min\{length(\Sigma) \,|\, \Sigma \in Split(\theta)\}$. The next equivalence holds

$$dc(C_2, C_1) = \begin{cases} 0 & \text{if } C_1 = C_2 \\ 1 & \text{if } C_1 \neq C_2 \text{ and } C_1 \subseteq C_2 \\ min\{weight(\theta) \,|\, C_1\theta \subseteq C_2\} & \text{if } C_1 \succeq C_2 \text{ and } C_1 \not\subseteq C_2 \\ +\infty & \text{if } C_1 \not\succeq C_2 \end{cases}$$

We can decide if $C_1 \succeq C_2$ and, if it holds, we can get the finite set of all θ such that $C_1\theta \subseteq C_2$, so to conclude the theorem we have to give an algorithm which computes $weight(\theta)$ for each θ. The Fig. 2 shows a non-deterministic algorithm which generates elements of $Split(\theta)$. The algorithm finishes and for all $\Sigma \in Split(\theta)$ it outputs $\Sigma^* \in Split(\theta)$ verifying $length(\Sigma^*) \leq length(\Sigma)$.

The previous theorem provides a method for computing dc, but deciding whether two clauses are related by subsumption is an NP-complete problem [1], so, from a practical point of view we need a quick estimation of the quasi-metric before deciding the subsumption. The next result settles an upper and lower bounds for the quasi-metric under the assumption of subsumption.

Theorem 3. *Let C_1 and C_2 be two clauses such that $C_1 \not\subseteq C_2$. If $C_1 \succeq C_2$ then*

$$|Var(C_1) - Var(C_2)| \leq dc(C_2, C_1) \leq min\{2 \cdot |Var(C_1)|\,,\, |Var(C_1)| + |Var(C_2)|\}$$

Proof (Outline). For each θ such that $C_1\theta \subseteq C_2$, θ has at least $|Var(C_1) - Var(C_2)|$ bindings and we need at least one LOS for each binding, hence the first inequality holds. For the second one, if $C_1\theta \subseteq C_2$ then we can find n substitutions $\sigma_1, \dots, \sigma_n$ with $\sigma_1 = \{x_i/t_i\}$ and $x_i \notin Var(t_i)$ such that $C_1\sigma_1 \dots \sigma_2 \subseteq C_2$ verifying $n = |\theta| + |Ran(\theta) \cap Dom(\theta)|$. The inequality holds since $Ran(\theta) \subseteq Var(C_2)$, $Dom(\theta) \subseteq Var(C_1)$ and $|\theta| \leq Var(C_1)$. If $C_1 = \{p(x_1, x_2)\}$, $C_2 = \{p(a,b)\}$ and $C_3 = \{p(f(x_1,x_2), f(x_2,x_1))\}$ then

$$dc(C_2, C_1) = |Var(C_1) - Var(C_2)| = 2$$
$$dc(C_3, C_1) = min\{2 \cdot |Var(C_1)|\,,\, |Var(C_1)| + |Var(C_2)|\} = 4$$

The above examples show that these bounds cannot be improved.

Input: A non-empty substitution θ
Output: An element of $Split(\theta)$
Set $\theta_0 = \theta$ and $U_0 = Dom(\theta) \cup Ran(\theta)$
Step 1:
If θ_i is the empty substitution
Then stop
Otherwise: Consider $\theta_i = \{x_1/t_1, \ldots, x_n/t_n\}$ and go to **Step 2**.
Step 2:
If there exists $x_j \in Dom(\theta_i)$ such that $x_j \notin Ran(\theta_i)$
Then for all $k \in \{1, \ldots, j-1, j+1, \ldots, n\}$ let t_k^* be a term
such that $t^k = t_k^*\{x_j/t_j\}$ Set
$\theta_{i+1} = \{x_1/t_1^*, \ldots, x_{j-1}/t_{j-1}^*, x_{j+1}/t_{j+1}^*, \ldots, x_n/t_n^*\}$
$\sigma_{i+1} = \{x_j/t_j\}$
$U_{i+1} = U_i$
set i to $i+1$ and go to **Step 1**.
Otherwise: Go to **Step 3**.
Step 3:
In this case let z_i be a variable which does not belong to U_i and set
$U_{i+1} = U_i \cup \{z_i\}$
choose $j \in \{1, \ldots, n\}$ y let T be a subterm of t_j such that T is not a variable belonging to U_{i+1}. Then, for all $k \in \{1, \ldots, n\}$ let t_k^* be a term
such that $t^k = t_k^*\{z/T\}$. Set
$\theta_{i+1} = \{x_1/t_1^*, \ldots, x_n/t_n^*\}$
$\sigma_{i+1} = \{z/T\}$
set i to $i+1$ and go to **Step 1**.

