Constructing Refinement Operators by Decomposing Logical Implication

Shan-Hwei Nienhuys-Cheng
cheng@cs.few.eur.nl
Department of Computer Science

Patrick R.J. van der Laag
patrick@cs.few.eur.nl
Tinbergen Institute and
Department of Computer Science

Leendert W.N. van der Torre
torre@cs.few.eur.nl
EURIDIS and
Department of Computer Science

Erasmus University of Rotterdam
P.O.Box 1738, 3000 DR Rotterdam
the Netherlands

September 11, 1995

Abstract

Inductive learning models [14] [17] often use a search space of clauses, ordered by a generalization hierarchy. To find solutions in the model, search algorithms use different generalization and specialization operators. In this article we will decompose the quasi-ordering induced by logical implication into six increasingly weak orderings. The difference between two successive orderings will be small, and can therefore be understood easily. Using this decomposition, we will describe upward and downward refinement operators for all orderings, including θ-subsumption and logical implication.

1 Introduction

It is well known that logical implication can be considered as an ordering on clauses. In this article, three questions are discussed. Each answer will give us a starting point for the next question:

1. How can we weaken the ordering, induced by logical implication?

2. How can we split up logical derivations into simple operations?

3. How can we find generalizations and specializations of a clause?

Logical implication can be described by *resolution.* This will be the starting point of our investigations to answer the first question. Five times we will weaken the ordering induced by logical implication. Every ordering lacks one feature of the former and is less flexible, but more mechanical and manageable. The following example will show this idea.

Example 1.1. The first weakening results in the well known θ-subsumption ordering. Consider

$C = p(f(X)) \leftarrow p(X)$ and
$D = p(f(f(Y))) \leftarrow p(Y)$.

Then C logically implies D but doesn't θ-subsume it. So, in the logical implication ordering C and D are comparable but in the weaker θ-subsumption ordering they are not. □

To answer the second question, we will use our decomposition. If we analyze how the orderings are defined, we will notice that different operations like substitution, permutation and addition of literals are introduced in different orderings.

Example 1.2. Consider

$C = p(f(X)) \leftarrow p(X)$,
$D = p(f(f(X))) \leftarrow p(X)$,
$E = p(f(f(X))) \leftarrow p(X), q(Y)$ and
$F = p(f(f(a))) \leftarrow p(a), q(b)$.

Clause F can be logically derived from clause C. Our analysis will show this by observing there is a resolution step (from C to D), an addition of a literal (from D to E) and a substitution step (from E to F). □

The motivation of the first two questions is to find an answer for the third question. First we will define operators that find refinements (generalizations or specializations) of a clause for the weakest ordering. For each stronger ordering, we split up the new operations into small steps. By extending the operators with these small steps, refinement operators for all orderings are found. Having understood the orderings well, we will show that some of the refinements are redundant. Eventually we will come up with a refinement operator for logical implication.

To summarize, the ordering of logical implication is deconstructed in five steps to substitution. This deconstruction results in the decomposition of logical implication into six basic operations. These operations are used to find refinement operators for all orderings.

1.1 Related work

Questions related to orderings on clauses and refinements have received a lot of attention within machine learning, especially within Inductive Logic Programming (ILP). Our approach of decomposing logical implication to find refinement operators is new. Here we will summarize the other approaches and their motivations.

1.1.1 Orderings

Within inductive learning, logical derivations are used as explanations of examples by a theory: a theory T explains positive E^+ and negative E^- examples iff $T \models E^+$ and $T \not\models E^-$. Machine learning algorithms like the well-known algorithms ID3 and AQ11 construct a theory using the examples. Within Inductive Logic Programming (ILP), people focus on incremental learning which makes the construction of a theory a search process [7]. Using search, it seems natural (although not necessary per se) to stick to a generalization hierarchy with logical implication as ordering.

In this article we focus on orderings on clauses. Search algorithms usually try to find individual clauses of the theory instead of the whole theory at once. This simplifies the generalization hierarchy (which becomes an ordering on clauses only) and search process, but also leads to some problems. Some of the problems were conquered by making the ordering relative to background knowledge. Orderings with background knowledge consisting of ground atoms [14] look like orderings without background knowledge. Orderings with general background knowledge [15] [2] can be translated into constructive operators in the context of inverse resolution [9]. The latest developments [8] [10] are that general background knowledge is translated to background knowledge of ground atoms (called a model of the theory).

Lapointe and Matwin [4] were the first to define a generalization operator for logical implication. Only a restricted set of clauses can be found, but they can be found very efficiently (with a few examples). Their operator consists of two steps. In the first step a recursive clause D' is (implicitly) build from two given clauses D_1 and D_2. In the second step this clause D' is generalized to C.

Example 1.3.[4] Consider

$D_1 = append([], L, L)$,
$D_2 = append([a,b,c],[1,2],[a,b,c,1,2])$,
$D' = append([a,b,c|X], Y, [a,b,c|Z]) \leftarrow append(X,Y,Z)$ and
$C = append([V|X], Y, [V|Z]) \leftarrow append(X,Y,Z)$.

The first step constructs D' which is generalized in the second step to C. □

Muggleton [10] generalizes the second step such that it is possible to find any generalization C of D' such that D' can be derived from C by resolution.

