Refinement Operators can be (Weakly) Perfect

Liviu Badea and **Monica Stanciu**

AI Lab, Research Institute for Informatics
8-10 Averescu Blvd., Bucharest, Romania
e-mail: `badea@ici.ro`

Abstract. Our aim is to construct a *perfect* (i.e. minimal and optimal) ILP refinement operator for hypotheses spaces bounded below by a most specific clause and subject to syntactical restrictions in the form of input/output variable declarations (like in Progol). Since unfortunately no such optimal refinement operators exist, we settle for a weaker form of optimality and introduce an associated weaker form of subsumption which exactly captures a first incompleteness of Progol's refinement operator. We argue that this sort of incompleteness is not a drawback, as it is justified by the examples and the MDL heuristic.
A second type of incompleteness of Progol (due to subtle interactions between the requirements of non-redundancy, completeness and the variable dependencies) is more problematic, since it may sometimes lead to unpredictable results. We remove this incompleteness by constructing a sequence of increasingly more complex refinement operators which eventually produces the first (weakly) *perfect* refinement operator for a Progol-like ILP system.

1 Introduction

Learning logic programs from examples in Inductive Logic Programming (ILP) involves traversing large spaces of hypotheses. Various heuristics, such as information gain or example coverage, can be used to guide this search. A simple search algorithm (even a complete and non-redundant one) would not do, unless it allows for a flexible traversal of the search space, based on an external heuristic. Refinement operators allow us to decouple the heuristic from the search algorithm.

In order not to miss solutions, the refinement operator should be (weakly) complete. In order not to revisit already visited portions of the search space it should also be non-redundant. Such weakly complete non-redundant refinement operators are called *optimal*.

Various top-down ILP systems set a lower bound (usually called most specific clause (MSC) or saturant) on the hypotheses space in order to limit its size. Syntactical restrictions in the form of mode declarations on the predicate arguments are also used as a declarative bias.

Devising an optimal refinement operator for a hypotheses space bounded below by a MSC in the presence of input/output ($\pm$) variable dependencies is not only a challenging issue given the subtle interactions of the above- mentioned features, but also a practically important one since refinement operators represent the core of an ILP system.

The Progol refinement operator [4], for example, is incomplete in two ways. First, it is incomplete w.r.t. ordinary subsumption since each literal from the MSC has *at most one* corresponding literal in each hypothesis (only variabilized *subsets* of the MSC are considered as hypotheses). Rather than being a drawback, we argue that this incompleteness is exactly the sort of behavior we would expect. In order to make this observation precise, we introduce a weaker form of subsumption under which the refinement operator is complete.

The second type of incompleteness (see also example 30 of [4]) is more problematic since it cannot be characterized in a clean way and since it depends on the ordering of mode declarations and examples. In order to achieve non-redundancy and at the same time obey the ±variable dependencies imposed by the mode declarations, Progol scans the MSC left-to-right and non-deterministically decides for each literal whether to include it in (or exclude it from) the current hypothesis. A variabilized version of the corresponding MSC literal is included in the current hypothesis only if all its input (+) variables are preceded by suitable output (−) variables. This approach is incomplete since it would reject a literal l_i that obtains a +variable from a literal l_j that will be considered only later:

$$\ldots, l_i(\cdots, +\overrightarrow{X, \cdots), \ldots, l_j(\cdots, -}X, \cdots), \ldots$$

although l_j, l_i would constitute a valid hypothesis.

Note that a simple idea like reordering the literals in the MSC would not help in general, since the MSC may exhibit cyclic variable dependencies while still admitting acyclic subsets.

The following example illustrates the above-mentioned incompleteness:

```
:- modeh(1, p(+any, +t))?  :- modeb(1, f(-any, -t))?
:- modeb(1, g(+any, +t))?  :- modeb(1, h(-any, -t))?
p(1,a).  p(2,a).  p(3,a).  :-p(4,a).
f(1,b).  f(2,b).  f(3,b).  f(4,b).
g(1,a).  g(2,c).  g(3,c).
h(1,a).  h(2,c).  h(3,c).  h(4,a).
```

As long as the mode declaration for `g` precedes that of `h`, Progol will produce a MSC `p(A,B) :- f(A,C), g(A,B), h(A,B)` in which the `g` literal cannot obtain its +variables from `h` since the former precedes the latter in the MSC. Thus Progol will miss the solution `p(A,B) :- h(A,C), g(A,C)` which can be found only if we move the mode declaration for `h` before that of `g`. This type of incompleteness may sometimes lead to unpredictable results and a reordering of the mode declarations will not always be helpful in solving the problem.

Although Progol's algorithm for constructing the MSC makes sure that each +variable occurrence is preceded by a corresponding −variable, there may be several other −variable occurrences in literals ordered after the literal containing the +variable. These potential "future links" will be missed by Progol. In cases of many such "future links", the probability of the correct ordering being in the search space is exponentially low in the number of "future links".