Fig. 2. Algorithm scheme to compute the subset of $Split(\theta)$

5 Programs

In this section we extend the study of subsumption to programs.

Definition 6. *The program P_2 subsumes the program P_1, $P_2 \succeq P_1$, if there exists a mapping $F : P_1 \to P_2$ such that $F(C) \succeq C$, for all $C \in P_1$.*

In the case of definite programs, the subsumption is related to the semantics via the immediate consequence operator. The proof is adapted from [7].

Proposition 2. *Let P_1 and P_2 be two definite programs. Then $P_1 \succeq P_2$ if and only if for all interpretation I, $T_{P_2}(I) \subseteq T_{P_1}(I)$*

The operators for programs are sets of pairs assignment–variable. These operators represent a characterization for the subsumption relation between programs, as we will show below.

Definition 7. *A* composed LOS *is a finite set of pairs $\Theta = \{\Delta_1/x_1, \ldots, \Delta_n/x_n\}$ where $\{\Delta_i/x_i\}$ is a LOS for all $i \in \{1, \ldots, n\}$.*

For applying a composed LOS to a program we need an auxiliary mapping which associates one LOS to each clause of the program.

Definition 8. *Let P be a program and Θ a composed LOS. An* auxiliary mapping for applying (amfa) *is a mapping $a : P \to \Theta$ such that for all clause $C \in P$, the clause $C\{a(C)\}$ is not the empty clause. The program $P_a\Theta = \{C\{a(C)\} \mid C \in P\}$ is the program obtained by applying Θ to P via the* amfa *a.*

For example, consider the program[2] $P = \{C_1, C_2, C_3\}$ with

$$C_1 = sum(0, s(s(0)), s(s(0))) \leftarrow sum(0, s(0), s(0))$$
$$C_2 = sum(s(x), s(0), s(z)) \leftarrow sum(s(0), 0, s(0)), sum(x, s(0), z)$$
$$C_3 = sum(s(y), y, s(z)) \leftarrow sum(0, y, y), sum(y, y, z)$$

and $\Theta = \{\Delta_1/x,\ \Delta_2/y,\ \Delta_3/x\}$ with

$$\Delta_1(L) = \begin{cases} \{\{2, 3\}\} & \text{if } L = sum(0, s(s(0)), s(s(0))) \\ \emptyset & \text{otherwise} \end{cases}$$

$$\Delta_2(L) = \begin{cases} \{\{2\}\} & \text{if } L \in \left\{ \begin{array}{l} sum(s(x), s(0), s(z)) \\ \neg sum(x, s(0), z) \end{array} \right\} \\ \emptyset & \text{otherwise} \end{cases}$$

$$\Delta_3(L) = \begin{cases} \{\{1 \cdot 1\}\} & \text{if } L = sum(s(y), y, s(z)) \\ \{\{1\}\} & \text{if } L = \neg sum(y, y, z) \\ \emptyset & \text{otherwise} \end{cases}$$

Consider the *amfa* $a : P \to \Theta$ such that $a(C_1) = \Delta_1/x$, $a(C_2) = \Delta_2/y$, $a(C_3) = \Delta_3/x$. Then

$$C_1\{a(C_1)\} = C_1\{\Delta_1/x\} \equiv sum(0, x, x) \leftarrow$$
$$C_2\{a(C_2)\} = C_2\{\Delta_2/y\} \equiv sum(s(x), y, s(z)) \leftarrow sum(x, y, z)$$
$$C_3\{a(C_2)\} = C_3\{\Delta_3/x\} \equiv sum(s(x), y, s(z)) \leftarrow sum(x, y, z)$$

Therefore

$$P_a\Theta = \left\{ \begin{array}{l} sum(0, x, x) \leftarrow \\ sum(s(x), y, s(z)) \leftarrow sum(x, y, z) \end{array} \right\}$$

The composed LOS also represent an operational characterization of the subsumption relation among programs. The main results are theorems 4 and 5.

Theorem 4. *Let P_1 and P_2 be two programs and Θ a composed LOS. If $P_2\theta \subseteq P_1$ then $P_1 \succeq P_2$, where Θ is applied to P via an appropriate* amfa.

