θ -subsumption for Structural Matching

Luc De Raedt¹ and Peter Idestam-Almquist² and Gunther Sablon¹

¹ Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium

² Department of Computer and Systems Sciences, Stockholm University
Electrum 230, S-164 40 Kista, Sweden

Abstract. Structural matching, originally introduced by Steven Vere, implements and formalizes the notion of a “most specific generalisation” of two productions, possibly in the presence of a background theory. Despite various studies in the mid-seventies and early eighties, several problems remained. These include the use of background knowledge, the non-uniqueness of most specific generalisations, and handling in-equalities. We show how Gordon Plotkin’s notions of “least general generalisation” and “relative least general generalisation” defined on clauses can be adapted for use in structural matching such that the remaining problems disappear. Defining clauses as universally quantified disjunctions of literals and productions as existentially quantified conjunctions of literals, it is shown that the lattice on clauses imposed by θ -subsumption is order-isomorphic to the lattice on productions needed for structural matching.

1 Introduction

In the early seventies, Gordon Plotkin [18, 19], introduced a notion of θ -subsumption between different clauses (i.e., universally quantified disjunctions of literals). In 1971, Plotkin [20] extended θ -subsumption into relative θ -subsumption with regard to a background theory. Now, θ -subsumption and relative θ -subsumption are the basic notions of generality employed by many inductive logic programming (ILP) systems, cf., e.g., Golem [16], Claudien [3], FOIL [21].

In the mid seventies, inspired by Plotkin, Steven Vere [23] studied the problem of generality between productions (i.e., existentially quantified conjunctions of literals). More specifically, he analyzed the issues involved in computing maximally specific generalizations of productions. In later work [24, 25], this notion was extended in order to enable the use of background knowledge. The notion of a maximally specific generalization is of central importance to the field of concept-learning, as many concept-learning systems repeatedly compute the maximally specific generalizations of their examples. Despite the fact that the popular field of inductive logic programming employs clauses (and Vere employs productions), we believe that productions are still a powerful and interesting representation for concept-learning. This is confirmed by the fact that several recent learning systems still employ productions (e.g. [1, 7]). Furthermore, the inductive logic programming setting known under the name of *learning from interpretations* [5, 4] seems closely related to using productions.

Vere's work was important and influential, as it was continued by many other researchers such as [6, 26, 7, 8]. Nevertheless, there are at least two remaining problems:

First, there is the open question as to how this line of research on structural matching is related to the earlier work on least general generalisation by Gordon Plotkin. Our first contribution is to answer this open question by showing that these two approaches are actually each others duals. This is realized by adapting Plotkin's original notions of θ -subsumption and relative θ -subsumption for use to Vere's problem. In particular, we explicitly give an isomorphism between Plotkin's approach and structural matching. This in turn shows – for the first time – how the field of inductive logic programming relates to that of structural matching.

Secondly, there were several limitations of Vere's work and that of his followers, that are not present in Plotkin's work: One limitation is that Vere only considered constants and variables in productions, and not the more general functors, hence, Vere considered only a subset of first order logic. Another major problem with Vere's work and that of his followers is that the maximally specific generalization of two productions is not necessary unique³. Vere [23] formulates this control problem as follows :

Because a generalization replaces just two products, and a number of distinct generalizations may exist, there remains the question of strategies in the application of the induction procedure in an iterative manner to a large number of positive products.

To see that this is problematic from an efficiency point of view, consider the classical concept-learning task where the aim is to find a single production that generalizes a given set of positive and negative examples. In order to decide whether a solution exists to this problem one has to consider all alternative maximally specific generalizations of the set of positive examples and to test whether these are consistent with the negatives. If the maximally specific generalizations are not unique, this corresponds to testing (in the worst case) an exponential number of alternative generalizations. If on the other hand, the maximally specific generalization of any two productions is unique, only a single generalization needs to be computed and tested. This should be much more efficient.

As an immediate consequence of this problem, the programs developed by Vere and his followers all included a number of heuristics to choose among competing maximally specific generalizations. These heuristics were often mixed with the logical aspects of generalization (as e.g. in [6], who use the number of variables in a production as a heuristic). In contrast, our work will allow to separate the logical from the control (or heuristic) aspects of generalization.

³ The main reason for this is that rather than using normal substitutions to define generality, Vere employs inverse substitutions. Within inductive logic programming, it is well-known that inverse substitutions are much harder to deal with than the classical deductive substitutions, mainly because of their non-uniqueness.