Many people within the ILP community use our second ordering, θ-subsumption, because θ-subsumption is more manipulable than resolution. We can see this idea for example in the Model Inference System of Shapiro [17]. Plotkin [13] introduced θ-subsumption as a kind of explanation. He used this ordering to compute a least general generalization (lgg), which always exists (in contrast to the lgg of logical implication, as was shown by Niblett [11]). The application of lgg was quite restricted so he incorporated background knowledge in the ordering to achieve relative lgg's. However, both the logical implication and subsumption rlgg don't have to exist [11], which prompted Niblett to question the advantages of θ-subsumption over logical implication.

A basic operation of θ-subsumption is substitution, introduced in machine learning for ordering atoms by Reynolds [16]. He showed that computing an lgg of atoms is a kind of dual to unification.

1.1.2 Refinement operators

In [3], Laird has described a general framework for upward and downward refinement operators which can respectively find more general and more specific clauses. (Downward) refinement operators were introduced by Shapiro [17]. In his Model Inference System, refinement operators are used to replace clauses by more specific ones if the theory is too strong. In Ling's system SIM [6], abstraction operators do the opposite if the theory is too weak. Some known refinement operators get more attention in Section 3, where they are related to ours.

2 Decomposing Logical Implication

In this section, we will decompose the ordering on clauses induced by logical implication into six increasingly weak quasi-orderings, $\succeq_6, \ldots, \succeq_1$.

The two first and strongest orderings are the already mentioned logical implication and θ-subsumption. The following three orderings are new. The last and weakest ordering was defined by Plotkin [13] and Reynolds [16] for atoms, but we generalize it to compare clauses.

2.1 Definitions

Definition 2.1. Given a set clauses S and clauses $C, D, E \in S$, we use the following related notions:

- A partially defined binary relation $\geq$ on S is called a *partial ordering* on S iff it is reflexive ($C \geq C$), transitive ($C \geq D$ and $D \geq E$ imply $C \geq E$) and antisymmetric ($C \geq D$ and $D \geq C$ imply $C = D$).

- A partially defined binary relation $\succeq$ on S that is reflexive and transitive but not necessarily antisymmetric is called a *quasi-ordering.*

- If $\succeq_1$ and $\succeq_2$ are two quasi-orderings then $\succeq_2$ is *stronger* than $\succeq_1$ if $C \succeq_1 D$ implies $C \succeq_2 D$. If also for some C, D, $C \not\succeq_1 D$ and $C \succeq_2 D$ then $\succeq_2$ is *strictly stronger* than $\succeq_1$.

- If $C \succeq D$ or $D \succeq C$ then C and D are called *comparable.*

- C *covers* D iff $C \succ D$ ($C \succeq D$ and $D \not\succeq C$) and there exists no E such that $C \succ E \succ D$. If C covers D then C is called an *upward cover* of D, and D is called a *downward cover* of C. We denote the set of all downward and upward covers of a clause C by $dc(C)$ and $uc(C)$.

- For every quasi-ordering $\succeq$ we can define an *equivalence* relation: $C \sim D$ iff $C \succeq D$ and $D \succeq C$.

Within the logical language used in this article there is an explicit distinction between the representation of a clause as a set and as a sequence of literals. This is necessary to describe our new orderings. When we say 'clause C' we mean a sequence of literals:

$C = L_0 \leftarrow L_1, \ldots, L_n$.

The set representation is common in ILP and will, in this article, sometimes be used to facilitate definitions. By writing $\dot{C}$ we mean that clause C is implicitly considered as a set of literals and thus the internal ordering and repetition of literals play no role. For example, the clauses $p(X) \leftarrow q(X), r(X), r(X)$ and $p(X) \leftarrow r(X), q(X)$ have the same set representation $\dot{C} = \{p(X), \neg q(X), \neg r(X)\}$.

Whenever we say clauses we mean Horn clauses. The results of this article however are easily generalized to clauses.

2.2 The Logical Implication and θ-Subsumption Ordering

The logical implication ordering is defined model-theoretically. Niblett [11] showed that logical implication between clauses is decidable, in contrast to logical implication between theories (which is only semi-decidable).

Ordering 6. The *logical implication ordering* $\succeq_6$ is defined by $C \succeq_6 D$ iff $C \models D$. □

To work with this ordering, we need a proof-theoretic counterpart. This was given by Muggleton and Bain (a reproof of Lee [5]) and uses the resolution closure of Robinson.

Definition 2.2. Let T be a set of clauses. The *resolution closure* $\mathcal{L}^*(T)$ is defined by the function $\mathcal{L}$:

1. $\mathcal{L}^0(T) = T$
2. $\mathcal{L}^n(T) = \{C \mid D_1 \in \mathcal{L}^{n-1}(T),\ D_2 \in T$ and C is a resolvent of D_1 and $D_2\}$
3. $\mathcal{L}^*(T) = \mathcal{L}^0(T) \cup \mathcal{L}^1(T) \cup \ldots$

Theorem 2.3.[1, Theorem 7] $C \succeq_6 D$ iff D is a tautology or there is a $D' \in \mathcal{L}^*(\{C\})$ and a substitution θ such that $\dot{D}'\theta \subseteq \dot{D}$. □

This ordering can be naturally divided into two parts: the construction of D' using resolution and the derivation of D from D'. Since C only resolves with C itself or with one of its resolvents, this is called *self-resolution* by Muggleton [10].

If no resolution steps are applied in the logical implication ordering, all that rests is θ-subsumption.