For a *very small* number of literals in the body and/or a small variable depth, this incompleteness may not be too severe, especially if we order the mode declarations appropriately. The problem becomes important for hypotheses with a larger number of literals.

2 Refinement Operators

In order to be able to guide the search in the space of hypotheses by means of an external heuristic, we need to construct a refinement operator. For a top-down search, we shall deal with a *downward* refinement operator, i.e. one that constructs clause *specializations*. In the following we will consider refinement operators w.r.t. the subsumption ordering between clauses.

Definition 1. Clause C *subsumes* clause D, $C \succeq D$ iff there exists a substitution θ such that $C\theta \subseteq D$ (the clauses being viewed as sets of literals). C *properly subsumes* D, $C \succ D$ iff $C \succeq D$ and $D \not\succeq C$. C and D are *subsume-equivalent* $C \sim D$ iff $C \succeq D$ and $D \succeq C$.

Lemma 2. *[5] For a most general literal L w.r.t. clause C (one with new and distinct variables), C* properly *subsumes $C \cup \{L\}$ iff L is* incompatible *with all literals in C (i.e. it has a different predicate symbol).*

The somewhat subtle and counter-intuitive properties of subsumption are due to the incompatibility of the induced subsumption-equivalence relation $\sim$ with the elementary operations of a refinement operator, such as adding a literal or performing a substitution.

Remark. Note that not all reduced specializations D of a reduced clause C can be obtained just by adding one literal or by making a simple substitution $\{X/Y\}$. It may be necessary to add several literals and make several simple substitutions in one refinement step, since each of these elementary operations applied separately would just produce a clause that is subsume- equivalent with C.

Definition 3. ρ is a (downward) *refinement operator* iff for all clauses C, ρ produces only specializations of C: $\rho(C) \subseteq \{D \mid C \succeq D\}$.

Definition 4. A refinement operator ρ is called

- *(locally) finite* iff $\rho(C)$ is finite and computable for all C.
- *proper* iff for all C, $\rho(C)$ contains no $D \sim C$.
- *complete* iff for all C and D, $C \succ D \Rightarrow \exists E \in \rho^*(C)$ such that $E \sim D$.
- *weakly complete* iff $\rho^*(\Box) =$ the entire set of clauses.
- *non-redundant* iff for all C_1, C_2 and D, $D \in \rho^*(C_1) \cap \rho^*(C_2) \Rightarrow C_1 \in \rho^*(C_2)$ or $C_2 \in \rho^*(C_1)$.
- *ideal* iff it is locally finite, proper and complete.

– *optimal* iff it is locally finite, non-redundant and weakly complete.
– *minimal* iff for all C, $\rho(C)$ contains only downward covers[1] and all its elements are incomparable ($D_1, D_2 \in \rho(C) \Rightarrow D_1 \not\succeq D_2$ and $D_2 \not\succeq D_1$).
– *(downward) cover set* iff $\rho(C)$ is a maximal set of non- equivalent downward covers of C.
– *perfect* iff it is minimal and optimal.

Theorem 5. *[6]. For a language containing at least one predicate symbol of arity ≥ 2, there exist no ideal (downward) refinement operators.*

The nonexistence of ideal refinement operators is due to the incompleteness of the (unique) cover set of a clause C, because of *uncovered infinite ascending chains* $C \succ \ldots \succ E_{i+1} \succ E_i \succ E_{i-1} \succ \ldots \succ E_1$ (for which there exists no maximal element $E \succeq E_i$ for all i, such that $C \succ E$). Indeed, since none of the E_i can be a downward cover of C, C cannot have a complete downward cover set.

Every ideal refinement operator ρ determines a finite and complete downward cover set $\rho^{dc}(C) \subseteq \rho(C)$, obtained from $\rho(C)$ by removing all E covered by some $D \in \rho(C)$: $D \succeq E$.

3 Ideal versus optimal refinement operators

The subsumption lattice of hypotheses is far from being tree-like: a given clause D can be reachable from several incomparable hypotheses $C_1, C_2, \ldots$.

Theorem 6. *A refinement operator cannot be both* complete *(a feature of* ideal *operators) and* non-redundant *(a feature of* optimal *operators).*

Proposition 7. *For each ideal refinement operator ρ we can construct an optimal refinement operator $\rho^{(o)}$.*

$\rho^{(o)}$ is obtained from ρ such that for $D \in \rho(C_1) \cap \ldots \cap \rho(C_n)$ we have $\exists i. D \in \rho^{(o)}(C_i)$ and $\forall j \neq i. D \notin \rho^{(o)}(C_j)$.

The efficiency of ideal and respectively optimal refinement operators depends on the density of solutions in the search space. *Ideal* operators are preferable for search spaces with *dense solutions,* for which almost any refinement path leads to a solution. In such cases, an optimal (non-redundant) operator might get quite close to a solution C but could backtrack just before finding it for reasons of non-redundancy (for example because C is scheduled to be visited on a different path and thus it avoids revisiting it). Despite this problem, the solutions are dense, so an optimal operator would not behave too badly, after all.

