Generalization with Taxonomic Information

Alan M. Frisch and **C. David Page Jr.**

Dept. of Computer Science and Beckman Institute
University of Illinois
405 North Mathews Ave.
Urbana, IL 61801

Abstract

This paper studies sorted generalization—the generalization, with respect to an arbitrary taxonomic theory, of atomic formulas containing sorted variables. It develops an algorithm for the task, discusses the algorithm and task complexity, and presents semantic properties of sorted generalization. Based on its semantic properties, we show how sorted generalization is applicable to such problems as abduction, induction, knowledge base vivification, and analogical reasoning. Significant distinctions between this work and related work with taxonomic information arise from the generality of the taxonomic theories we allow, which may be any first-order taxonomic theories, and the semantic completeness properties of sorted generalization.

Introduction

Unification plays an important role in many automated deduction systems. It comes in many forms including term unification, string unification, unification with built-in equality, sorted unification, and feature structure unification, as well as variations of these. The common feature of all these forms of unification is that they compute maximal lower bounds in some partially-ordered set of syntactic objects. Turning unification on its head yields an operation called "generalization," or "anti-unification," that computes minimal upper bounds. As with unification, one can imagine many forms of generalization. We conjecture that generalization will play a role in non-deductive reasoning (e.g., abduction, induction and analogy) as important as that played by unification in deductive reasoning. (Unification has some non-deductive applications, and, as we shall see, generalization can have deductive applications.) This paper studies sorted generalization, the dual of sorted unification. Whereas ordinary generalization, which has been studied by Plotkin [1970; 1971], Reynolds [1970], Lassez, Maher and Marriott [1988], and Lassez and Marriott [1987], is a syntactic operation that operates solely on the basis of expression structure, sorted generalization takes into account a body of taxonomic information. Giving special treatment to taxonomic information has been a fruitful approach to the construction of AI reasoning systems. For example, sorted logics distinguish taxonomic information from other information, and their deductive systems can exploit this distinction by handling the taxonomic information with special-purpose methods such as sorted unification. This paper formalizes the sorted generalization problem, formulates an algorithm for its solution, discusses the complexity of the problem and the algorithm, and shows how sorted generalization can be applied to problems in abduction, induction, knowledge base vivification and analogical reasoning.

Before formalizing the task, let's intuitively examine an example of sorted generalization and compare it with unsorted generalization. The examples in this paper use the following taxonomic information:

$$\begin{aligned}\Sigma_1 = \{&\forall x\ \text{UNIV}(x),\ \text{GOOD-COOK}(\mathit{mother}(\mathit{clyde})),\\ &\text{GOOD-COOK}(\mathit{mother}(\mathit{jumbo})),\\ &\text{ELEPHANT}(\mathit{clyde}),\ \text{CIRCUS-ANIMAL}(\mathit{clyde}),\\ &\text{ELEPHANT}(\mathit{jumbo}),\ \text{CIRCUS-ANIMAL}(\mathit{jumbo}),\\ &\forall x\ \text{ELEPHANT}(x) \rightarrow \text{ELEPHANT}(\mathit{mother}(x))\}.\end{aligned}$$

Example 1
Let $E = \{\mathit{eats}(\mathit{clyde},\mathit{peanuts}),\ \mathit{eats}(\mathit{jumbo},\mathit{peanuts})\}$.
E has two maximally-specific sorted generalizations with respect to Σ_1:
$\mathit{eats}(x{:}\text{ELEPHANT},\mathit{peanuts})$ and
$\mathit{eats}(y{:}\text{CIRCUS-ANIMAL},\mathit{peanuts})$.
E has a most specific unsorted generalization:
$\mathit{eats}(x,\mathit{peanuts})$.

As Example 1 illustrates, there are cases in which there is no most specific sorted generalization. Neither $\mathit{eats}(x{:}\text{ELEPHANT},\mathit{peanuts})$ nor $\mathit{eats}(y{:}\text{CIRCUS-ANIMAL},\mathit{peanuts})$ is more specific than the other. Notice also that the sorted generalizations are more specific than the unsorted generalization.[1]

*This research was partially funded by a grant from McDonnell Douglas Corporation. The writing of the paper was completed while the first author was on leave at IBM T. J. Watson Research Center.

[1] Section 2 precisely defines "specific" and "general."

Sorted generalizations are never less specific than unsorted generalizations.