Ordering 5. The *θ-subsumption ordering* $\succeq_5$ is defined by $C \succeq_5 D$ iff $\dot{C}\theta \subseteq \dot{D}$ for some substitution θ. □

The difference between θ-subsumption and logical implication is characterized exactly by the operations involving self-resolution [10]. The example in Section 1 showed that logical implication is strictly stronger than θ-subsumption.

To describe the equivalence classes, we have to discriminate between tautologies and other clauses. A clause is a tautology iff it contains a literal and its negation. Tautologies are more specific than any other clause and are only equivalent with other tautologies. For non-tautologies, we can use the definition of reduction, introduced by Plotkin [13]:

Definition 2.4.

- A clause C is *reduced* iff $\dot{D} \subseteq \dot{C}$ and $D \sim_5 C$ imply $\dot{D} = \dot{C}$.

In words, a clause is reduced iff it is not equivalent to a proper subset of itself when regarded as a set. If a clause C is not reduced, Plotkin's reduction algorithm returns a reduced clause D such that $\dot{D} \subset \dot{C}$. We call all literals in $\dot{C} \setminus \dot{D}$ *redundant.* Plotkin proved the following propositions.

Proposition 2.5.[15] $C \sim_6 D$ iff both C and D are tautologies or after reduction $\dot{C}$ and $\dot{D}$ are renamings. $\square$

Proposition 2.6.[13] $C \sim_5 D$ iff after reduction $\dot{C}$ and $\dot{D}$ are renamings. $\square$

Self-resolution with a reduced clause C does not produce equivalent clauses unless C is a tautology. From the previous propositions, it follows that addition of non-redundant literals to reduced non-tautologies as well as applying substitutions that are not renamings result in proper specializations in the logical implication ordering.

In the two following sections, we will weaken θ-subsumption, $\dot{C}\theta \subseteq \dot{D}$, to $\dot{C}\theta = \dot{D}$, in two steps.

2.3 The Restricted θ-Subsumption Ordering

Example 2.7. Consider
$C = p(X) \leftarrow q(f(X))$,
$D = p(X) \leftarrow q(f(X)), q(Y)$,
$E = p(X) \leftarrow q(f(X)), q(g(V))$ and
$F = p(X) \leftarrow q(f(X)), r(W)$.
Then clearly $C \succeq_5 D$, $C \succeq_5 E$ and $C \succeq_5 F$. By $\theta = \{Y/f(X)\}$, we also have $D \succeq_5 C$, but there is no substitution that maps $q(g(V))$ in E or $r(W)$ in F to a literal of C. Therefore, $C \sim_5 D$, $C \succ_5 E$ and $C \succ_5 F$. $\square$

Definition 2.8.

- Two literals are called *compatible* iff they have the same predicate symbol and sign.

It can be verified that adding to a clause C a literal L that is incompatible with every literal in C always results in a proper specialization ($\succ_5$). If L is compatible with a literal in C, then the resulting clause is equivalent iff L is redundant or already in C. In the new ordering we exclude the case of addition of incompatible literals:

Ordering 4. In the *restricted θ-subsumption ordering* $\succeq_4$, $C \succeq_4 D$ iff there exists a substitution θ such that $\dot{C}\theta \subseteq \dot{D}$ and every literal in D is compatible with a literal in C. $\square$

Revisiting the last example, we see that in the restricted θ-subsumption ordering, $C \sim_4 D$ and $C \succ_4 E$ hold like in $\succeq_5$, but C and F have become incomparable. So $\succeq_5$ is strictly stronger than $\succeq_4$.

If $C \sim_5 D$, then C and D cannot contain a literal that is incompatible with every literal in the other clause. Therefore, equivalence in the θ-subsumption and restricted θ-subsumption ordering amounts to the same.

Proposition 2.9. $C \sim_4 D$ iff after reduction $\dot{C}$ and $\dot{D}$ are renamings. $\square$

2.4 The Set Ordering

Ordering 3. In the *set ordering* $\succeq_3$, $C \succeq_3 D$ iff there exist a substitution θ such that $\dot{C}\theta = \dot{D}$. $\square$

Example 2.10. Consider
$C = p(X) \leftarrow q(X,Y), q(Y,Z), q(Z,X)$,
$D = p(X) \leftarrow q(X,Y), q(Y,X), q(X,X)$ and
$E = p(X) \leftarrow q(X,X)$.
In the restricted θ-subsumption ordering we have $C \succ_4 D \sim_4 E$. In the set ordering, D and E are no longer equivalent since the literal $q(X,X)$ in E cannot be mapped to more than one literal in D. We have $C \succ_3 D \succ_3 E$. $\square$

Proposition 2.11.[12] $C \sim_3 D$ iff $\dot{C}$ and $\dot{D}$ are renamings. $\square$

We can also express this proposition without the use of the set-notation of clauses. For this we need the following definition:

Definition 2.12.

- *Set reduction* of a clause is the removal of all duplicate literals in it. A clause is *set reduced* iff it contains no duplicate literals.

Thus, clauses remain equivalent after set reduction ($\sim_3, \ldots, \sim_6$). Clauses are equivalent in the set ordering iff their set reduced equivalents are permuted renamings.

In the two following sections, the set properties of the orderings are removed. Firstly the introduction (and removal) of duplicate literals and secondly permutation is prohibited.