On the other hand, *optimal* operators are preferable for search spaces with rare solutions, case in which a significant portion of the search space would be traversed and any redundancies in the search due to an ideal operator would be very time consuming.

[1] D is a downward cover of C iff $C \succ D$ and no E satisfies $C \succ E \succ D$.

Thus, unless we are dealing with a hypotheses space with a very high solution density, we shall prefer an optimal operator over an ideal one. However, in practice we shall proceed by first constructing an ideal refinement operator ρ and only subsequently modifying it, as in proposition 7, to produce an optimal operator $\rho^{(o)}$.

4 Refinement operators for hypotheses spaces bounded below by a MSC

Limiting the hypotheses space below by a most specific (bottom) clause $\perp$ leads to a more efficient search.[2] This strategy has proven successful in state-of-the-art systems like Progol, which search the space of hypotheses C between the most general clause (for example the empty clause $\square$) and the most specific clause $\perp$: $\square \succeq C \succeq \perp$ (for efficiency reasons, the generality ordering employed is subsumption rather than full logical implication).

Formalizing Progol's behavior amounts to considering hypotheses spaces consisting of clause-substitution pairs $C = (cl(C), \theta_\perp(C))$ such that $cl(C)\theta_\perp(C) \subseteq \perp$. (For simplicity, we shall identify in the following $cl(C)$ with C.[3])

The following refinement operator is a generalization of Laird's operator in the case of hypotheses spaces bounded below by a MSC $\perp$.

$D \in \rho_\perp^{(L)}(C)$ iff either

(1) $D = C \cup \{L'\}$ with $L \in \perp$ (L' denotes a literal with the same predicate symbol as L, but with new and distinct variables), or
(2) $D = C\{X_j/X_i\}$ with $\{X_i/A, X_j/A\} \subseteq \theta_\perp(C)$.

Note that in (2) we unify only variables X_i, X_j corresponding to the same variable A from $\perp$ (since otherwise we would obtain a clause more specific than $\perp$).

$\rho_\perp^{(L)}$ is finite, complete, but improper. The lack of properness is due to the possibility of selecting a given literal $L \in \perp$ several times in the current hypothesis (using (1)). It can be easily shown that the nonexistence result 5 for ideal refinement operators can be restated in the case of hypotheses spaces bounded below by a MSC. Therefore, we cannot hope to convert $\rho_\perp^{(L)}$ (which is improper) to an ideal operator.

On the other hand, the Progol implementation uses a slightly weaker refinement operator that considers each literal $L \in \perp$ for selection *only once*. This weaker operator is no longer complete, anyway not w.r.t. ordinary subsumption.

For example, if $\perp = \ldots \leftarrow p(A, A)$, then the weaker refinement operator would construct only hypotheses with a single p-literal, such as $H_1 = \ldots \leftarrow$

[2] In the following, we restrict ourselves for simplicity to refinement operators for *flattened definite Horn clauses*.

[3] In general, for a given clause C there can be several distinct substitutions θ_i such that $C\theta_i \subseteq \perp$. Viewing the various clause-substitution pairs (C, θ_i) as distinct hypotheses amounts to distinguishing the $\perp$-literals associated to each of the literals of C.

$p(X,X)$, or $H_2 = \ldots \leftarrow p(X,Y)$, but it will never consider hypotheses with multiple p-literals, like $H_3 = \ldots \leftarrow p(X,Y), p(Y,X)$, or $H_4 = \ldots \leftarrow p(X,Y), p(Y,Z), p(Z,W)$, etc. since such hypotheses could be constructed only if we would allow selecting (suitably variabilized versions of) the literal $p(A,A)$ *several times* (and not just once, as in Progol). Note that H_3 is strictly more general (w.r.t. subsumption) than H_1, but also strictly more specific than H_2: $H_2 \succ \boxed{H_3} \succ H_1$. Since H_3 is not even in the search space, Progol's refinement operator is, in a way, incomplete.[4] This incompleteness is due to the fact that $\perp$ is scanned only once for literal selection. It could be easily bridged by scanning $\perp$ repeatedly, so that a given literal can be selected several times. Unfortunately, in principle we cannot bound the number of traversals, although in practice we can set an upper bound.

On the other hand, looking at the above-mentioned incompleteness more carefully, we are led to the idea that it is somehow *justified by the examples* and the MDL principle.