Formalizing the Problem

We formalize the sorted generalization problem using Sorted First Order Predicate Calculus (SFOPC), a language that extends First Order Predicate Calculus (FOPC) with special notation for encoding taxonomic information [Frisch, 1989]. SFOPC is no more expressive than FOPC; each sentence of SFOPC is logically equivalent to one of FOPC. SFOPC is written with a lexicon that contains the usual function and predicate symbols and, in addition, contains a countable set of *sort symbols.* Typographically, sort symbols are written entirely in upper-case. Semantically, a sort symbol denotes a nonempty subset of the domain, called a sort. SFOPC contains two kinds of expressions, "A-expressions" and "S-expressions." We refer to A-expressions that are terms as "A-terms" and A-expressions that are formulas as "A-formulas," and we use "S-term" and "S-formula" analogously for S-expressions. We say that an expression is atomic if it is a term or an atomic formula. Roughly, the goal of sorted generalization is to find the maximally-specific generalizations of a set of atomic A-expressions with respect to a theory consisting of S-expressions.

A-expressions are similar to ordinary FOPC expressions except that they may contain *sorted variables*, variables that are restricted to range over specified subsets of the domain. A sorted variable is a pair, $x{:}\tau$, where x is a variable name and τ is a sort symbol. For clarity, variables are sometimes written in angle brackets, such as $\langle x{:}\tau\rangle$. τ and ω are used as metalinguistic symbols that always stand for sort symbols. The meanings of A-expressions are similar to those of FOPC expressions, except for the following rules for quantification over sorted variables. In these semantic rules $[\![\psi]\!]^{M,e}$ is the semantic value assigned to an expression or symbol ψ by a model M and an assignment to variables e, and $e[d/x]$ is the assignment to variables that is identical to e with the possible exception that x is assigned d.

$$[\![\forall x{:}\tau\ \phi]\!]^{M,e} = \text{True if, and only if, for every } d \in [\![\tau]\!]^{M,e}, [\![\phi]\!]^{M,e[d/x]} = \text{True}$$

$$[\![\exists x{:}\tau\ \phi]\!]^{M,e} = \text{True if, and only if, for some } d \in [\![\tau]\!]^{M,e}, [\![\phi]\!]^{M,e[d/x]} = \text{True}$$

The role of S-expressions is to express relationships among the sorts and the sortal behavior of the functions. S-expressions are constructed like ordinary expressions of FOPC, except they contain no ordinary predicate symbols; in their place are sort symbols acting as monadic predicate symbols. Hence, every atomic S-formula is of the form $\tau(t)$, where τ is a sort symbol and t is an ordinary term. S-formulas are assigned truth values in the usual Tarskian manner. A finite satisfiable set of S-sentences is called a *sort theory* and is frequently denoted by Σ. Σ_1 is a simple example of a sort theory. Sort theories may also contain sentences of other forms, such as $\text{DOG}(\textit{fred}) \vee \text{ELEPHANT}(\textit{fred})$ and $\forall x\ \text{DOG}(x) \wedge \text{YOUTH}(x) \rightarrow \text{PUPPY}(x)$.

We are often interested in whether one sort is a subset of another sort according to a given sort theory Σ. τ' is a subsort of τ according to Σ if, and only if, $\Sigma \models \forall x\ \tau'(x) \rightarrow \tau(x)$; we write $\tau' \preceq_\Sigma \tau$. To ensure that any two terms have some generalization, as in the unsorted case, we augment every sort theory with a sort symbol, UNIV, representing the sort consisting of the entire domain (the universal sort). We therefore assume that each sort theory contains the S-sentence $\forall x\ \text{UNIV}(x)$. It follows from this assumption that $\tau \preceq_\Sigma \text{UNIV}$ for every sort symbol τ and every sort theory Σ. Note that a variable of the universal sort behaves as an unsorted variable.

Unification and generalization for FOPC formulas are defined in terms of an instantiation ordering, that of substitutions. Sorted generalization, as well as sorted unification, is defined in terms of an analogous instantiation ordering for SFOPC, that of well-sorted substitutions. Intuitively, a well-sorted substitution is a substitution that respects the sorts of the variables. More precisely, a substitution θ is well-sorted relative to a sort theory Σ if, and only if, for every variable $x{:}\tau$, $\langle x{:}\tau\rangle\theta$ is a term t such that $\Sigma \models \forall\tau(t)$.[2] Thus, for example, $\theta_1 = \{\textit{clyde}/x{:}\text{ELEPHANT}\}$ is a well-sorted substitution relative to Σ_1, but $\theta_2 = \{\textit{fido}/x{:}\text{ELEPHANT}\}$ is not. An A-expression e is said to be Σ-more general than e' if $e' = e\theta$, for some substitution θ that is well sorted relative to Σ. We also say that e Σ-subsumes e', and we write $e \geq_\Sigma e'$. An A-expression that Σ-subsumes each member of a set of A-expressions is a Σ-generalization of the set.