The following corollary is immediate.

Corollary 1. *Let P_1 and P_2 be two programs and $\Theta_1 \ldots \Theta_n$ a finite chain of composed LOS. If $P_2\Theta_1 \ldots \Theta_n \subseteq P_1$ then $P_1 \succeq P_2$, where each Θ is applied via an appropriate* amfa *a_i.*

The next result is the converse of the corollary 1.

[2] We use the Prolog notation $A \leftarrow B_1, \ldots, B_n$ instead of $\{A, \neg B_1, \ldots, \neg B_n\}$

Theorem 5. *Let P_1 and P_2 be two programs. If $P_1 \succeq P_2$ then there exists a finite chain of composed LOS $\Theta_1 \dots \Theta_n$ such that $P_2\Theta_1 \dots \Theta_n \subseteq P_1$, where each Θ is applied via an appropriate* amfa *a_i.*

The proof of this theorem can be obtained straightforwardly from the theorem 1. If $P_1 \succeq P_2$ then for each $C \in P_2$ there exists $D \in P_1$ such that $D \succeq C$ and we can find a finite chain of LOS $\{\Delta_1/x_1\}, \dots, \{\Delta_n/x_n\}$ such that $D = C\{\Delta_1/x_1\}, \dots, \{\Delta_n/x_n\}$. By joining appropriately the LOS from these chains we have the composed LOS.

6 Quantifying Closeness among Programs

If P_1 subsumes P_2 we can find a finite chain of composed LOS which maps P_2 onto a subset of P_1. This chain has not to be unique. In a similar way to the clausal case, the shortest chain quantifies the closeness between programs. We formalize this idea in the next definitions.

Definition 9. *Let P_1 and P_2 be two programs such that $P_1 \succeq P_2$. We will say that $\mathcal{C} = \langle\langle \Theta_1, a_1\rangle, \dots, \langle \Theta_n, a_n\rangle\rangle$ is a chain from P_1 to P_2 if*

- *$\Theta_1, \dots, \Theta_n$ are composed LOS.*
- *For all $i \in \{1, \dots, n\}$, $a_i : P_2\Theta_1 \dots \Theta_{i-1} \to \Theta_i$ is an* amfa.
- *$P_2\Theta_1 \dots \Theta_n \subseteq P_1$ where the composed LOS have been applied via the correspondent a_i.*

In this case we will say that $\mathcal{C}$ is a chain of length n and we will denote it by $|\mathcal{C}| = n$. If $P_1 \subseteq P_2$ we will say that the empty chain, of length zero, is a chain from P_1 to P_2. The set of chains from P_1 to P_2 will be denoted by $\mathbf{L}(P_1, P_2)$.

If $P_1 \succeq P_2$, the set $\mathbf{L}(P_1, P_2)$ is not empty and the next definition makes sense.

Definition 10. *We will define the mapping $dp : \mathbb{P} \times \mathbb{P} \to [0, +\infty]$ as follows:*

$$dp(P_1, P_2) = \begin{cases} min\{|\mathcal{C}| \,:\, \mathcal{C} \in \mathbf{L}(P_1, P_2)\} & \text{if } P_1 \succeq P_2 \\ +\infty & \text{otherwise} \end{cases}$$

The mapping dp verifies the following properties:

- $P_1 \subseteq P_2 \Leftrightarrow dp(P_2, P_1) = 0$, in particular, $dp(P, P) = 0$
- In general, $dp(P_1, P_2) \neq dp(P_2, P_1)$, $P_1, P_2 \in \mathbb{P}$
- $dp(P_1, P_2) \leq dp(P_1, P_0) + dp(P_0, P_2)$ for all $P_1, P_2, P_3 \in \mathbb{P}$

hence, dp is a *pseudo-quasi-metric* and $(\mathbb{P}, dp)$ is a *quantitative domain*.

The next equivalence summarize our study about the generalization of programs under subsumption, by putting together our operators, the subsumption relation, the semantics of definite programs and the weak metric dp.

Theorem 6. *Let P_1 and P_2 be two definite programs. The following sentences are equivalent:*

- *For all interpretation* I, $T_{P_2}(I) \subseteq T_{P_1}(I)$.
- $P_1 \succeq P_2$.
- *There exists a finite chain of composed LOS* $\Theta_1 \dots \Theta_n$ *such that* $P_2\Theta_1 \dots \Theta_n \subseteq P_1$, *where each* Θ *is applied via an appropriate* amfa.
- $dp(P_1, P_2) < +\infty$.