In our previous example, if $p(A,A)$ is the only p-literal in $\perp$, then it may be that something like: `p(a,a). p(b,b). p(c,c). [ex]` are the only examples. In any case, we could not have had examples like `p(a,b). p(b,a).` which would have generated $\perp = \ldots \leftarrow p(A,B), p(B,A)$ instead of $\perp = \ldots \leftarrow p(A,A)$. Now, it seems reasonable to assume that a hypothesis like $H = \ldots \leftarrow p(X,Y), p(Y,X)$, although logically consistent with the examples `[ex]`, is not "required" by them. So, although Progol generally returns the most general hypothesis consistent with the examples, in the case it has to choose between hypotheses with multiple occurrences of the same literal from $\perp$, it behaves as if it would always prefer the more specific one (the one with just one occurrence). A justification of this behavior could be that the more general hypotheses are not "required" by the examples. Also, the more general hypotheses (with several occurrences of some literal from $\perp$) are always longer (while covering the same number of examples) and thus will be discarded by the MDL principle anyway.

As we have already mentioned, the subtle properties of subsumption are due to the possibility of clauses with more literals being more general than clauses with fewer literals. This is only possible in the case of multiple occurrences of literals with the same predicate symbol (as for example in uncovered infinite ascending chains).

In the following, we introduce a weaker form of subsumption, which exactly captures Progol's behavior by disallowing substitutions that identify literals.

Definition 8. Clause C *weakly-subsumes* clause D *relative to* $\perp$, $C \succeq_w D$ iff $C\theta \subseteq D$ for some substitution θ that does not identify literals (i.e. for which there are no literals $L_1, L_2 \in C$ such that $L_1\theta = L_2\theta$) and such that $\theta_\perp(D) \circ \theta = \theta_\perp(C)$.

Note that although in the above example $H_3 \succ H_1$ w.r.t. (ordinary) subsumption, they become incomparable w.r.t. weak subsumption because the substitution $\theta = \{Y/X\}$ that ensures the subsumption relationship $H_3 \succ H_1$ identifies the literals $\overline{p(X,Y)}$ and $\overline{p(Y,X)}$.

[4] It's *search* however, is complete if we use the MDL heuristic. See below.

Although Progol's refinement operator is in a way incomplete w.r.t. ordinary subsumption, it is complete w.r.t. weak subsumption. Disallowing substitutions that identify literals entails the following properties of weak subsumption.

Proposition 9. *If $C \succeq_w D$ then $|C| \leq |D|$ (where $|C|$ is the length of the clause C, i.e. the number of its literals).*

Lemma 10. *(a) In the space of clauses ordered by weak subsumption there exist no infinite ascending chains (and therefore no uncovered infinite ascending chains).*
(b) there exist no uncovered *infinite descending chains.*

Lemma 10(a) implies the existence of *complete downward cover sets*, which can play the role of ideal operators for weak subsumption.

A form of subsumption even weaker than weak subsumption is *subsumption under object identity* [1]: $C \succeq_{OI} D$ iff $C\theta \subseteq D$ for some substitution θ that does not unify variables of C. For example, $p(X,Y) \not\succeq_{OI} p(X,X)$, showing that subsumption under object identity is too weak for our purposes.

A form of subsumption slightly stronger than weak subsumption (but still weaker than ordinary subsumption) is *"non-decreasing" subsumption*: $C \succeq_{ND} D$ iff $C\theta \subseteq D$ and $|C| \leq |D|$. (Such a substitution θ can identify literals of C, but other literals have to be left out when going from C to D to ensure $|C| \leq |D|$. This leads to somewhat cumbersome properties of "non-decreasing" subsumption.)

Concluding, we have the following chain of increasingly stronger forms of subsumption: $C \succeq_{OI} D \Rightarrow C \succeq_w D \Rightarrow C \succeq_{ND} D \Rightarrow C \succeq D$.

We have seen that Laird's operator $\rho_\perp^{(L)}$ is locally finite, complete, but improper and that it cannot be converted to an ideal operator w.r.t. subsumption. However, it can be converted to an *ideal operator w.r.t. weak subsumption* by selecting each literal $L \in \perp$ at most once:

$D \in \rho_\perp^{(1)}(C)$ iff either

(1) $D = C \cup \{L'\}$ with $L \in \perp \setminus C\theta_\perp(C)$ (L' being L with new and distinct variables), or
(2) $D = C\{X_j/X_i\}$ with $\{X_i/A, X_j/A\} \subseteq \theta_\perp(C)$.

Since literals from $\perp$ are selected only once, $\rho_\perp^{(1)}$ turns out proper, and although it looses completeness w.r.t. ordinary subsumption, it is still complete w.r.t. weak subsumption.

4.1 From ideal to optimal refinement operators

We have already seen (theorem 6) that, due to completeness, ideal refinement operators cannot be non-redundant and therefore optimal. As already argued in section 3, non-redundancy is extremely important for efficiency. We shall therefore transform the ideal refinement operator $\rho_\perp^{(1)}$ to an optimal operator $\rho_\perp^{(1o)}$ by replacing the stronger requirement of completeness with the weaker one of weak completeness. Non-redundancy is achieved (like in proposition 7) by assigning a successor $D \in \rho_\perp^{(1)}(C_i) \cap \ldots \cap \rho_\perp^{(1)}(C_n)$ to one and only one of its predecessors C_i: $D \in \rho_\perp^{(1o)}(C_i)$ and $\forall j \neq i . D \notin \rho_\perp^{(1o)}(C_j)$. The refinement graph of such a non-redundant operator becomes tree-like. If the operator is additionally weakly complete, then every element in the search space can be reached through exactly one refinement chain.