2.5 The Permutation Ordering

Ordering 2. In the *permutation ordering* $\succeq_2$, if $C = L_0 \leftarrow L_1, \ldots, L_m$ and $D = M_0 \leftarrow M_1, \ldots, M_n$, then $C \succeq_2 D$ iff $m = n$ and there exist a permutation π of $\{1, \ldots, m\}$ and a substitution θ such that $L_0\theta = M_0$ and $L_{\pi(i)}\theta = M_i$, $i = 1, \ldots, m$. $\square$

Example 2.13. Consider
$C = p(X) \leftarrow q(X,Y), q(Y,X)$ and
$D = p(X) \leftarrow q(X,X)$.
In the set ordering, $C \succ_3 D$. Since the number of literals in C and D differ, C and D are incomparable in the permutation ordering. $\square$

From the definition of $\succeq_2$ it follows directly that addition of literals results in incomparable clauses. Furthermore,

Proposition 2.14.[12] $C \sim_2 D$ iff C and D are permuted renamings. $\square$

From this proposition it follows that in the permutation ordering, in order to find proper specializations, permutations are useless and only substitutions that are not renamings are relevant.

2.6 The Substitution Ordering

To obtain the last, weakest and simplest ordering, we remove the free exchange of positions of literals from the permutation ordering.

Ordering 1. In the *substitution ordering* $\succeq_1$, if $C = L_0 \leftarrow L_1, \ldots, L_m$ and $D = M_0 \leftarrow M_1, \ldots, M_n$, then $C \succeq_1 D$ iff $m = n$ and there exists a substitution θ such that $L_i\theta = M_i$, $i = 0, \ldots, m$. □

Example 2.15. Consider
$C = p(X) \leftarrow q(X,Y), r(Y,X)$ and
$D = p(X) \leftarrow r(X,X), q(X,X)$.
In the permutation ordering C and D are comparable such that $C \succ_2 D$. In the substitution ordering C and D are no longer comparable. □

This ordering is a generalization of Reynolds' [16] ordering on atoms. Clauses are only comparable iff there exists a substitution such that every literal in one clause is mapped onto the corresponding literal in the other clause. The following proposition is a direct generalization of Lemma 1. in [16].

Proposition 2.16. $C \sim_1 D$ iff C and D are renamings. □

As a consequence of the definition of $\succeq_1$ and this proposition, substitutions that are not renamings result in proper specializations.

2.7 Summary

In this section we have defined six increasingly weak orderings on clauses, by deleting generality relations at each stage. Every ordering lacks one feature of the former, it is less flexible but more mechanical and manageable. The whole decomposition gives us a clearer view on logical implication and θ-subsumption.

A side-effect of this approach is that along the decomposition steps the number of equivalent clauses decreases. Equivalence classes are partitioned into smaller equivalence classes of weaker orderings.

Now we are ready to answer the second question of this article: How can we split up logical derivations into simple operations? To answer this question we approach the orderings from weak to strong. From the substitution ordering to the restricted θ-subsumption ordering, equivalence classes melt together by 'recognizing' equivalent clauses that differ in the operations permutation, repetition of literals and addition of redundant literals respectively. In these first three strengthenings substitutions determine the comparability of clauses; if $C \succeq_i D$ then there exists $C' \sim_i C$ and $D' \sim_i D$ such that $C' \succeq_1 D'$, $i = 2, 3, 4$. In the last two strengthenings, two new operations are involved in comparability: in the θ-subsumption ordering the addition of incompatible literals and in the logical implication ordering self-resolution.

The last question of this article – how to find generalizations and specializations – will be answered in the next section, using this decomposition. For every ordering, the results of the weaker orderings can be used and only the new operation has to be examined.

3 Constructing Refinement Operators

Refinement operators can be used to find specializations and generalizations of clauses. If in the ordering $\succeq$ the relation $C \succ D$ holds, then a clause equivalent with D can be derived from C and vice versa by repeatedly applying downward and upward refinement operators respectively.

3.1 Definitions

The following definitions are related to refinement operators.

Definition 3.1. Given a set of clauses S, some quasi-ordering $\succeq$ on S and clauses $C, D \in S$, we use the following related notions:

- For every C, $\{D | C \succeq D\}$ is the set of *downward refinements* of C. If $C \succeq D$ and $D \not\succeq C$ then D is a *proper downward refinement* of C. (Proper) upward refinements are defined dually.
- A *downward (upward) refinement operator* ρ (δ) is a mapping defined on S such that for every $C \in S$, $\rho(C)$ ($\delta(C)$) is a subset of the downward (upward) refinements of C.

The terminology of downward and upward refinements is adopted from Laird [3]. Shapiro's refinement operators [17] are downward refinement operators, and Ling's abstraction operators [6] are upward refinement operators.

The following definitions are in terms of downward refinement operators ρ but hold similarly for upward refinement operators δ.

Definition 3.2. Let ρ be a downward refinement operator, then

- $\rho^0(C) = \{C\}$ and $\rho^n(C) = \{D | \exists E \in \rho^{n-1}(C)$ such that $D \in \rho(E)\}$.
- $\rho^*(C) = \bigcup_{i=0}^{\infty} \rho^i(C)$

3.2 Complete refinement operators

Definition 3.3. Let ρ (δ) be a downward (upward) refinement operator for clauses ordered by $\succeq$. Then

- ρ (δ) is called *complete for the ordering* $\succeq$ iff for every pair of clauses C, D, if $C \succ D$ then $\exists E$: $E \in \rho^*(C)$ and $E \sim D$ ($\exists F$: $F \in \delta^*(D)$ and $F \sim C$).
- ρ is called *downward cover complete* iff $\forall D \in dc(C) \exists E : E \in \rho(C)$ and $E \sim D$. *Upward cover completeness* is defined dually.