At a more general level, the mapping between the representations and operations employed in structural matching and ILP makes the relation between these two techniques – for the first time – explicit. From this point of view, our work can be regarded as a rational reconstruction of structural matching in (inductive) logic programming terminology.

The paper is organised as follows: in Section 2, we introduce the logical concepts needed; in Section 3, we briefly summarize the main characteristics of θ -subsumption and "least general generalisation"; in Section 4, we outline how this notion can be mapped onto structural matching; in Sections 5 and 6, we generalize the results of Sections 5 and 6 to relative θ -subsumption and "relative least generalisation"; finally, in Section 7, we conclude.

2 Logic programming concepts

We first outline some standard logic programming concepts.

A first-order alphabet is a set of variables, functor symbols, and predicate symbols. An atom $p(t_1, \dots, t_n)$ is a predicate symbol p followed by a bracketed n -tuple of terms. A positive literal is an atom, a negative literal is a negated atom. A term t is a variable V or a functor symbol $f(t_1, \dots, t_n)$ immediately followed by a bracketed n -tuple of terms. Constants are functor symbols of arity 0.

Definition 1. A *clause* is a finite disjunction of literals closed under universal quantification $\forall(l_1 \vee \dots \vee l_n)$. The *clausal language* $\mathcal{D}_{\mathcal{A}}$ given by a first-order alphabet $\mathcal{A}$ consists of the set of all clauses that can be constructed from the symbols in $\mathcal{A}$.

An example clause D_1 is, e.g., $\forall X(\neg \text{block}(X) \vee \text{black}(X))$, stating that all blocks are black.

Definition 2. A *production* is a finite conjunction of literals closed under existential quantification $\exists(l_1 \wedge \dots \wedge l_n)$. The *production language* $\mathcal{C}_{\mathcal{A}}$ given by a first-order alphabet $\mathcal{A}$ consists of the set of all productions that can be constructed from the symbols in $\mathcal{A}$.

An example production C_1 is, e.g., $\exists X(\text{block}(X) \wedge \neg \text{black}(X))$, stating that there is a block which is not black. The results and discussion in this paper will be independent of the choice of first-order alphabet $\mathcal{A}$, and we will therefore for simplicity in the following write $\mathcal{D}$ and $\mathcal{C}$ instead of $\mathcal{D}_{\mathcal{A}}$ and $\mathcal{C}_{\mathcal{A}}$.

For the purposes of this paper, a *formula* F will be a production or a clause. We will denote the set of literals in a formula F by $\text{set}(F)$. E.g., the example clause D_1 above corresponds to the set $\text{set}(D_1) = \{\neg \text{block}(X), \text{black}(X)\}$.

Also, we introduce two one-to-one mappings, which will be used further on.

Definition 3. The mapping $\mathcal{F}_{\mathcal{C}} : \mathcal{C} \mapsto \mathcal{D}$ maps every production in $\mathcal{C}$ to its negation in $\mathcal{D}$. The mapping $\mathcal{F}_{\mathcal{D}} : \mathcal{D} \mapsto \mathcal{C}$ maps every clause in $\mathcal{D}$ to its negation in $\mathcal{C}$.

This means that

$$\mathcal{F}_C(\exists(l_1 \wedge \dots \wedge l_n)) = \forall(\neg l_1 \vee \dots \vee \neg l_n),$$

and

$$\mathcal{F}_D(\forall(l_1 \vee \dots \vee l_n)) = \exists(\neg l_1 \wedge \dots \wedge \neg l_n).$$

The reader may notice that $\mathcal{F}_C^{-1} = \mathcal{F}_D$ and $\mathcal{F}_D^{-1} = \mathcal{F}_C$, since $\neg\neg l = l$ for any literal l . Furthermore, $\mathcal{F}_C(c) = \neg c$ and $\mathcal{F}_D(d) = \neg d$.

A substitution $\theta = \{V_1/t_1, \dots, V_n/t_n\}$ is an assignment of terms to variables. Applying a substitution θ to a term, literal, or formula F yields the instantiated term, literal, or formula $F\theta$, where, for every $1 \leq i \leq n$, all occurrences of the variables V_i in F are simultaneously replaced by the corresponding terms t_i .

3 θ -subsumption

Let us now introduce Plotkin's notion of θ -subsumption.