The most general common instance, e, of a set, E, of atomic expressions may be found with unification and characterizes the common instances of E: an expression is a common instance of E if, and only if, it is an instance of e. Likewise, the most specific generalization, g, of E characterizes the expressions that are more general than every expression in E: an expression is more general than every member of E if, and only if, it is more general than g. We would like to use a Σ-generalization to similarly characterize the A-expressions that are Σ-more general than any given set of atomic A-expressions. But Example 1 shows that there may be no most specific Σ-generalization. An analogous characterization therefore requires a set of maximally specific Σ-generalizations, none Σ-more general than another. Specifically, the goal of sorted generalization is to find a *complete set of incomparable Σ-generalizations* (or CIG_Σ) for a given set of atomic

[2]For any expression ψ, $\forall\psi$ denotes the universal closure of ψ, the result of universally quantifying all free variables of ψ. Similarly, $\exists\psi$ denotes the existential closure of ψ.

A-expressions. A set G is a CIG_Σ of a set E of atomic A-expressions if, and only if,

- each member of G is Σ-more general than every member of E [Correctness],
- any atomic A-expression that is Σ-more general than every member of E is Σ-more general than some member of G [Completeness], and
- no member of G is Σ-more general than any other [Incomparability].

Some sets of atomic A-expressions have no Σ-generalization; the CIG_Σ of any such set is the empty set. For example, there is no A-expression that is Σ-more general than two atomic A-formulas with different predicate symbols. More completely, a set of atomic A-expressions has an empty CIG_Σ if, and only if, it contains both A-terms and atomic A-formulas or contains atomic A-formulas with different predicate symbols. Might other sets have more than one CIG_Σ? Can a CIG_Σ be infinite?

Lemma 1 *All CIG_Σs of a set of atomic A-expressions are variants. That is, any CIG_Σ can be obtained from another by uniformly renaming variables.*

Lemma 2 *There are only a finite number of A-expressions Σ-more general than a given A-expression, up to renaming.*

Corollary 3 *Every CIG_Σ is finite.*

The following two examples of sorted generalization illustrate additional distinctions between sorted and unsorted generalization, which a sorted generalization algorithm must accommodate. In Example 2 "*mother*" is a function symbol, so E is a set of terms.

Example 2
Let $E = \{mother(clyde),\ mother(jumbo)\}$.
Then a CIG_{Σ_1} of E is

$$\{y{:}\textsc{good-cook},\ mother(x{:}\textsc{elephant}),\ mother(z{:}\textsc{circus-animal})\},$$

and a most specific unsorted generalization of E is $mother(x)$.

In Example 2, y:GOOD-COOK Σ_1-subsumes neither $mother(x$:ELEPHANT$)$ nor $mother(z$:CIRCUS-ANIMAL$)$. This contrasts with the unsorted case, where a variable always subsumes any other term. We refer to a variable that Σ-subsumes all members of a set of A-terms as a "variable generalization" of that set. We refer to a non-variable A-expression that Σ-subsumes all members of a set of A-expressions as a "structured generalization." A sorted generalization algorithm cannot assume that a variable generalization of a set of A-terms Σ-subsumes every structured generalization of that set. It must therefore compare the variable generalizations with the structured generalizations.

Because variable generalizations need not Σ-subsume structured generalizations, members of a CIG_Σ may have different structures. In the CIG_{Σ_1} of Example 2, one generalization is a variable while the other two are built from the function symbol "*mother*." Such structural differences can become more remarkable as the expressions being generalized grow more complex. That variable generalizations need not Σ-subsume structured generalizations is a result of *function polymorphism.* A function is polymorphic if terms built from that function's symbol may denote members of different sorts based on the arguments in the terms. *mother* is a polymorphic function, because whether a term built from "*mother*" denotes a member of the sort GOOD-COOK depends on the argument to *mother.* If the argument is *clyde* or *jumbo*, the term denotes a good cook; otherwise it does not. If *mother* were instead a monomorphic function, that is, if all mothers were known to be good cooks, the variable y:GOOD-COOK would Σ_1-subsume any structured generalization.