It is easy to see that if ρ is complete for $\succeq$, then for any C, $\rho^*(C)$ must contain equivalents of all downward covers of C. If ρ is complete and returns proper refinements only, then ρ returns all these covers (may return more) in one refinement step, i.e., completeness implies cover completeness. The reverse however does not hold. All refinement operators in the rest of this section will be cover complete. In Section 3.9 we wil discuss the relations among (cover) completeness and restricting the search space.

Remark. Our notion of completeness differs from Shapiro's [17]. His definition of 'global' completeness for the downward case, $\rho^*(\Box) = S$ where $\Box$ denotes the empty clause, only describes derivability of clauses from $\Box$.

3.3 The Substitution Ordering

In the substitution ordering, substitutions that are not renamings determine proper refinements. We will define a downward refinement operator ρ_1 that divides this operation in smallest steps, using Reynolds' cover-relation for atoms. By Theorem 4 of Reynolds [16], ρ_1 returns exactly all downward covers. δ_1 is obtained by inverting the substitution of ρ_1, and returns exactly the upward covers.

Refinement operators 1. Let C be a clause, then

- $D \in \rho_1(C)$ iff one of the following holds:
  1. $D = C\theta$, where $\theta = \{Y/X\}$, $X \neq Y$ and both X and Y occur in C.
  2. $D = C\theta$ where $\theta = \{X/c\}$, c is a constant symbol and X occurs in C.
  3. $D = C\theta$ where $\theta = \{Y/f(X_1, \ldots, X_n)\}$, f is a n-ary function symbol, Y occurs in C and all X_i's are distinct variables not occurring in C.
- $D \in \delta_1(C)$ iff one of the following holds:
  1. D is C after some (not all) occurrences of a variable X in C are replaced by a variable Y not in C.
  2. D is C after some or all occurrences of a constant c are replaced by a variable X not in C.
  3. D is C after all occurrences of $f(X_1, \ldots, X_n)$ are replaced by a variable Y, where f is a n-ary function symbol, Y does not occur in C and all X_i's are distinct variables not occurring elsewhere in C besides in terms $f(X_1, \ldots, X_n)$.

The downward refinement operator ρ_1 corresponds with Shapiro's [17] refinement operator for atoms (also named ρ_1).

In [6], Ling describes an upward refinement operator for atoms. This so-called abstraction operator is defined as follows:

- $D \in \delta_L(C)$ iff one of the following holds:
  1. D is C after some or all occurrences of a constant c are replaced by variable X not in C.
  2. D is C after some or all occurrences of a compound term $f(t_1, \ldots, t_n)$ are replaced by a variable Y not in C.

The omission of anti-unification of variables (first part of δ_1) implies that some proper upward refinements cannot be derived by δ_L. E.g., $p(X, Y)$ cannot be derived from $p(X, X)$. However, all generalizations of ground atoms can be derived.

In contrast with δ_1, δ_L returns clauses that are not covers.

Example 3.4. Consider

$C_i = p(f^i(X)),\ i \geq 0.$

Then C_i covers C_{i+1}, and in our approach $\delta_1(C_{i+1}) = \{C_i\}$. In Ling's approach, the number of upward refinements depends on i: $\delta_L(C_{i+1}) = \{C_j | 0 \leq j \leq i\}$. $\square$

3.4 The Permutation Ordering

If D covers C in the substitution ordering, we may expect that D is no longer a cover in the permutation ordering because $\succeq_2$ is stronger than $\succeq_1$, i.e., there may exists a clause E such that $D \succ_2 E \succ_2 C$. However, this cannot happen as is shown in [12] by Nienhuys-Cheng. It is proved that D is a cover of C in the permutation ordering iff we can find a clause $D' \sim_2 D$ such that D' is a cover of C in the substitution ordering. From this it follows that ρ_1 and δ_1 also return all covers for the permutation ordering.[1]

Refinement operators 2. Let C be a clause, then

- $D \in \rho_2(C)$ iff $D \in \rho_1(C)$.
- $D \in \delta_2(C)$ iff $D \in \delta_1(C)$.

Example 3.5. Consider

$C = p(X) \leftarrow q(Y), r(Y)$ and
$D = p(X) \leftarrow r(X), q(X)$.

$C \succ_2 D$, but D is not derivable from C by applying ρ_2. However, if we define

$D' = p(X) \leftarrow q(X), r(X),$

then $D' \sim_2 D$ and $D' \in \rho_2(C)$. $\square$

3.5 The Set Ordering

In the set ordering as well as in all stronger orderings, clauses may be regarded as sets of literals.

Example 3.6. We revisit the clauses of Example 2.13,

$C = p(X) \leftarrow q(X,Y), q(Y,X)$ and
$D = p(X) \leftarrow q(X,X)$.

In the permutation ordering C and D were incomparable, but $C \succ_3 D$. We cannot derive C from D using δ_2. However, if we define

$D' = p(X) \leftarrow q(X,X), q(X,X)$

then $D' \sim_3 D$ and $C \in \delta_2(D')$. This motives our definition of δ_3 later on. $\square$

In the case of downward refinement operators, equal literals are of no use since equal literals remain equal after substitution. Hence there is no need to duplicate literals at any time and they can be removed as soon as they appear.

As the last example showed for the case of upward refinement, literals sometimes should be repeated before inverse substitutions can be applied. We can easily define an operator that repeats a literal:

Definition 3.7. Let $C = L_0 \leftarrow L_1, \ldots, L_n$ be a clause, then

[1]Remember that only one clause of an equivalence class has to be reached.