Definition 4. A clause D_1 θ -subsumes a clause D_2 if and only if there exists a substitution θ such that $\text{set}(D_1)\theta \subseteq \text{set}(D_2)$. We write $D_1 \preceq_D D_2$ to denote that D_1 θ -subsumes D_2 .

For example, $\text{father}(X, Y) \vee \neg \text{parent}(X, Y) \vee \neg \text{male}(X)$ θ -subsumes $\text{father}(\text{jef}, \text{paul}) \vee \neg \text{parent}(\text{jef}, \text{paul}) \vee \neg \text{parent}(\text{jef}, \text{ann}) \vee \neg \text{male}(\text{jef}) \vee \neg \text{female}(\text{ann})$ with $\theta = \{X / \text{jef}, Y / \text{paul}\}$.

The notion of least general generalisation of a set of clauses is then defined as follows:

Definition 5. A clause G is a *generalisation* of a set of clauses $S = \{D_1, \dots, D_n\}$ if and only if, for every $1 \leq i \leq n$, $G \preceq_D D_i$. A generalisation G of a set of clauses S is a *least general generalisation* (lgg_D) of S if and only if, for every generalisation G' of S , $G' \preceq_D G$.

The operation of computing an lgg_D underlies many ILP systems. Plotkin [18] has investigated the structure imposed by θ -subsumption. He has proven the following key properties (cf. [13]):

Implication. If D_1 θ -subsumes D_2 , then $D_1 \models D_2$. The opposite does not hold for self-recursive clauses: let $D_1 = p(f(X)) \vee \neg p(X)$ and $D_2 = p(f(f(Y))) \vee \neg p(Y)$; then $D_1 \models D_2$ but D_1 does not θ -subsume D_2 . Therefore deduction using θ -subsumption is not equivalent to implication among clauses. This also motivated research on inverting implication by Idestam-Almqvist [9], Muggleton [14], Lapointe and Matwin [11].

This property also shows that θ -subsumption is a meaningful structure on $\mathcal{D}$ for ILP systems. Indeed, typical ILP systems employ a set of positive examples P (represented by ground clauses) and aim at deriving a hypothesis H (represented by a *set* of clauses) that implies the facts, i.e., $H \models P$. Because of the implication property, this aim will be fulfilled when all clauses in P are θ -subsumed by clauses in H .

Equivalence. There exist different clauses that are equivalent under θ -subsumption, e.g., $\text{parent}(X, Y) \vee \neg \text{mother}(X, Y) \vee \neg \text{mother}(X, Z)$ θ -subsumes $\text{parent}(X, Y) \vee \neg \text{mother}(X, Y)$ and vice versa. Because two clauses equivalent under θ -subsumption are also logically equivalent (i.e., by implication), learning systems should generate at most one clause of each equivalence class. For an extended discussion of equivalence, see Maher [12].

Reduction. To get around this problem, Plotkin defined equivalence classes of clauses, and showed that there is a unique representative (up to variable renamings) of each equivalence class, which he named the *reduced* clause. A clause D is reduced if and only if there exists no literal l such that $(D - \{l\})$ and D are equivalent under θ -subsumption. Learning systems can get around the problem of equivalent clauses when working with reduced clauses only. An algorithm to reduce clauses directly follows from the definition of a reduced clause: repeatedly delete a literal from a clause and check whether the resulting clause is θ -subsumed by the original clause; if it is, apply the corresponding substitution, omit duplicate literals, and repeat the procedure; if no literal can be deleted in this way, conclude that the clause is reduced.

Lattice. The set of reduced clauses forms a lattice, i.e., any two reduced clauses have a unique lub_D (least upper bound, which is the lgg_D) and a unique glb_D (greatest lower bound). The least general generalisation operator is the basic operation of many specific to general ILP systems, e.g., Golem [16], Clint [2], ITVS [22] applied on ILP.