Example 3
Let $E = \{loves(clyde, mother(clyde)),$
$loves(jumbo, mother(jumbo))\}$.
Then a CIG_{Σ_1} of E is

$$\begin{aligned}\{&loves(x{:}\textsc{elephant},\ mother(x{:}\textsc{elephant})),\\ &loves(x{:}\textsc{elephant},\ mother(z{:}\textsc{circus-animal})),\\ &loves(x{:}\textsc{elephant},\ y{:}\textsc{good-cook}),\\ &loves(z{:}\textsc{circus-animal},\ mother(x{:}\textsc{elephant})),\\ &loves(z{:}\textsc{circus-animal},\ mother(z{:}\textsc{circus-animal})),\\ &loves(z{:}\textsc{circus-animal},\ y{:}\textsc{good-cook})\},\end{aligned}$$

and a most specific unsorted generalization of E is $loves(x, mother(x))$.

The interesting point of Example 3 is that the CIG_{Σ_1} of E is built from the cross-product of CIG_{Σ_1}s of the parts—the arguments—of the formulas in E. Care is required in building the CIG_{Σ_1} to ensure that variables repeat in exactly the right places, that is, that variable co-references are correct. There might also be variable generalizations of E; in this instance there are not, because E contains A-formulas rather than A-terms.

A Sorted Generalization Algorithm

The 2-SG algorithm, shown in Figure 1, computes the CIG_Σ of any pair of atomic A-expressions. The 2-SG algorithm uses a bijection ϕ from any triple of the form (t, s, τ) to a variable of sort τ.[3] To avoid accidental variable collisions, we further stipulate that the range of ϕ is disjoint from the alphabet from which the A-expressions input to the algorithm are built.

[3]This function is similar to the bijection between variable names and pairs of terms that is used in the anti-unification (unsorted generalization) algorithm of [Lassez *et al.*, 1988].

```
Input: Two atomic A-expressions, $\alpha_1$ and $\alpha_2$, and a sort theory $\Sigma$.
Output: A $CIG_\Sigma$ of $\alpha_1$ and $\alpha_2$.
1. If $\alpha_1 = \alpha_2$ then return $\{\alpha_1\}$.
2. If $\alpha_1$ and $\alpha_2$ are A-formulas with different predicate symbols or one is an A-term and the other an
   A-formula, then return $\{\ \}$.
3. If $\alpha_1$ and $\alpha_2$ are $p(s_1, s_2, ..., s_n)$ and $p(t_1, t_2, ..., t_n)$, respectively, where $p$ is a predicate/function symbol,
   and $s_1, s_2, ..., s_n, t_1, t_2, ..., t_n$ are A-terms, then
      Let Structured-Set = $\{p(r_1, r_2, ..., r_n) \mid r_i \in$ 2-SG$(s_i, t_i, \Sigma)$, for all $1 \leq i \leq n\}$.
   Otherwise,
      Let Structured-Set = $\{\ \}$.
4. If $\alpha_1$ and $\alpha_2$ are A-terms, then
      a. Let Variable-Set = $\{\phi(\alpha_1, \alpha_2, \tau) \mid \tau$ is any sort symbol occurring in $\Sigma$ for which $\Sigma \models \forall\tau(\alpha_1)$,
         $\Sigma \models \forall\tau(\alpha_2)$, and for every $e \in$ Structured-Set $\Sigma \not\models \forall\tau(e)\}$.
      b. For each variable $x{:}\tau \in$ Variable-Set:
            If there is another variable $y{:}\omega$ remaining in Variable-Set such that $\omega \preceq_\Sigma \tau$, then
               Remove $x{:}\tau$ from Variable-Set.
   Otherwise,
      Variable-Set = $\{\ \}$.
5. Return Structured-Set $\cup$ Variable-Set.
```

Figure 1: The 2-SG Algorithm

Theorem 4 (2-SG Algorithm Correctness) *Given oracles for determining whether or not any given universally-closed atomic S-formula follows from Σ and whether or not $\tau \preceq_\Sigma \omega$ for any two given sorts τ and ω, the 2-SG algorithm halts and returns a CIG_Σ of α_1 and α_2.*

The algorithm interacts with the sort theory through two kinds of taxonomic queries. It asks whether the sort theory entails a given universally-closed atomic S-formula, and it asks whether $\preceq_\Sigma$ is true of a given pair of sorts. Therefore oracles, or decision procedures, for these taxonomic queries are sufficient for computation of sorted generalization; if the taxonomic queries are decidable for a given sort theory, then sorted generalization relative to that sort theory is computable. The converse is also true; if sorted generalization is computable relative to a given sort theory, then the taxonomic queries are decidable relative to that theory because any algorithm for sorted generalization can be used to answer the queries. Therefore, oracles for the taxonomic queries are necessary, as well as sufficient, for sorted generalization. In general such oracles cannot be computed, but restrictions can be placed on the sort theory so that they can be.