- $D \in eq_3(C)$ iff $D = L_0 \leftarrow L_1, \ldots, L_i, L_i, \ldots, L_n$ for some $i = 1, \ldots, n$.

By repeatedly applying eq_3, $eq_3^*(C)$ contains infinitely many clauses $C' \sim_3 C$ such that
$C' = L_0 \leftarrow L_1, \ldots, L_1, L_2, \ldots, L_2, \ldots, L_n, \ldots, L_n$.

Refinement operators 3. Let C be a set reduced clause, then

- $D \in \rho_3(C)$ iff there is a $D' \in \rho_1(C)$ and D is the set reduced equivalent of D'.
- $D \in \delta_3(C)$ iff there are $C' \in eq_3^*(C)$, $D \in \delta_1(C')$ and D is set reduced.

Note that all clauses D that are found by δ_3 are proper refinements. To obtain set reduced upward refinements D, if a literal occurs k times in a clause C' then at least $k-1$ occurrences are changed by δ_1.

The following example shows that ρ_3 and δ_3 return also clauses that are not covers:

Example 3.8. Consider
$C = q(c) \leftarrow p(f(a)), p(b), p(Y)$,
$D = q(c) \leftarrow p(f(a)), p(f(X)), p(b)$ and
$E = q(c) \leftarrow p(f(a)), p(b)$.

Then $C \succ_3 D \succ_3 E$, so E is not a downward cover of C and C is not a upward cover of E. However, $E \in \rho_3(C)$ and $C \in \delta_3(E)$. □

3.6 The Restricted θ-Subsumption Ordering

Example 3.9. We revisit the clauses of Example 2.10,
$C = p(X) \leftarrow q(X,Y), q(Y,Z), q(Z,X)$,
$D = p(X) \leftarrow q(X,Y), q(Y,X), q(X,X)$ and
$E = p(X) \leftarrow q(X,X)$.

In the set ordering, $C \succ_3 D \succ_3 E$, $D \in \delta_3(E)$, $C \in \delta_3(D)$ and both upward refinements step generate proper refinements.

In the restricted θ-subsumption ordering, D and E are equivalent, since the literals $q(Y,X)$ and $q(X,Y)$ are redundant in D. We can now derive C from E in two ways. If we allow $\delta_4(E)$ to contain clauses $E'(\sim_4 E)$, then we can use δ_3 unmodified as δ_4 and $D \in \delta_4(E)$. If we want proper refinements only, then, for defining δ_4, we must add the redundant literals before applying δ_3. Since proper refinements are simpler to handle, we choose the second approach. □

ρ_3 cannot derive all proper downward refinements of the restricted θ-subsumption ordering.

Example 3.10. Consider
$C = p(X) \leftarrow q(X,Y)$ and
$D = p(X) \leftarrow q(X,Y), q(U,V), q(V,W)$.

$C \succ_4 D$, but by substitutions and removal of duplicate literals, D cannot be derived from C. If we first add the redundant literals $q(U,V)$ and $q(Z,W)$ to C, then D can be derived by ρ_3. □

In [19], an inverse reduction algorithm is presented. Given a reduced clause C and a bound n, this algorithm returns all clauses that contain C and has at most n redundant literals. By $eq_4(C)$ we denote the operation of adding one redundant literal to C. By repeatedly applying eq_4, $eq_4^*(C)$ contains all clauses that contain C and arbitrary many redundant literals.

Refinement operators 4. Let C be a reduced clause, then

- $D \in \rho_4(C)$ iff there are $C' \in eq_4^*(C)$, $D' \in \rho_3(C')$ such that $D' \not\sim_4 C'$ and D is the reduced equivalent of D'.
- $D \in \delta_4(C)$ iff there are $C' \in eq_4^*(C)$, $D' \in \delta_3(C')$ such that $D' \not\sim_4 C'$ and D is the reduced equivalent of D'.

All clauses of Example 3.5 are equivalent in the restricted θ-subsumption ordering, and can not be found with these refinement operators because of the equivalence check. Still ρ_4 and δ_4 return also clauses that are not covers:

Example 3.11. Consider
$C = p(X) \leftarrow q(X,Y), q(Y,Z), q(Z,W), q(W,X)$,
$D = p(X) \leftarrow q(X,Y), q(Y,X)$ and
$E = p(X) \leftarrow q(X,X)$.
Then $C \succ_4 D \succ_4 E$, so E is not a downward cover of C and C is not a upward cover of E. However, $E \in \rho_4(C)$ and $C \in \delta_4(E)$. □

3.7 The θ-Subsumption Ordering

In [19], a downward refinement operator for the θ-subsumption ordering is described in detail. It differs from ρ_4 in the operation of adding incompatible literals.
Refinement operators 5. Let C be a reduced clause, then

- $D \in \rho_5(C)$ iff $D \in \rho_4(C)$ or

  D is C after a literal that is incompatible with every literal in C is added. This literal has only new and distinct variables as arguments.
- $D \in \delta_5(C)$ iff $D \in \delta_4(C)$ or

  D is C after a literal that is incompatible with every other literal in C is removed. This literal has only distinct variables as arguments that do not occur in elsewhere in C.