Plotkin has also given a procedure to compute the lgg_D of two clauses: the lgg_D of the terms $f(s_1, \dots, s_n)$ and $f(t_1, \dots, t_n)$ is $f(\text{lgg}_D(s_1, t_1), \dots, \text{lgg}_D(s_n, t_n))$. The lgg_D of the terms $f(s_1, \dots, s_n)$ and $g(t_1, \dots, t_m)$, with $f \neq g$, is the variable v , where v represents this pair of terms throughout. The lgg_D of two atoms $p(s_1, \dots, s_n)$ and $p(t_1, \dots, t_n)$ is $p(\text{lgg}_D(s_1, t_1), \dots, \text{lgg}_D(s_n, t_n))$, the lgg_D being undefined when the sign or the predicate symbols are unequal. Finally, the lgg_D of two clauses D_1 and D_2 is then

$$\bigvee_{l_1 \in \text{set}(D_1)} \bigvee_{l_2 \in \text{set}(D_2)} \text{lgg}_D(l_1, l_2).$$

For example, the lgg_D of $\text{father}(\text{tom}, \text{ann}) \vee \neg \text{parent}(\text{tom}, \text{ann}) \vee \neg \text{male}(\text{tom}) \vee \neg \text{female}(\text{ann})$ and $\text{father}(\text{jef}, \text{paul}) \vee \neg \text{parent}(\text{jef}, \text{paul}) \vee \neg \text{male}(\text{jef}) \vee \neg \text{male}(\text{paul})$ is $\text{father}(X, Y) \vee \neg \text{parent}(X, Y) \vee \neg \text{male}(X) \vee \neg \text{male}(Z)$. The equivalent reduced clause is $\text{father}(X, Y) \vee \neg \text{parent}(X, Y) \vee \neg \text{male}(X)$.

Infinite Ascending Chains. There exist infinite strictly ascending chains in the above lattice, e.g.,

$$\begin{aligned} &h(X_1) \vee \neg p(X_1, X_2) \\ &h(X_1) \vee \neg p(X_1, X_2) \vee \neg p(X_2, X_3) \\ &h(X_1) \vee \neg p(X_1, X_2) \vee \neg p(X_2, X_3) \vee \neg p(X_3, X_4) \\ &\dots \end{aligned}$$

Infinite Descending Chains. There also exist infinite strictly descending chains in the lattice, e.g.,

$$\begin{aligned}
& h(X_1) \vee \neg p(X_1, X_2) \vee \neg p(X_2, X_1) \\
& h(X_1) \vee \neg p(X_1, X_2) \vee \neg p(X_2, X_3) \vee \neg p(X_3, X_4) \vee \neg p(X_4, X_1) \\
& h(X_1) \vee \neg p(X_1, X_2) \vee \neg p(X_2, X_3) \vee \neg p(X_3, X_4) \vee \neg p(X_4, X_5) \vee \neg \\
& p(X_5, X_6) \vee \neg p(X_6, X_7) \vee \neg p(X_7, X_8) \vee \neg p(X_8, X_1) \\
& \dots
\end{aligned}$$

All clauses in both these infinite series θ -subsume the clause $h(X) \vee \neg p(X, X)$ and are θ -subsumed by the clause $h(X) \vee \neg p(X, Y)$.

4 Structural matching

In this section, we map Plotkin's notion of θ -subsumption onto productions. We will show that the resulting structure is order-isomorphic to θ -subsumption on clauses. As a result the algorithms and results by Plotkin can be directly mapped to productions.

Definition 6. A production C_1 θ -subsumes a production C_2 if and only if there exists a substitution θ such that $set(C_1)\theta \subseteq set(C_2)$. We will write $C_1 \preceq_C C_2$ to denote that C_1 θ -subsumes C_2 .

For example, $\exists X, Y (circle(X) \wedge red(X) \wedge circle(Y) \wedge large(Y))$ θ -subsumes $circle(a) \wedge red(a) \wedge large(a)$ with substitution $\theta = \{X/a, Y/a\}$.

Vere employed a different notion of generalization. More specifically, according to Vere, a production C_2 is more specific than a production C_1 if there is an inverse substitution θ^{-1} such that $set(C_2)\theta^{-1} \subseteq set(C_1)$. An inverse substitution maps terms *at specified places* onto more general terms (or variables). Within the inductive logic programming literature, it is well-known that inverse substitutions are much harder to use and reason about than classical deductive substitutions [17].

A straightforward property of $\preceq_C$, which could be used as an alternative definition, is:

Lemma 7. Let C_1 and C_2 be two productions. Then, $C_1 \preceq_C C_2$ if and only if $\mathcal{F}_C(C_1) \preceq_D \mathcal{F}_C(C_2)$.

Let us now show that θ -subsumption imposes a meaningful structure on $\mathcal{C}$. This follows from :

Lemma 8. Let C_1 and C_2 be two productions. If $C_1 \preceq_C C_2$, then $C_2 \models C_1$.