The essential cause for the redundancy of a refinement operator (like $\rho_\perp^{(1)}$) is the *commutativity* of the operations of the operator (such as literal addition (1) and elementary substitution (2) in the case of $\rho_\perp^{(1)}$). For example, $D' \cup \{L_1, L_2\}$ can be reached both from $D' \cup \{L_2\}$ by adding L_1 and from $D' \cup \{L_1\}$ by adding L_2. This redundancy is due to the commutativity of the operations of adding literal L_1 and literal L_2 respectively. A similar phenomenon turns up in the case of substitutions.

The assignment of D to one of its successors C_i is largely arbitrary, but has to be done for ensuring non-redundancy. This can be achieved by imposing an ordering on the literals in $\perp$ and making the selection decisions for the literals $L_i \in \perp$ in the given order. We also impose an ordering on the variable occurrences in $\perp$ and make the unification decisions for these variable occurrences in the given order. Finally, we have to make sure that literal additions (1) do not commute with elementary substitutions (2). This is achieved by allowing only substitutions involving newly introduced ("fresh") variables (the substitutions involving "old" variables having been performed already).

Optimal refinement operators have been introduced in [2] for the system $\mathcal{C}$LAUDIEN. However, the refinement operator of $\mathcal{C}$LAUDIEN is optimal only w.r.t. literal selection (which makes the problem a lot easier since variabilizations are not considered). One could simulate variabilizations by using explicit equality literals in the $\mathcal{D}$LAB templates, but the resulting algorithm is no longer optimal since the transitivity of equality is not taken into account. For example, in case of a template containing $\ldots \leftarrow [X = Y, X = Z, Y = Z]$, the algorithm would generate the following equivalent clauses: $\ldots \leftarrow X = Y, X = Z$ and $\ldots \leftarrow X = Y, Y = Z$.

In the following, we construct an optimal operator $\rho_\perp^{(1o)}$ (w.r.t. weak subsumption) associated to the ideal operator $\rho_\perp^{(1)}$. We start by assuming an ordering of the literals $L_k \in \perp$: L_k precedes L_l in this ordering iff $k < l$. This ordering will be used to order the selection decisions for the literals of $\perp$: we will not consider selecting a literal L_k if a decision for L_l, $l > k$ has already been made (we shall call this rule the 'literal rule').

The ordering of the selection decisions for literals also induces an ordering on the variable occurrences[5] in $\perp$: X_i precedes X_j in this ordering $(i < j)$ iff X_i is a variable occurrence in a literal selected before the literal containing X_j, or X_i precedes X_j in the same literal.

To achieve non-redundancy, we will impose an order in which the substi-

[5] To each argument of each literal of $\perp$ we assign a new and distinct variable X_i (denoting a variable occurrence).

tutions are to be performed. Namely, we shall disallow substitutions $\{X_j/X_i\}$, $i < j$ if some X_k with $k > j$ had already been involved in a previous substitution $\{X_k/X_l\}$ (we shall refer to this rule as the 'substitution rule'). Roughly speaking, we make substitutions in increasing order of the variables involved. This ensures the nonredundancy of the operation of making substitutions.

Having operators for selecting literals and making substitutions that are non-redundant if taken separately, does not ensure a non-redundant refinement operator when the two operators are combined: we also have to make sure that literal additions (1) and elementary substitutions (2) do not commute. For example, if $\perp = \ldots \leftarrow p(A, A), q(B)$, the clause $\ldots \leftarrow p(X, X), q(Z)$ could be obtained either by adding the literal $\overline{q(Z)}$ to $C_1 = \ldots \leftarrow p(X, X)$ or by making the substitution $\{Y/X\}$ in $C_2 = \ldots \leftarrow p(X, Y), q(Z)$. The latter operation should be disallowed since it involves no "fresh" variables (assuming that $\overline{q(Z)}$ was the last literal added, Z is the only "fresh" variable).

In general, we shall allow only substitutions involving at least a "fresh" variable (we shall call this rule the 'fresh variables rule'). The set of "fresh" variables is initialized when adding a new literal L with the variables of L. Variables involved in subsequent substitutions are removed from the list of "fresh" variables. Substitutions involving no fresh variables are assumed to have already been tried on the ancestors of the current clause and are therefore disallowed.