Example 3.12. We revisit the clauses C and F of Example 2.7,
$C = p(X) \leftarrow q(f(X))$ and
$F = p(X) \leftarrow q(f(X)), r(W)$.
C and F have become comparable such that $C \succ_5 F$ and $F \in \rho_5(C)$ and $C \in \delta_5(F)$. □

For the θ-subsumption ordering, a number of refinement operators are known. Shapiro[17] has defined a downward refinement operator (ρ_0) for reduced first order clauses, Laird did the same [3] for not necessarily reduced first order clauses. In [18] and [19] we have shown that Shapiro's ρ_0 cannot derive all reduced clauses, as he claimed. We also proved in [18] that Laird's refinement operator is complete. A difference between Laird's downward refinement operator and ρ_5 is that Laird allows the addition of literals that are compatible with a literal already in the clause. Thus redundant literals are not added explicitly and the separation between equivalent and proper refinement steps is lost.

Ling [6] has described an upward refinement operator for clauses, probably intended for the θ-subsumption ordering. Clauses are treated as atoms in the substitution ordering (see δ_L in Section 3.3), with one addition, deletion of arbitrary literals. Clearly, there is no way that Ling's operator can derive the clause $p(X) \leftarrow q(X,Y), q(Y,X)$ from the clause $p(X) \leftarrow q(X,X)$ since no literals can be added. In his learning system SIM this causes no problems. If a clause D should be derived from a clause C then it is assumed that C contains a ground instantiations of D. Furthermore C has different ground terms at the positions where D has different variables.

3.8 The Logical Implication Ordering

It was noted before that Muggleton's [10] so-called n-th powers and n-th roots of clauses characterize exactly the difference between θ-subsumption and logical implication. If $D \in \mathcal{L}^n(\{C\})$ then D is called an *n-th power* of C. Conversely, C is called an *n-th root* of D. Although these operations were described in an inverse resolution context, since they describe implication relations between clauses, they can also be regarded as refinement operators.

Refinement operators 6. Let C be a reduced clause, then

- $D \in \rho_6(C)$ iff $D \in \rho_5(C)$ or
  D is the reduced equivalent of D' for some $D' \in \mathcal{L}^n(\{C\})$, $n \geq 1$
- $D \in \delta_6(C)$ iff $D \in \delta_5(C)$ or
  D is the reduced equivalent of D' for some $C \in \mathcal{L}^n(\{D'\})$, $n \geq 1$

Remark. The naming powers and roots might suggest that self-resolution satisfies that all n^m'th powers are m'th powers of n'th powers. This is true when there is just one negative literal in the clause that can be unified with the positive literal, but not in the general case. The following is a counterexample, it implies non-associativity of self-resolution.

Example 3.13. Let C be the recursive clause

$C = p(X) \leftarrow p(f(X)), p(g(X))$.

Then C can be resolved with itself in two ways, one for each literal in the body of C. The resulting 2-powers are:

$C_1^2 = p(X) \leftarrow p(g(X)), p(f(f(X))), p(g(f(X)))$
$C_2^2 = p(X) \leftarrow p(f(X)), p(f(g(X))), p(g(g(X)))$

Resolving C with C_1^2 on the literals $p(f(X))$ and $p(X)$ respectively yields a 3-power of C, C_1^3:

$C_1^3 = p(X) \leftarrow p(g(X)), p(g(f(X))), p(f(f(f(X)))), p(g(f(f(X))))$

Resolving C with C_1^3 on the literals $p(g(X))$ and $p(X)$ respectively yields a 4-power of C, C_1^4:

$C_1^4 = p(X) \leftarrow p(f(X)), p(g(g(X))), p(g(f(g(X)))), p(f(f(f(g(X))))), p(g(f(f(g(X)))))$

Now C^4 is a 4-power of C, but it is not a 2-power of one of C's 2-powers, C_1^2 and C_2^2. $\square$

3.9 Restricting the Search Space

Two problems with refinement operators have not been discussed. We will solve these problems by restricting the search space.

1. Some refinement operators are not locally finite.

2. Cover complete operators don't have to be complete.

An operator is *locally finite* iff it returns finitely many clauses in finite time. It follows directly from the definitions of ρ_1, ρ_2, δ_1, δ_2 and ρ_3 that they are locally finite. We can easily make δ_3 locally finite because we can prove that the number of non-equivalent covers in the set ordering is finite, hence we need only a finite part of eq_3^*. To define locally finite refinement operators for the other orderings, operators that refine a clause C can be allowed to return clauses equivalent with C. E.g., by defining $D \in \delta_4(C)$ iff $D \in \delta_3(C)$.

We can also obtain locally finite refinement operators by restricting the search space. Using a restricted search space S, we can easily see that all refinement operators are locally finite because we consider only the intersection of $eq_3^*(C)$ and $eq_4^*(C)$ with S.

In Section 3.2 we have shown that cover completeness is a necessary condition for complete refinement operators that return proper refinements only. It is, however, not a sufficient condition. Problems arise if $C \succ D$ and there exists an infinite chain of proper refinements $C \succ C_1 \succ C_2 \succ \ldots$ such that every C_i satisfies $C_i \succ D$. Then, if we keep on refining C_i's, D will never be derived.

The following example is borrowed from Tim Niblett (personal communication).

Example 3.14. Consider

$C_n = p(X_1) \leftarrow q(X_1, X_2), \ldots, q(X_{n-1}, X_n), q(X_n, X_{n+1})$ and
$D_n = p(X_1) \leftarrow q(X_1, X_2), \ldots, q(X_{n-1}, X_n), q(X_n, X_1)$.