Proof. This will be proven as a special case of Lemma 14. □

In structural matching examples are ground (i.e., variable free) productions. Ground productions closely correspond to logical interpretations (which can also be represented by a conjunction of true and false ground facts). When using examples that are interpretations, Lemma 8 shows that θ -subsumption is doing

precisely what is expected: all interpretations that are a model for a production C_2 (i.e., that are *covered* by C_2) are also a model for every production C_1 that θ -subsumes C_2 . Thus, θ -subsumption provides a genuine generality relation on $\mathcal{C}$.

It is straightforward to see that the structure imposed by θ -subsumption on $\mathcal{C}$ is exactly the same as that imposed on $\mathcal{D}$. This is because both productions and clauses are mapped to sets of literals when reasoning about θ -subsumption.

Formally, we have:

Theorem 9. *The ordered set $(\mathcal{C}, \preceq_{\mathcal{C}})$ is order-isomorphic to the ordered set $(\mathcal{D}, \preceq_{\mathcal{D}})$.*

Proof: We have to show that there is a one-to-one mapping from $\mathcal{C}$ to $\mathcal{D}$, which is monotone. The mapping $\mathcal{F}_{\mathcal{C}}$ clearly is one-to-one. It is monotone because for any productions C_1 and C_2 , $C_1 \preceq_{\mathcal{C}} C_2$ if and only if $\mathcal{F}_{\mathcal{C}}(C_1) \preceq_{\mathcal{D}} \mathcal{F}_{\mathcal{C}}(C_2)$ (Lemma 7). $\square$

As a consequence, all results proven by Plotkin and all algorithms on clauses apply to productions as well. For learning algorithms the most important results are concerned with reduced descriptions and least general generalisations.

We do not provide detailed descriptions of the resulting algorithms as they are essentially the same as Plotkin's, due to the uniform set representation. However, we can now easily define $lgg_{\mathcal{C}}$ in terms of $lgg_{\mathcal{D}}$ and the mappings $\mathcal{F}_{\mathcal{C}}$ and $\mathcal{F}_{\mathcal{D}}$ because of Theorem 9.

Corollary 10. *Let C_1 and C_2 be two productions. Then,*

$$lgg_{\mathcal{C}}(C_1, C_2) = \mathcal{F}_{\mathcal{D}}(lgg_{\mathcal{D}}(\mathcal{F}_{\mathcal{C}}(C_1), \mathcal{F}_{\mathcal{C}}(C_2))).$$

Let us also illustrate that the resulting structure on the search space solves some of the remaining problems with structural matching.

Example 1. Consider the following descriptions of scenes (from [8]).

$$C_1 = \text{circle}(a) \wedge \text{red}(a) \wedge \text{large}(a)$$

$$C_2 = \text{circle}(b) \wedge \text{circle}(c) \wedge \text{red}(b) \wedge \text{green}(c) \wedge \text{small}(b) \wedge \text{large}(c)$$

$$G_1 = \exists X(\text{circle}(X) \wedge \text{red}(X))$$

$$G_2 = \exists X(\text{circle}(X) \wedge \text{large}(X))$$

$$G = \exists X, Y(\text{circle}(X) \wedge \text{large}(X) \wedge \text{circle}(Y) \wedge \text{red}(Y))$$

Hayes-Roth and McDermott's algorithm would return both G_1 and G_2 as maximally specific generalisations of C_1 and C_2 . As a consequence, their system has to explore two different paths through the search space. Similar problems exist for the framework of Steven Vere [23]. In contrast, according to our notion of generality G is the unique least general generalisation of C_1 and C_2 . Furthermore, G is reduced.

Another problem solved adopting the ordered set $(\mathcal{C}, \preceq_{\mathcal{C}})$ is that it enables to handle function symbols.

5 Relative θ -subsumption

Plotkin [20] extended the notion of θ -subsumption to θ -subsumption relative to a theory (i.e., a finite set of clauses).

Definition 11. A clause D_1 θ -subsumes a clause D_2 relative to a theory T , denoted $D_1 \preceq_{\mathcal{D}, T} D_2$, if and only if there exists a C-derivation of a clause R from D_1 and T such that $R \preceq_{\mathcal{D}} D_2$.