Since the programs are finite set of clauses, the next theorem provides a method to compute the pseudo–quasi–distance of two programs.

Theorem 7. *Let* P_1 *and* P_2 *be two programs.*

$$dp(P_1, P_2) = \max_{D \in P_2} \left\{ \min_{C \in P_1} \{dc(C, D)\} \right\}$$

Note that the mapping $dp^*(P_1, P_2) = \max\{dp(P_1, P_2), dp(P_2, P_1)\}$ is the Hausdorff metric between programs based on the quasi-metric dc.

7 Related Work and Examples

The problem of quantifying the closeness among clauses has already been studied previously by offering distinct alternatives of solution to the problem. In the literature, a metric is firstly defined on the set of literals and then, the Hausdorff metric is used to get, from this metric, a metric on the set of clauses.

In [8], Nienhuys-Cheng defines a distance for ground atoms

- $d_{nc,g}(e, e) = 0$
- $p/n \neq q/m \Rightarrow d_{nc,g}(p(s_1, \dots, s_n), q(t_1, \dots, t_m)) = 1$
- $d_{nc,g}(p(s_1, \dots, s_n), p(t_1, \dots, t_n)) = \frac{1}{2n} \sum_{i=1}^{n} d_{nc,g}(s_i, t_i)$

then she uses the Hausdorff metric to define a metric on sets of ground atoms

$$d_h(A, B) = \max \left\{ \max_{a \in A}\{\min_{b \in B}\{d_{nc,g}(a, b)\}\}, \max_{b \in B}\{\min_{a \in A}\{d_{nc,g}(a, b)\}\} \right\}$$

The aim of this distance was to define a distance between Herbrand interpretations, so $d_{nc,g}$ was only defined on ground atoms. In [10], Ramon and Bruynooghe extended it to handle non–ground expressions:

- $d_{nc}(e_1, e_2) = d_{nc,g}(e_1, e_2)$ if e_1, e_2 are ground expressions
- $d_{nc}(p(s_1, \dots, s_n), X) = d_{nc}(X, p(s_1, \dots, s_n)) = 1$ with X a variable.
- $d_{nc}(X, Y) = 1$ and $d_{nc}(X, X) = 0$ for all $X \neq Y$ with X and Y variables.

This metric can be easily extended to literals: If A and B are atoms, we consider $d_{nc}(\neg A, B) = d_{nc}(A, \neg B) = 1$ and $d_{nc}(\neg A, \neg B) = d_{nc}(A, B)$. By applying the Hausdorff metric to d_{nc} we have a metric d_h on clauses. We have implemented dc and d_h with Prolog programs. The following example allows us to compare this metric with our quasi-metric.

Table 1. Comparison of dc vs. d_h

N	$dc(C_n, D_n)$		$d_h(C_n, D_n)$	
	Sec	Q–dist	Sec	Dist
64	0.02	3	0.11	$\sim 2.7\,10^{-20}$
128	0.06	3	0.21	$\sim 1.4\,10^{-39}$
256	0.1	3	0.43	$\sim 4.3\,10^{-78}$
512	0.26	3	0.93	$\sim 3.7\,10^{-155}$
1024	0.67	3	2.03	$\sim 2.7\,10^{-309}$

For all $n \geq 0$, consider the clauses

$$C_n \equiv sum(s^{n+1}(x_1), s^n(y_1), s^{n+1}(z_1)) \quad \leftarrow sum(s^n(x_1), s^n(y_1), s^n(z_1))$$
$$D_n \equiv sum(s^{2n+1}(x_2), s^{2n}(y_2), s^{2n+1}(z_2)) \leftarrow sum(s^{2n}(x_2), s^{2n}(y_2), s^{2n}(z_2))$$

and the substitution $\theta_n = \{x_1/s^n(x_2), y_1/s^n(y_2), x_3/s^n(y_3)\}$. Then $C_n\theta_n = D_n$ for all n and hence, $C_n \succeq D_n$. Table 1 shows the values of the quasi-metric $dc(C_n, D_n)$ and the metric $d_h(C_n, D_n)$ for several values of N as well as the time of computation on a PIII 800 Mhz. in an implementation for SWI-Prolog 4.0.11. It can be easily calculated that, for all $n \geq 0$, $dc(C_n, D_n) = 3$. If we use the Hausdorff metric d_h based on d_{nc} we have that, for all $n \geq 0$, $d_h(C_n, D_n) = \dfrac{1}{2^{n+1}}$ which tends to zero in spite of the subsumption relation holds for all n.