The three rules above (the literal, substitution and fresh variables rules) are sufficient to turn $\rho_\perp^{(1)}$ into an optimal operator w.r.t. weak subsumption. However, the substitution rule forces us to explicitly work with variable occurrences, instead of just working with the substitutions $\theta_\perp(C)$ (if X_l and X_k are variable occurrences, then the substitution $\{X_k/X_l\}$ would eliminate X_k when working just with substitutions, and later on we wouldn't be able to interdict a substitution $\{X_j/X_i\}$ for $j < k$ because we would no longer have X_k).

Fortunately, it is possible to find a more elegant formulation of the substitution and fresh variable rules combined. For each clause C, we shall keep a list of fresh substitutions $\mathcal{F}(C)$ that is initialized with $\theta_\perp(L)$ when adding a new literal L. As before, we shall allow only substitutions $\{X_j/X_i\}$ involving a fresh variable X_j. But we eliminate from $\mathcal{F}(C)$ not just X_j/A, but all X_k/B with $k \le j$ ('prefix deletion rule'). This ensures that after performing a substitution of X_j, no substitution involving a "smaller" X_k will ever be attempted. This is essentially the substitution rule in disguise, only that now we can deal with substitutions instead of a more cumbersome representation that uses variable occurrences.

The optimal refinement operator can thus be defined as:

$D \in \rho_\perp^{(1o)}(C)$ iff either

(1) $D = C \cup \{L'\}$ with $L \in \perp \setminus \text{prefix}_\perp(C\theta_\perp(C))$, where $\text{prefix}_\perp(C) = \{L_i \in \perp \mid \exists L_j \in C \text{ such that } i \le j\}$ and L' is L with new and distinct variables. $\theta_\perp(D) = \theta_\perp(C) \cup \theta_\perp(L')$, $\mathcal{F}(D) = \theta_\perp(L')$, or

(2) $D = C\{X_j/X_i\}$ with $i < j$, $\{X_i/A, X_j/A\} \subseteq \theta_\perp(C)$ and $X_j/A \in \mathcal{F}(C)$. $\theta_\perp(D) = \theta_\perp(C) \cup \{X_j/X_i\}$, $\mathcal{F}(D) = \mathcal{F}(C) \setminus \{X_k/B \in \mathcal{F}(C) \mid k \le j\}$.

Example 1. Let $\bot = p(A, A, A, A)$ and $C_1 = p(X_1, X_2, X_3, X_4)$, $\theta_\bot(C_1) = \{X_1/A, X_2/A, X_3/A,\ X_4/A\}$, $\mathcal{F}(C_1) = \{\underbrace{X_1/A, X_2/A, X_3/A}, X_4/A\}$. The substitution $\{X_3/X_1\}$ eliminates the entire prefix $\{X_1/A, X_2/A, X_3/A\}$ from $\mathcal{F}(C_1)$: $C_2 = C_1\{X_3/X_1\} = p(X_1, X_2, X_1, X_4)$, $\theta_\bot(C_2) = \{X_1/A, X_2/A, X_4/A\}$, $\mathcal{F}(C_2) = \{X_4/A\}$. Now, only the substitutions involving X_4 are allowed for C_2.

5 Refinement operators for clauses with variable dependencies

Most implemented ILP systems restrict the search space by providing mode declarations that impose constraints on the types of the variables. Also, variables are declared as either input (+) or output (−). In this setting, *valid* clauses (clauses verifying the mode declarations) are *ordered* sets of literals such that every input variable $+X$ in some literal L_i is preceded by a literal L_j containing a corresponding output variable $-X$.[6] These variable dependencies induced by the ±mode declarations affect a refinement operator for an unrestricted language (like $\rho_\bot^{(1o)}$), since now not all refinements according to the unrestricted $\rho_\bot^{(1o)}$ are valid clauses. Nevertheless, we can use $\rho_\bot^{(1o)}$ to construct an optimal refinement operator $\rho_\bot^{(1o)(\pm)}$ for clauses with ±variable dependencies by repeatedly using the "unrestricted" $\rho_\bot^{(1o)}$ to generate refinements until a valid refinement is obtained. One-step refinements w.r.t. $\rho_\bot^{(1o)(\pm)}$ can thus be multi-step refinements w.r.t. $\rho_\bot^{(1o)}$ (all intermediate steps being necessarily invalid).

$$\rho_\bot^{(1o)(\pm)}(C) = \{D_n \mid D_1 \in \rho_\bot^{(1o)}(C), D_2 \in \rho_\bot^{(1o)}(D_1), \ldots, D_n \in \rho_\bot^{(1o)}(D_{n-1}) \text{ such that } D_n \text{ is valid but } D_1, \ldots, D_{n-1} \text{ are invalid}\}.$$

(Here, a clause C is called valid iff it admits an ordering that satisfies the mode declarations. Such an ordering can easily be obtained with a kind of topological sort.)