A C-derivation of R from D_1 and T is a resolution derivation of R from D_1 and T in which the clause D_1 is used exactly once. Our definition slightly differs from Plotkin's definition; in Plotkin's definition a resolution derivation is a C-derivation if and only if the clause D_1 is used *at most* once. Hence, our notion of relative θ -subsumption differs from Plotkin's w.r.t. clauses that logically follow from the theory only. With our definition a clause is a $lgg_{\mathcal{D}}$ of a set of clauses if and only if it is a $rlgg_{\mathcal{D}}$ of this set of clauses w.r.t. the empty theory. This is not the case with Plotkin's definition. However, the difference between the two definitions is not important since clauses that follow from the theory only are usually not considered in inductive learning.

Note that if a clause D_1 θ -subsumes a clause D_2 , then it also θ -subsumes D_2 w.r.t. a theory T , by choosing $R = D_1$.

Definition 12. A clause G is a *relative generalisation* of a set of clauses $S = \{D_1, \dots, D_n\}$ w.r.t. a theory T if and only if, for every $1 \leq i \leq n$, $G \preceq_{\mathcal{D}, T} D_i$. A relative generalisation G of a set of clauses S w.r.t. a theory T is a *relative least general generalisation* ($rlgg_{\mathcal{D}, T}$) of S if and only if, for every relative generalisation G' of S , $G' \preceq_{\mathcal{D}, T} G$.

The idea of relative θ -subsumption is that one can take into account a background theory when learning. Relative θ -subsumption has similar properties as θ -subsumption.

Implication. If a clause D_1 θ -subsumes a clause D_2 relative to a theory T , then $T \cup \{D_1\} \models D_2$.

This property shows that relative θ -subsumption imposes a meaningful structure on $\mathcal{D}$ for ILP systems in the presence of a background theory. Indeed, typical ILP systems employ a set P of positive examples (represented by ground definite clauses) and a theory T , and aim at deriving a hypothesis H that implies the clauses, i.e., $T \cup H \models P$. Because of the implication property, the aim of ILP will be fulfilled when all clauses in P are θ -subsumed by clauses in H relative to T .

Equivalence, Reduction and Lattice. Because relative θ -subsumption is directly derived from θ -subsumption, the structure $(\mathcal{D}, \preceq_{\mathcal{D}, T})$ is very similar to $(\mathcal{D}, \preceq_{\mathcal{D}})$. Indeed, Plotkin [20] also defines notions of equivalence, reduction, and relative least general generalisation w.r.t. a theory T . Unfortunately, the theoretical results for generalisation relative to a theory are not as good as for generalisation without respect to a theory. In particular, in general there

does not exist an $rlgg_{\mathcal{D},T}$ of a set of clauses w.r.t. a theory. This is due to the fact that a clause by definition is a finite disjunction of literals, while a relative least general generalisation may include an infinite number of literals. However, whenever there exists an $rlgg_{\mathcal{D},T}$ it is unique up to equivalence. The $rlgg_{\mathcal{D},T}$ of a set of definite clauses $\{D_1, \dots, D_n\}$ w.r.t. a theory T can be computed in the following way [15, 10] For each clause D_i ($1 \leq i \leq n$) collect all ground unit clauses that are derivable from T and the complement $\overline{D_i}$ of D_i , and construct a clause D'_i that is the complement of this set of unit clauses. Then the $rlgg_{\mathcal{D},T}$ of $\{D_1, \dots, D_n\}$ w.r.t. T can be found by computing the $lgg_{\mathcal{D}}$ of $\{D'_1, \dots, D'_n\}$.

6 Relative Structural Matching

Kodratoff and Ganascia [6], and Vrain [26] have all given definitions of structurally matching two productions in the presence of background knowledge, thereby building on Vere's original framework. However, similarly as Vere's original framework, these techniques suffer from a number of problems, which include the fact that there may be multiple relative most specific generalisations, and difficulties in handling functors. The question thus arises as to whether it is possible to map Plotkin's structure of relative θ -subsumptions onto the set of productions.

One way to define relative subsumption for productions would be to map Plotkin's definition directly onto productions. As this would involve modifying the proof-procedure, this would be rather complicated. Therefore, we will rather upgrade the alternative definition of θ -subsumption for production given in Lemma 7. This results in the following definition :

Definition 13. A production C_1 θ -subsumes a production C_2 relative to a theory T , denoted $C_1 \preceq_{C,T} C_2$, if and only if $\mathcal{F}_C(C_1) \preceq_{\mathcal{D},T} \mathcal{F}_C(C_2)$.