The reader is encouraged to walk through the 2-SG algorithm's computation of the generalizations in the examples. Let's take the examples in order, considering how the algorithm addresses the major issue raised by each. Example 1 shows that a set of atomic A-expressions may have more than one maximally-specific sorted generalization. The algorithm therefore returns a set rather than a single generalization. Example 2 shows that variable generalizations are not guaranteed to Σ-subsume structured generalizations. Therefore, step 4 of the 2-SG algorithm compares variable generalizations with structured generalizations as it builds the Variable-Set. Notice that the algorithm never removes a structured generalization. Structured generalizations cannot Σ-subsume variable generalizations, and, because of the way they are built, no structured generalization built by 2-SG Σ-subsumes another. Example 3 illustrates the cross-product operation inherent in generalizing atomic A-expressions involving functions or predicates with more than one argument. It also shows that variables must be named properly to ensure correct variable co-references when cross-products are taken. Step 3 of the 2-SG algorithm builds structured generalizations of atomic A-expressions using the cross-product of the sorted generalizations of their components. Correct variable co-references are ensured by the bijection ϕ used for variable naming in step 4, the only step at which new variables are introduced. For example, if ϕ maps (*clyde*,*jumbo*,ELEPHANT) to x:ELEPHANT during generalization of the first arguments in the A-expressions *loves*(*clyde*,*mother*(*clyde*)) and *loves*(*jumbo*,*mother*(*jumbo*)), then during generalization of the second arguments, it maps an input to x:ELEPHANT if, and only if, that input is also (*clyde*,*jumbo*,ELEPHANT). Thus a variable repeats in a Σ-generalization only if a pair of A-terms, one from each input A-expression, repeats in the same way in the input A-expressions. Note that repetition of a pair of A-terms in the input A-expressions does not guarantee a corresponding variable repetition in every Σ-generalization, because the pair of A-terms may be generalized in more than one way.

The CIG_Σ Decomposition Theorem, which follows, shows that the 2-SG algorithm can be used repeatedly to generalize three or more atomic A-expressions.

Theorem 5 (**CIG_Σ Decomposition Theorem**) *Let A and B be sets of atomic A-expressions and let Σ be a sort theory. Then*

$$\{x \mid x \in CIG_\Sigma(a,b) \text{ for some } a \in CIG_\Sigma(A), \text{ and } b \in CIG_\Sigma(B)\}$$

is a complete set of Σ-generalizations of $A \cup B$.

The CIG_Σ Decomposition Theorem says nothing about whether the complete set of Σ-generalizations contains comparable A-expressions, that is, whether it is a CIG_Σ. In general it may not be. But given the oracles used in the 2-SG algorithm, the complete set can be filtered into a CIG_Σ by the following procedure. Let E be the set of A-expressions we wish to filter. From E choose distinct A-expressions e_1 and e_2 such that $e_1 \geq_\Sigma e_2$ and remove e_1 from E, until no such A-expressions remain in E. It follows from Lemma 2 that E is initially finite. Given the oracles, $\geq_\Sigma$ is decidable, so this procedure halts and returns a CIG_Σ that characterizes the same set as the original complete set of Σ-generalizations.

Algorithm and Problem Complexity

We have seen that sorted generalization is computable given oracles for the taxonomic queries. But given polynomial-time response to the taxonomic queries, the 2-SG algorithm requires exponential time in the worst case. Exponential time is a requirement of the problem itself because, in the worst case, the size of the CIG_Σ of two atomic A-expressions is exponential in the size of the smaller A-expression. The size is $O(t^n)$, where t is the number of sort symbols in Σ and n is the size of the smaller A-expression.