Note that $\rho_\bot^{(1o)(\pm)}$ is optimal, but *non-minimal*, since it may add more than one literal in one refinement step. For example, if $\bot = \ldots \leftarrow p(+A), q(-A)$, it would generate both $C_1 = \ldots \leftarrow q(-X), p(+X)$ and $C_2 = \ldots \leftarrow q(-X)$ as one-step refinements of $\Box$, whereas it may seem more natural to view C_1 as a one-step refinement of C_2 (rather than of $\Box$).

To obtain a *minimal* optimal (i.e. a *perfect*) refinement operator, we need to postpone the choice of unselectable (variabilizations of) literals until they become selectable, instead of making a selection decision right away. We also have to make sure that selectable literals obtain all their +variables from previous −variables by making all the corresponding substitutions in one refinement step.

(1) add a new literal (to the right of all selected ones) and link all its +variables according to "old" substitutions

[6] This is true for literals in the body of the clause. Input variables in the head of the clause behave exactly like output variables of body literals. To simplify the presentation, we shall not explicitly refer to the head literal.

(2) make a substitution (involving at least a "fresh" variable)
(3) wake-up a previously unselectable literal (to the left of the rightmost selected literal).

If several literals can be added at a given time, we shall add them in the order induced by $\bot$. In other words, we shall disallow adding a literal $L_2 < L_1$ after adding L_1 if L_2 was selectable even before adding L_1 (since otherwise we would redundantly generate C, L_1, L_2 both from C, L_1 by adding L_2 and from C, L_2 by adding L_1).

On the other hand, the multiple +variables of several literals can be linked to the same −variable ("source"). For example, if $\bot = \ldots \leftarrow p_1(+A), p_2(+A), q(-A)$, then $\rho(\Box) = \{C_1\}$, $\rho(C_1) = \{C_2, C_3\}$, $\rho(C_2) = \{C_4\}$ and $\rho(C_3) = \rho(C_4) = \emptyset$, where $C_1 = \ldots \leftarrow q(-X)$, $C_2 = \ldots \leftarrow q(-X), p_1(+X)$, $C_3 = \ldots \leftarrow q(-X), p_2(+X)$, $C_4 = \ldots \leftarrow q(-X), p_1(+X), p_2(+X)$. Note that both $p_1(+X)$ and $p_2(+X)$ use the same "source" $q(-X)$. Also note that adding $p_2(+X)$ to C_2 is allowed since $p_2 > p_1$, while adding $p_1(+X)$ to C_3 is disallowed because $p_1 < p_2$ was selectable even before adding p_2 (to C_1).

Using the notation $\theta_\bot^+(L')$ (and respectively $\theta_\bot^-(L')$) for the substitutions of $+(-)$variables of $\theta_\bot(L')$, we obtain the following *perfect* (*minimal*[7] and *optimal*) refinement operator $\rho_\bot^{(1o\pm)}$ for clauses with variable dependencies.

$D \in \rho_\bot^{(1o\pm)}(C)$ iff either

(1) $D = C \cup \{L'\}$ with $L \in \bot \setminus \text{prefix}_\bot(C\theta_\bot(C))$, where L' is L with new and distinct −variables[8], and +variables such that $\theta_\bot^+(L') \subseteq \theta_\bot(D) = \theta_\bot(C) \cup \theta_\bot^-(L')$.
$\mathcal{F}(D) = \theta_\bot^-(L') \setminus \{X_k/B \in \theta_\bot^-(L') \mid k \le j \text{ for some } X_j/A \in \theta_\bot^+(L') \cup \theta_\bot^-(L')\}$, or
(2) $D = C\{X_j/X_i\}$ with $i < j$, $\{X_i/A, X_j/A\} \subseteq \theta_\bot(C)$ and $X_j/A \in \mathcal{F}(C)$.
$\theta_\bot(D) = \theta_\bot(C) \cup \{X_j/X_i\}$, $\mathcal{F}(D) = \mathcal{F}(C) \setminus \{X_k/B \in \mathcal{F}(C) \mid k \le j\}$, or
(3) $D = C \cup \{L'\}$ with $L \in \text{prefix}_\bot(C\theta_\bot(C)) \setminus C\theta_\bot(C)$, where L' is L with new and distinct −variables[7], and +variables such that $\theta_\bot^+(L') \subseteq \theta_\bot(D) = \theta_\bot(C) \cup \theta_\bot^-(L')$, and for all $L_i \in C$ such that $L_i > L'$, $\mathit{first}_{L_i}(C) \cup \{L'\}$ is invalid, where $\mathit{first}_{L_i}(C)$ are the literals added to C before L_i.
$\mathcal{F}(D) = \theta_\bot^-(L') \setminus \{X_k/B \in \theta_\bot^-(L') \mid k \le j \text{ for some } X_j/A \in \theta_\bot^+(L') \cup \theta_\bot^-(L')\}$.