First note that in this definition T is also a clausal theory. Second, if T is empty, this definition again reduces to θ -subsumption for productions.

Let us again show that this structure is meaningful for concept-learning when using interpretations or ground productions as examples and a clausal background theory:

Lemma 14. Let C_1 and C_2 be productions and T a theory. If $C_1 \preceq_{C,T} C_2$ then $T \wedge C_1 \models C_2$.

Proof. $C_1 \preceq_{C,T} C_2$ iff

$\mathcal{F}_C(C_1) \preceq_{\mathcal{D},T} \mathcal{F}_C(C_2)$ only if (because of the relation of $\preceq_{\mathcal{D},T}$ to implication)

$T \wedge \mathcal{F}_C(C_1) \models \mathcal{F}_C(C_2)$ only if

$T \wedge \neg \mathcal{F}_C(C_2) \models \neg \mathcal{F}_C(C_1)$ only if

$T \wedge C_2 \models C_1$ (because $\neg \mathcal{F}_C(C) = C$ for all productions C). □

This lemma reduces to Lemma 8 when T is empty.

Again it is straightforward to show that

Theorem 15. $(C, \preceq_{C,T})$ and $(D, \preceq_{D,T})$ are order-isomorphic for all theories T .

And again this allows us to compute the $rlgg_C$, whose definition is analogous, using $rlgg_D$.

Corollary 16. Let C_1 and C_2 be two productions and T a theory. Then

$$rlgg_{C,T}(C_1, C_2) = \mathcal{F}_D(rlgg_{D,T}(\mathcal{F}_C(C_1), \mathcal{F}_C(C_2))).$$

This also means that the $rlgg_{C,T}$ inherits all the properties of the dual $rlgg_{D,T}$. Using this operation again solves many of the problems with the operations proposed by Vere [23, 24], Kodratoff and Ganascia [6] and Vrain [26], in particular, uniqueness, use of functors and also handling inequalities. To handle inequalities, it suffices to add the clauses $different(a, b)$ and $different(a, c)$, etc. for all different combinations of known constants to the theory.

To illustrate the use of background knowledge we take the following example of Kodratoff and Ganascia [6]. The example also illustrates that the use of functors is possible.

Example 2. Consider

$C_1 = red(a) \wedge red(b) \wedge shape(b, square)$, and

$C_2 = red(c) \wedge shape(d, square)$.

The background knowledge T contains the “color hierarchy”:

$color(X, red) \vee \neg red(X)$.

$color(X, green) \vee \neg green(X)$.

$color(X, yellow) \vee \neg yellow(X)$.

Also known is the fact that each object has a color. Kodratoff and Ganascia express this as: $\forall X \exists Y : color(X, Y)$. In clausal form we can make use of a Skolem functor $col/1$, and express this as the clause $color(X, col(X))$ in T .

Finally T contains the above mentioned clauses that express that different constants represent different objects.

The production $red(X) \wedge red(Y) \wedge square(Z) \wedge different(X, Z)$ is an lgg_C of the two productions C_1 and C_2 . The “reduced” form of this production is: $G_1 = red(X) \wedge square(Z) \wedge different(X, Z)$.

The production $red(X) \wedge red(Y) \wedge square(Z) \wedge different(X, Z) \wedge color(U, V) \wedge color(W, Z)$ is an $rlgg_{C,T}$ of C_1 and C_2 with respect to the given background knowledge. The “reduced” form of this production w.r.t. θ -subsumption is: $G_2 = red(X) \wedge square(Z) \wedge different(X, Z) \wedge color(W, Z)$.

Kodratoff and Ganascia obtain G_1 and G_2 by structurally matching C_1 and C_2 in two different ways. Whereas they call these two possible generalisations between which one cannot choose “because they have an equal number of variables”, the relationship of structural matching with Plotkin’s work reveals that G_1 *subsumes* G_2 w.r.t. θ -subsumption. This means that G_2 is a more specific generalisation, which could therefore be preferred over G_1 .

Note that, unlike in structural matching, this has the advantage that no further heuristics (e.g., the counting of variables) is needed, since the relative least general generalisation is unique modulo reduction. Furthermore, whereas structural matching explicitly records all variable bindings and controls the introduction of new variables heuristically, Plotkin's algorithm deals with all these problems in a uniform way.