The exponential size is a result of the cross-product operation. We can represent CIG_Σs more compactly by using an explicit cross-product operator rather than computing cross-products. A CIG_Σ can always be represented in size polynomial in the size of the atomic A-expressions generalized and the number of sort symbols available. We can modify the 2-SG algorithm to compute such a compact representation. Unfortunately, the problem requires an exponential time algorithm (assuming $P \neq NP$) even when we allow a compact representation of the CIG_Σs and are guaranteed polynomial-time (in t and n) response to taxonomic queries. We call such a version of the problem "simplified sorted generalization."

Theorem 6 *Simplified sorted generalization is NP-hard.*

The proof of this theorem shows that 3-Satisfiability is polynomially reducible to simplified sorted generalization. This result does not depend on any particular compact representation of CIG_Σs. It holds for any representation of CIG_Σs with which we can test in polynomial time whether a given A-expression is a member of a given CIG_Σ.

This complexity result indicates that efficient response to taxonomic queries is not enough to allow efficient sorted generalization. It is possible, nonetheless, to restrict the taxonomic information in other ways to guarantee polynomial-time computation of CIG_Σs in a compact representation. It is possible as well to restrict the taxonomic information so that the size of all CIG_Σs is polynomial in the size of the input. The study of such restrictions is an interesting area for further work. It is also possible to compute the compact representation of a complete set of Σ-generalizations in polynomial time, given polynomial-time response to taxonomic queries, if the set is allowed to contain comparable A-expressions. Thus the complexity of simplified sorted generalization is in ensuring incomparability.

Applications

So far we have ordered atomic A-expressions by well-sorted substitutions, but we have said nothing about the semantic relationships between A-expressions. We could not have done so, as we have not even specified whether the variables in the A-expressions are universally or existentially quantified. It turns out that there are correspondences between the ordering $\geq_\Sigma$, based on well-sorted substitutions, and orderings based on entailment. It is because of these correspondences that we can use the algorithm for various reasoning tasks.

The first application we consider might arguably be classified as abduction or induction. Let O, a set of observations, be any finite set of sentences, and let B, a body of background information, be any finite satisfiable set of sentences. Informally, the problem is to find an hypothesis, h, that taken with the background information, B, explains each of the observations. More formally, by "explains" we mean that h is consistent with B, and h together with B entails $\wedge O$ ($\wedge O$ is the conjunction of the observations). We say that h is B-stronger than $\wedge O$ and $\wedge O$ is B-weaker than h. But there may be many such hypotheses, some better than others, so we would like to find the best one, or characterize the set of all hypotheses. The weakest hypothesis with respect to the background theory characterizes the set of all hypotheses, but again there may be no single weakest hypothesis. We therefore want a complete set of incomparable hypotheses, that is, a set H of sentences from an hypothesis language such that

- every member of H is B-stronger than $\wedge O$,
- any sentence in the hypothesis language that is B-stronger than $\wedge O$ is B-stronger than some member of H, and
- no member of H is B-weaker than any other one.

It is unclear whether to classify this problem as abduction or induction. In abduction there may be only a single observation, and the background information is normally a causal theory. Abduction typically finds hypotheses that explain the observation(s) through chaining, possibly extensive, using modus ponens. Induction, on the other hand, frequently works with many observations and produces a general, universally-quantified sentence that requires less inference to entail the observations. So whereas background information is crucial to abduction, it is less prominent, sometimes even absent, in induction. The current problem seems to include many instances of both abduction and induction.

Let's consider a special case of the problem, in which O may be any finite set of universally-closed atomic A-formulas and B may be any sort theory. We wish to explain the observations with a universally-closed atomic A-formula. As the reader may have guessed, the problem can be solved by sorted generalization. The CIG_Σ of the atomic A-formulas is a complete set of incomparable hypotheses, once its formulas are universally closed. Why? Because the instantiation ordering $\geq_\Sigma$ over atomic A-formulas is equivalent to the entailment ordering of those A-formulas, universally closed.

Theorem 7 *Let Σ be a sort theory and let α_1 and α_2 be atomic A-formulas. $\alpha_1 \geq_\Sigma \alpha_2$ if, and only if, $\{\forall\alpha_1\} \cup \Sigma \models \forall\alpha_2$.*

Therefore the universal closure of any member of the CIG_Σ of a set of observations is a maximally-weak hypothesis relative to Σ. If we allow disjunction in the hypothesis language, there are yet Σ-weaker hypotheses. In fact, there is a Σ-weakest one, which is the disjunction of the universal closures of the A-formulas in the CIG_Σ.