In the literature, other formalizations of the closeness among clauses (e.g. [4] or [10]) can be found.

If we consider now the clauses

$$C'_n \equiv sum(0, s^n(u_1), s^n(u_1)) \qquad \text{and} \qquad D'_n \equiv sum(0, s^{2n}(u_2), s^{2n}(u_2))$$

and the programs $P_n^1 = \{C_n, C'_n\}$ and $P_n^2 = \{D_n, D'_n\}$ we have that, for all $n \geq 0$, $dp(P_n^1, P_n^2) = 3$ and

$$d_{hh}(P_n^1, P_n^2) = \frac{1}{2^{n+1}}$$

where d_{hh} is the Hausdorff metric associated to d_h.

8 Conclusions and Future Work

The operators presented in this paper might provide a general framework to specify learning process based on Logic Programming [3]. The operators are not related to any specific system, they can be easily implemented and used in any system. But the main property is that the LOS are sufficient for all generalization process of clauses based on subsumption. As we have showed, the LOS are a *complete* set of generalization operators.

We define a quasi-metric on the set of clauses and give an algorithm to compute it as well as a method for a quick estimation. The process of quantifying qualitative relations (as subsumption) is a hard and exciting problem which

arises in many fields of Computer Science (see [6]) which is far from a complete solution. We present a contribution to its study by defining a quasi-metric on the set of clauses in a natural way, as the minimum number of operators which map a clause into another. As we have seen, this quasi-metric considers the clauses as members of a net of relations via subsumption and overcomes the drawbacks found in others formalizations of closeness.

The definition of quasi-metric is completed with an algorithm to compute it and a bound for a quick estimation. This estimation can be a useful tool for the design of new learning algorithms based on subsumption.

In the second part of the paper we present a family of operators which also represents an operational characterization of the subsumption between programs. These operators provide a weak metric which captures the idea of closeness among programs based on subsumption. The main results about generalization of programs are summarized in the theorem 6. The relation between the composed LOS and T_P opens a door for studying in the future new links between these operators and the semantics of logic programs.

References

1. M.R.Garey and D.S. Johnson: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York, 1979.
2. M.A. Gutiérrez-Naranjo, J.A. Alonso-Jiménez and J. Borrego-Díaz: A topological study of the upward refinement operators in ILP. In ILP 2000, Work in progress track.
3. M.A. Gutiérez Naranjo. Operadores de generalización para el aprendizaje clausal. Ph.D. Thesis. Dept. of Computer Science and AI. University of Seville, 2002.
4. A. Hutchinson: Metrics on Terms and Clauses. ECML–97, LNCS 1224, Springer, 1997.
5. P.D. Laird: Learning from Good and Bad Data. Kluwer Academic Publishers, 1988
6. R. Lowen: Approach Spaces, the Missing Link in the Topology-Uniformity-Metric Triad. Oxford Mathematical Monographs, Oxford University Press, 1997.
7. M.J. Maher: Equivalences of Logic Programs. In Foundations of Deductive Databases and Logic Programming. J. Minker ed, Morgan Kaufmann, 1988
8. S-H. Nienhuys-Cheng: Distance between Herbrand interpretations: a measure for approximations to a target concept. Technical Report EUR–FEW–CS–97–05. Department of Computer Science, Erasmus University, the Netherlands, 1997.
9. G.D. Plotkin: A Note on Inductive Generalization. In Machine Intelligence 5, pp.: 153–163. Edinburgh University Press, Edinburgh, 1970.
10. J. Ramon and M. Bruynooghe: A framework for defining distances between first–order logic–objects. Report CW 263, Dept. of Computer Science, KU Leuven, 1998.
11. E.Y. Shapiro: Inductive Inference of Theories from Facts. Technical Report 624, Department of Computer Science, Yale University, New Haven, CT, 1981
12. M. Schmidt-Schauss. Implication of clauses is undecidable. Theoretical Computer Science, 59–3, pp. 287–296, 1988.
13. P.R.J. van der Laag, S.-H. Nienhuys-Cheng: Completeness and properness of refinement operators in Inductive Logic Programming. Journal of Logic Programming, Vol 34, n.3, pp.: 201–225, 1998.