7 Conclusion

We have shown the order-isomorphisms between Plotkin's framework and the framework of structural matching. This isomorphism implies that the results obtained in one framework can be transferred to the other. This in turn leads to an improved structure on the space of productions for use in concept-learning.

Finally, we hope that this work will stimulate further cross-fertilization between structural matching and ILP. Some challenging remaining research issues include: can we adapt rlgg based ILP systems, such as Muggleton and Feng's Golem for use in structural matching?

Acknowledgements

Luc De Raedt is supported by the Belgian National Fund for Scientific Research. Peter Idestam-Almquist is supported by the Swedish Research Council for Engineering Sciences (TFR). Luc De Raedt and Peter Idestam-Almquist were also supported by the ESPRIT Basic Research Projects No. 6020 and 20.237 on Inductive Logic Programming I and II. The authors would like to thank Maarten Van Someren for suggestions for improving this paper.

References

1. G. Bisson. Learning in FOL with a Similarity Measure. In *Proceedings of AAAI*, 1992.
2. L. De Raedt. Interactive Theory Revision: an Inductive Logic Programming approach, Academic Press, 1992.
3. L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1058–1063. Morgan Kaufmann, 1993.
4. L. De Raedt and S. Džeroski. First order jk -clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.
5. L. De Raedt. Induction in Logic. In *Proceedings of the 3rd Multistrategy Learning Workshop*, 1996.
6. J.G. Ganascia and Y. Kodratoff. Improving the generalization step in learning. In R.S Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: an artificial intelligence approach*, volume 2, pages 215–241. Morgan Kaufmann, 1986.
7. R. Gemello, F. Mana, and L. Saitta. Rigel: An inductive learning system. *Machine Learning*, 6:7–35, 1991.

8. F. Hayes-Roth and J. McDermott. An interference matching technique for inducing abstractions. *Communications of the ACM*, 21:401-410, 1978.
9. P. Idestam-Almquist. Generalisation under implication using or-introduction. In *Proceedings of the 6th European Conference on Machine Learning*, volume 667, pages 56-64. Lecture Notes in Artificial Intelligence, 1993.
10. P. Idestam-Almquist. Generalization of clauses relative to a theory *Machine Learning*, in press, 1997.
11. S. Lapointe and S. Matwin. Sub-unification: a tool for efficient induction of recursive programs. In *Proceedings of the 9th International Workshop on Machine Learning*. Morgan Kaufmann, 1992.
12. M.J. Maher. Equivalences of logic programs. In *Proceedings of Third International Conference on Logic Programming*, Berlin, 1986. Springer.
13. S. Muggleton and L. De Raedt. Inductive logic programming: theory and methods. *Journal of Logic Programming*, Vol. 19-20, 1994.
14. S. Muggleton. Inverting implication. *Artificial Intelligence*, 1994. To appear.
15. S. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8:295-317, 1991.
16. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proceedings of the 1st conference on algorithmic learning theory*, pages 368-381. Ohmsma, Tokyo, Japan, 1990.
17. S. Muggleton and W. Buntine. Machine invention of first order predicates by inverting resolution. In *Proceedings of the 5th International Workshop on Machine Learning*, Morgan Kaufmann, 1988.
18. G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153-163. Edinburgh University Press, 1970.
19. G. Plotkin. *Automatic Methods of Inductive Inference*. PhD thesis, Edinburgh University, 1971.
20. G. Plotkin. A further note on inductive generalization. In *Machine Intelligence*, volume 6, pages 101-124. Edinburgh University Press, 1971.
21. J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239-266, 1990.
22. G. Sablon, L. De Raedt, and M. Bruynooghe. Iterative Versionspaces. *Artificial Intelligence*, 69:393-409, 1994.
23. S.A. Vere. Induction of concepts in the predicate calculus. In *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, pages 282-287. Morgan Kaufmann, 1975.
24. S.A. Vere. Induction of relational productions in the presence of background information. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1977.
25. S.A. Vere. Multilevel counterfactuals for generalizations of relational concepts and productions. *Artificial Intelligence*, 14:139-164, 1980.
26. C. Vrain. Ogust: A system that learns using domain properties expressed as theorems. In Y. Kodratoff and R.S. Michalski, editors, *Machine Learning: an artificial intelligence approach*, volume 3, pages 360-381. Morgan Kaufmann, 1990.