We have seen how atomic A-formulas are ordered by entailment when variables are universally quantified. The next problem we consider involves existentially quantified variables. The problem is vivification, a promising approach to efficient deduction [Borgida and Etherington, 1989; Levesque, 1988]. The premise of vivification is that much of the complexity of automated deduction arises from incomplete knowledge in knowledge bases (KBs), in particular from disjunctions leading to reasoning by cases. Vivification weakens the knowledge base in order to remove such disjunctions. To use an example from Levesque [1988], suppose our KB includes *age*(*fred*,71) $\vee$ *age*(*fred*,72). Many of the interesting results of this fact follow from Fred being in his early seventies or, even more generally, being a senior citizen. If we know that 71 and 72 belong to the category low-seventies, we may use sorted generalization to replace *age*(*fred*,71) $\vee$ *age*(*fred*,72) with $\exists x$:LOW-SEVENTIES *age*(*fred*,x:LOW-SEVENTIES). As another example, if our KB includes the disjunction *age*(*fred*,70) $\vee$ *age*(*joe*,70), and the background information specifies that Fred and Joe are both golfers and dentists, a sorted generalization algorithm will find two vivifications of the disjunction: $\exists x$:DENTIST *age*(x:DENTIST,70) and $\exists y$:GOLFER *age*(y:GOLFER,70). It is then necessary to decide which vivification will replace the disjunction. It may be best to replace it with the conjunction of the vivifications, because replacing one formula with another of equal or greater size can increase efficiency if the replacement eliminates some reasoning by cases. But does sorted generalization necessarily give the desirable vivid form of all disjunctions? Does it give the strongest consequents of a disjunction together with specified knowledge from the KB? We now see that it does, where the disjuncts and consequents are equivalent to ground or existentially quantified atomic A-formulas and the background information is taxonomic.

Vivification is a special case of the following truth-preserving inference problem. Let D and B be finite sets of sentences, where B is satisfiable. Find a set C of sentences from a specified "consequent" language such that

- every member of C is B-weaker than $\vee D$ ($\vee D$ is the disjunction of the sentences in D),
- any sentence in the consequent language that is B-weaker than $\vee D$ is B-weaker than some member of C, and
- no member of C is B-stronger than any other one.

Sorted generalization solves all instances of this problem in which D is a set of atomic A-formulas, existentially closed, B is a sort theory, and C is allowed to be any set of existentially-closed atomic A-formulas. It solves these instances because, according to the following theorem, the instantiation ordering $\geq_\Sigma$ on atomic A-formulas is also equivalent to an entailment ordering on those A-formulas, existentially closed.

Theorem 8 *Let Σ be a sort theory and let α_1 and α_2 be atomic A-formulas. $\alpha_1 \geq_\Sigma \alpha_2$ if, and only if, $\{\exists\alpha_2\} \cup \Sigma \models \exists\alpha_1$.*

Finally, various methods for falsity-preserving and truth-preserving generalization are used in algorithms for analogical reasoning. Reasoning with taxonomic information is a large part of many such methods. Therefore, based on Theorems 7 and 8, we suggest that sorted generalization is a useful tool for analogical reasoning, whether truth-preserving or falsity-preserving generalizations are needed. For example, Leishman's [1989] analogical tool uses truth-preserving generalization of terms with respect to taxonomic information in finding constrained partial correspondences between conceptual graphs. These correspondences produce a more general conceptual graph. It follows from the truth-preserving character of this generalization method that the taxonomic "concepts" used in the more general graph may be understood as existentially quantified sorted variables.

Relationship to Other Work

The present work differs in two major ways from other studies of generalization with taxonomic background information. First, the generality of the taxonomic theories distinguishes this work. A simple consequence of this generality is that a set of atomic A-expressions may have more than one minimal Σ-generalization. Therefore, the set of atomic A-expressions, ordered by Σ-subsumption, is not a lattice. This contrasts with other methods such as those described in [Michalski, 1983; Mitchell *et al.*, 1983]. A less obvious consequence, further distinguishing this work, is the polymorphism of function symbols. The need for polymorphism is evident; if fido is a dog and jumbo is an elephant, we should be able to let *mother*(*fido*) and *mother*(*jumbo*) denote individuals that do not belong to exactly the same sorts. But providing this needed flexibility adds the subtlety involving structured generalizations and variable generalizations, as illustrated in Example 2.