Observe that substitutions (2) involving −variables (controlled by $\mathcal{F}$) are to be performed right away when $\mathcal{F}$ allows it, because later additions of woken-up literals will reset $\mathcal{F}$ and make those substitutions impossible. On the other hand, we can always wake-up literals (by solving their +variables) (3) *after*

[7] The refinement operator of *Markus* is not minimal (no description of this operator is available in [3], but see his footnote 4). As far as we know, our refinement operator is the first *minimal* and *optimal* one (w.r.t. weak subsumption).
[8] The −variables of L' are preceded by all −variables of C in our variable ordering. (This variable ordering is *dynamical* since it is induced by the selection decisions for literals.)

making those substitutions. In other words, we firmly establish our "sources" (i.e. $-$variables) before waking-up the "consumers" (i.e. $+$variables).

The following example illustrates the functioning of $\rho_\perp^{(1o\pm)}$.

Example 2. For $\perp = \ldots \leftarrow r(+B), q_1(+A,-B), q_2(+A,-B), p(-A)$, $\rho(\Box) = \{C_1\}$, $\rho(C_1) = \{C_2, C_3\}$, $\rho(C_2) = \{C_4, C_5\}$, $\rho(C_3) = \{C_6\}$, $\rho(C_4) = \{C_7\}$, $\rho(C_5) = \{C_8, C_9\}$, $\rho(C_7) = \{C_{10}\}$, $\rho(C_6) = \rho(C_8) = \rho(C_9) = \rho(C_{10}) = \emptyset$, where

$$\begin{array}{ll}
C_1 = \ldots \leftarrow p(-X) & C_2 = \ldots \leftarrow p(-X), q_1(+X,-Y) \\
C_3 = \ldots \leftarrow p(-X), q_2(+X,-Y) & C_4 = \ldots \leftarrow p(-X), q_1(+X,-Y), r(+Y) \\
C_5 = \ldots \leftarrow p(-X), q_1(+X,-Y), q_2(+X,-Y') & C_6 = \ldots \leftarrow p(-X), q_2(+X,-Y), r(+Y)
\end{array}$$

$$\begin{aligned}
C_7 &= \ldots \leftarrow\ p(-X), q_1(+X,-Y), r(+Y), q_2(+X,-Y') \\
C_8 &= \ldots \leftarrow\ p(-X), q_1(+X,-Y), q_2(+X,-Y) \\
C_9 &= \ldots \leftarrow\ p(-X), q_1(+X,-Y), q_2(+X,-Y'), r(+Y') \\
C_{10} &= \ldots \leftarrow\ p(-X), q_1(+X,-Y), q_2(+X,-Y), r(+Y).
\end{aligned}$$

Note that $C_{10} \notin \rho(C_8)$ because $q_2 > r$ in $\perp$ and $\mathit{first}_{q_2}(C_8) \cup \{\overline{r(+Y)}\} = C_2 \cup \{\overline{r(+Y)}\} = C_4$ is valid, thereby violating the condition of step (3). In other words, we cannot wake up $r(+Y)$ in C_8 because it was already selectable in C_2 (otherwise we would obtain the redundancy $C_{10} \in \rho(C_7) \cap \rho(C_8)$). For similar reasons, we don't have $C_7 \in \rho(C_5), C_5 \in \rho(C_3)$ or $C_9 \in \rho(C_6)$.

On the other hand, $C_{10} \notin \rho(C_9)$ because the substitution $\{Y'/Y\}$ in C_9 is disallowed by $\mathcal{F}(C_9) = \emptyset$ (the last literal added, $r(+Y')$, having no $-$variables).

Note that the incompleteness of Progol's refinement operator (which applies only steps (1) and (2)) is due to obtaining the substitutions of $+$variables only from $-$variables of already selected literals, whereas they could be obtained from $-$variables of literals that will be selected in the future (as in step (3)). For example, if $\perp = \ldots \leftarrow p(-A), q(+A), r(-A)$, then Progol's refinement of $C = \ldots \leftarrow p(-X)$ will miss the clause $D = \ldots \leftarrow p(-X), r(-Y), q(+Y)$ in which q obtains its $+Y$ from r.

We have implemented the refinement operators described in this paper and plan to use them as a component in a full ILP system.

References

1. Esposito F., A. Laterza, D. Malerba, G. Semeraro. *Refinement of Datalog Programs.* Workshop on Data Mining and ILP, Bari 1996.
2. De Raedt L., M. Bruynooghe. *A theory of clausal discovery.* IJCAI-93, 1058-1063.
3. Grobelnik M. *Induction of Prolog programs with Markus.* LOPSTR'93, 57-63.
4. Muggleton S. *Inverse entailment and Progol.* New Generation Computing Journal, 13:245-286, 1995.
5. van der Laag P. *An Analysis of Refinement Operators in ILP.* PhD Thesis, Tinbergen Inst. Res. Series 102, 1995.
6. van der Laag P., S.H. Nienhuys-Cheng. *Existence and Nonexistence of Complete Refinement Operators.* ECML-94, 307-322.

This article was processed using the LaTeX macro package with LLNCS style