Then, in the θ-subsumption ordering, $C_i \succ_5 C_{i+1}$, $C_{2^i} \succ_5 D_{2^i}$ and $D_{2^{i+1}} \succ_5 D_{2^i}$. Between C_2 and D_2 we can find the following infinite chain:

$C_2, C_3, \ldots, C_i, C_{i+1}, \ldots, D_{2^m}, D_{2^{m-1}}, \ldots, D_4, D_2$

such that $C_{i+1} \in \rho_5(C_i)$. □

Since C_{i+1} is only one of the downward refinements of C_i, this example does not imply that ρ_5 is not a complete refinement operator. Since $D_2 \in \rho_5(C_2)$ holds, we have a finite chain of refinements from C_2 to D_2. Completeness proofs of ρ_5 can be found in [18] and [19].

Assuming that no finite chain of covers from C_2 to D_2 exists, cover completeness is not sufficient for completeness. However, if we limit our search to a finite set of clauses S, then S cannot contain infinitely long chains of proper refinements.

Definition 3.15.

- Let S be a finite set of clauses and $\succeq$ a quasi-ordering, then $\succeq_S$ denotes the quasi-ordering $\succeq$ restricted to S.

Note that the set of covers induced by $\succeq_S$ is not necessarily a subset of the set of covers in $\succeq$. E.g., if we consider the (small) search space $S = \{C = p(X) \leftarrow, D = p(f(a)) \leftarrow\}$ then, for all six orderings restricted to S, C has become an upward cover of D.

For finite search spaces S ordered by $\succeq_S$, cover completeness is a necessary and sufficient condition for complete refinement operators that return proper refinements only.

In [19] we have introduced a new complexity measure to bound S. We have proved that ρ_5 is complete for every fixed bound.

4 Conclusions and Future Research

In this article we have decomposed logical implication into six increasingly weak quasi orderings. The restricted θ-subsumption, set and permutation ordering are new. Another, the substitution ordering, is new in its usage. We think that they help to clarify properties of θ-subsumption as well as the logical implication.

By reversing the decomposition, and looking at the small differences with the former weaker orderings, we were able to incrementally describe upward and downward refinement operators for all orderings including θ-subsumption and logical implication.

The results of the decomposition of logical implication are subject of further research. We are presently looking at least general generalizations (lgg's) of sets of clauses. Following Plotkin's definition, lgg's are unique if they exist, e.g. in the substitution and (restricted) θ-subsumption ordering. We want to loose the requirement of uniqueness and consider sets of incomparable minimal generalizations. So far, we can compute these sets for all other orderings except logical implication.

References

[1] M. Bain and S.H. Muggleton. Non-monotonic Learning. *Machine Intelligence*, 12, 1991.

[2] W. Buntine. Generalised Subsumption and its Applications to Induction and Redundancy. *Artificial Intelligence*, 36(2):149–176, 1988.

[3] P.D. Laird. *Learning from Good and Bad Data.* Kluwer Academic Publishers, 1988.

[4] S. Lapointe and S. Matwin. Subunification: A Tool for Efficient Induction of Recursive Programs. Technical Report TR-92-01, Department of Computer Science, University of Ottawa, Ottawa, Ontario, Canada., 1992.

[5] C. Lee. *A completeness theorem and a computer program for finding theorems derivable from given axioms.* PhD thesis, University of California, Berkely, 1967.

[6] C. Ling and M. Dawes. SIM the Inverse of Shapiro's MIS. Technical report, Department of Computer Science, University of Western Ontario, London, Ontario, Canada., 1990.

[7] T.M. Mitchell. Generalization as Search. *Artificial Intelligence*, 18:203–226, 1982.

[8] S. Muggleton and C. Feng. Efficient Induction of Logic Programs. Technical Report TIRM-90-044, The Turing Institute, October 1990.

[9] S.H. Muggleton. Inductive logic programming. In *First Conference on Algorithmic Learning Theory*, Ohmsha, Tokyo, 1990. Invited paper.

[10] S.H. Muggleton. Inverting Logical Implication. preprint, 1992.

[11] T. Niblett. A Study of Generalisation in Logica Programs. In *EWSL-88*, pages 131–138. Sigma Press, Wilmslow, England, 1988.

[12] S.H. Nienhuys-Cheng. Generalization and Refinement. Technical report, Erasmus University Rotterdam, Dept. of Computer Science, August 1992. Preprint.

[13] G.D. Plotkin. A Note on Inductive Generalization. *Machine Intelligence*, 5:153–163, 1970.

[14] G.D. Plotkin. A Further Note on Inductive Generalization. *Machine Intelligence*, 6:101–124, 1971.

[15] G.D. Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, Edinburgh, August 1971.

[16] J.C. Reynolds. Transformational Systems and the Algebraic Structure of Atomic Formulas. *Machine Intelligence*, 5:135–153, 1970.

[17] E.Y. Shapiro. Inductive Inference of Theories from Facts. Technical Report 624, Department of Computer Science, Yale University, New Haven. CT., 1981.

[18] P. Van der Laag. Een Meest Algemene Verfijningsoperator voor Gereduceerde Zinnen. In *NAIC-92*, pages 29–39. Delftse Universitaire Pers, 1992. In Dutch, English version has appeared as Technical Report EUR-CS-92-03, Erasmus Univerity of Rotterdam, Dept.of Computer Science.

[19] P.R.J. Van der Laag and S.H. Nienhuys-Cheng. Subsumption and Refinement in Model Inference. In *ECML-93*, 1993. To appear.