The second major distinction of this work is the pair of semantic properties asserted by Theorems 7 and 8. They assert falsity-preserving and truth-preserving correctness and completeness properties of sorted generalization. Plotkin's work on unsorted generalization begins with the semantic property that $\forall\alpha_1 \models \forall\alpha_2$ if, and only if, $\alpha_1 \geq \alpha_2$. We know of no such entailment-based result for methods of generalization with taxonomic background information, other than these presented here.

Finally, we relate sorted generalization to the *generalized subsumption* of Buntine [1988]. Generalized subsumption is defined over Horn clauses with respect to a logic program that acts as the background information. Because atomic A-formulas are equivalent to Horn clauses whose antecedents contain only taxonomic constraints on variables, we can compare Σ-subsumption, on which sorted generalization is based, with generalized subsumption, in cases where Σ consists entirely of Horn clauses. In these cases, Σ-subsumption and generalized subsumption are equivalent.

Because Σ-subsumption is based on purely taxonomic information, the 2-SG algorithm generalizes atomic A-expressions by decomposing them, generalizing their parts, and composing these generalizations. This compositional approach reduces the search for maximally-specific Σ-generalizations. It is reasonable to ask whether this approach can also reduce the search for generalization based on generalized subsumption with respect to arbitrary logic programs. It cannot be applied directly to generalization problems with such non-taxonomic theories, because in some cases it computes an incomplete set of generalizations. It may be possible, nevertheless, to use the 2-SG algorithm as part of a broader generalization procedure, much as algorithms for sorted unification are useful in improving the efficiency of deductive methods.

Acknowledgements

We thank the members of the Knowledge Representation and Reasoning Group and the Inductive Learning Group at the University of Illinois, and Tony Cohn, David Etherington, Mike Frazier, and Chris Matheus for helpful discussions and comments.

References

[Borgida and Etherington, 1989] A. Borgida and D. W. Etherington. Hierarchical knowledge bases and efficient disjunctive reasoning. In *First Int. Conf. on Principles of Knowledge Representation and Reasoning*, Toronto, Ontario, Canada, May 1989.

[Buntine, 1988] Wray Buntine. Generalized subsumption and its applications to induction and redundancy. *Artificial Intelligence*, 36(2):149–176, 1988.

[Frisch, 1989] A. M. Frisch. A general framework for sorted deduction: Fundamental results on hybrid reasoning. In *First Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 126–136, Toronto, Ontario, Canada, May 1989.

[Lassez and Marriott, 1987] J-L. Lassez and K. Marriott. Explicit representation of terms defined by counter examples. *Journal of Automated Reasoning*, 3:301–317, 1987.

[Lassez *et al.*, 1988] J-L. Lassez, M. J. Maher, and K. Marriott. Unification revisited. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 15, pages 587–625, Morgan Kaufmann Publishers, 1988.

[Leishman, 1989] D. Leishman. Analogy as a constrained partial correspondence over conceptual graphs. In *First Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 223–234, Toronto, Ontario, Canada, May 1989.

[Levesque, 1988] H. J. Levesque. Logic and the complexity of reasoning. *Journal of Philosophical Logic*, 17:355–389, 1988.

[Michalski, 1983] R. S. Michalski. A theory and methodology of inductive learning. In T.M. Mitchell R.S. Michalski, J.G. Carbonell, editor, *Machine Learning: An Artificial Intelligence Approach*, pages 82–132, Morgan Kaufmann Publishers, 1983.

[Mitchell *et al.*, 1983] T. G. Mitchell, P. E. Utgoff, and R. Banerji. Learning by experimentation: Acquiring and redefining problem solving heuristics. In T.M. Mitchell R.S. Michalski, J.G. Carbonell, editor, *Machine Learning: An Artificial Intelligence Approach*, chapter 5, pages 137–162, Morgan Kaufmann Publishers, 1983.

[Plotkin, 1970] G. D. Plotkin. *A Note on Inductive Generalization*, chapter 8, pages 153–163. Volume 5 of *Machine Intelligence*, Edinburgh University Press, Edinburgh, 1970.

[Plotkin, 1971] G. D. Plotkin. *A Further Note on Inductive Generalization*, chapter 8, pages 101–124. Volume 6 of *Machine Intelligence*, Edinburgh University Press, 1971.

[Reynolds, 1970] J. C. Reynolds. *Transformational Systems and Algebraic Structure of Atomic Formulas*, chapter 7, pages 135–52. Volume 5 of *Machine Intelligence*, Edinburgh University Press, Edinburgh, 1970.