ANTI-UNIFICATION IN CONSTRAINT LOGICS:
FOUNDATIONS AND APPLICATIONS TO LEARNABILITY IN
FIRST-ORDER LOGIC, TO SPEED-UP LEARNING, AND TO DEDUCTION

BY
CHARLES DAVID PAGE JR.
B.A., Clemson University, 1985
M.S., Clemson University, 1987

THESIS
Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1993

Urbana, Illinois
Unification is central to automated reasoning. Unification comes in a variety of forms, all of which compute (roughly stated) the greatest lower bound, or all maximal lower bounds, of any two or more syntactic objects in a partially-ordered set of such objects. The dual of unification is an operation called generalization, or anti-unification, which computes least or minimal upper bounds. As with unification, anti-unification comes in a variety of forms. The thesis of this dissertation is: anti-unification in its various forms is, like unification, a powerful tool for automated reasoning. In defense of the thesis, several forms of anti-unification in constraint logic, anti-unification relative to background information, are defined, and their semantic and computational properties are studied. It is shown that these forms of anti-unification are applicable to inductive logic programming (inductive learning of logic programs), speed-up learning, and knowledge base vivification (an approach to efficient deduction).
ACKNOWLEDGEMENTS

I thank my advisor, Alan Frisch, for teaching me about first-order logic, and for his guidance throughout my years of work on this dissertation. I thank him for countless discussions covering sorted logic, constraint logic, unification, and every other topic that appears in this dissertation.

I thank the other members of my thesis committee for their time, help, and encouragement: Gerald DeJong, Leonard Pitt, David Wilkins, and Marianne Winslett. I also thank Professor DeJong for teaching me about explanation-based learning and speed-up learning. I thank Professor Pitt for teaching me about computational learning theory, and I thank him for listening to my ideas and offering suggestions.

I thank Stephen Muggleton and Wray Buntine for their encouragement, and for the example of outstanding research that they have provided for me and for others.

I thank the many members of the Artificial Intelligence Group at the Beckman Institute, both faculty and students, for a tremendous research environment, for their critical evaluations, and for their friendship. I can only begin to list those whose knowledge and ideas have been invaluable to me. I thank Mike Frazier for many profitable discussions; I have learned much about computational learning theory, and computation theory in general, from him. I also thank Jon Gratch, Peter Haddawy, Rich Scherl, Scott Bennett, Melinda Gervasio, Steve Lavalle, Dan Oblinger, Eduardo Perez, Tomás Uribe, Larry Watanabe, Mike Mitchell, Ken Smith, and Henry Sutanto.

I thank Sharon Collins for all her help, especially with the red-bordered forms.

Most of all, I thank my family for love and encouragement, and for making so many sacrifices to help me complete this dissertation. I especially thank my wife—Lauren—and my parents. I also thank the members of Calvary Baptist Church, for they truly have become our extended family here in Illinois and have been a great source of support.
TABLE OF CONTENTS

Chapter
1 Introduction ................................................. 1
  1.1 The Thesis .............................................. 1
  1.2 What to Expect .......................................... 2
  1.3 Outline .................................................. 4
  1.4 Background: Ordinary Anti-unification ................. 5
    1.4.1 The Definition of Ordinary Anti-unification ......... 5
    1.4.2 Computing Ordinary Anti-Unification ................. 8
    1.4.3 Semantic Properties of Ordinary Anti-Unification ... 10

2 Constraint Logic and a Corresponding Instantiation Ordering .... 11
  2.1 Constrained Formulas and Constraint Theories ........... 12
  2.2 Additional Basic Definitions ............................ 13
  2.3 The General Instantiation Ordering ...................... 16
    2.3.1 Variable Abstraction ................................ 16
    2.3.2 The Ordering ........................................ 22
  2.4 Initial Models and Strong Compactness ................... 30

3 Special Classes of Constrained Formulas ..................... 39
  3.1 A Characterization in the Absence of Equality .......... 39
  3.2 A Characterization that Allows Limited Equality ......... 46
  3.3 The Instantiation Ordering for Sorted Logic ............. 52
  3.4 The Ordering Used in E-Unification and E-Anti-Unification . 57
  3.5 Orderings for E-Anti-Unification, Simple Constrained Anti-Unification, and Constrained E-Anti-Unification ........ 59

4 Algebraic and Combinatorial Properties of the Instantiation Ordering on Constrained Formulas ......................... 65
  4.1 Definitions ............................................. 66
  4.2 Cover Sets, Chains, and Antichains ...................... 69
    4.2.1 Cover Sets and Chains of Ordinary Atoms ............ 72
    4.2.2 Cover Sets and Chains of Constrained Atoms without Equality ... 75
    4.2.3 Ascending Chains of Sorted Atoms .................... 87
  4.3 Sufficient Conditions for GLBs and LUBs .................. 88
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>VC-Dimension of Ordinary Atoms</td>
<td>93</td>
</tr>
<tr>
<td>4.5</td>
<td>Directions for Further Work</td>
<td>94</td>
</tr>
<tr>
<td>5</td>
<td>Computing Sorted Anti-Unification</td>
<td>95</td>
</tr>
<tr>
<td>5.1</td>
<td>Preliminaries</td>
<td>95</td>
</tr>
<tr>
<td>5.2</td>
<td>A Sorted Anti-Unification Algorithm</td>
<td>101</td>
</tr>
<tr>
<td>5.3</td>
<td>Problem Complexity</td>
<td>107</td>
</tr>
<tr>
<td>6</td>
<td>Computing Simple Constrained Anti-Unification</td>
<td>115</td>
</tr>
<tr>
<td>6.1</td>
<td>Preliminaries</td>
<td>115</td>
</tr>
<tr>
<td>6.2</td>
<td>A Simple Constrained Anti-Unification Algorithm</td>
<td>117</td>
</tr>
<tr>
<td>6.3</td>
<td>Analysis: Extended Sorted Anti-Unification</td>
<td>124</td>
</tr>
<tr>
<td>6.4</td>
<td>VC-Dimension of Simple Constrained Atoms</td>
<td>127</td>
</tr>
<tr>
<td>7</td>
<td>E-Anti-Unification and E-Anti-Unification with Constraints</td>
<td>128</td>
</tr>
<tr>
<td>7.1</td>
<td>E-Anti-Unification</td>
<td>128</td>
</tr>
<tr>
<td>7.2</td>
<td>Partitioned Constrained E-Anti-Unification</td>
<td>134</td>
</tr>
<tr>
<td>8</td>
<td>Applications of Anti-Unification</td>
<td>144</td>
</tr>
<tr>
<td>8.1</td>
<td>Learnability Results for Inductive Logic Programming</td>
<td>144</td>
</tr>
<tr>
<td>8.1.1</td>
<td>The Learnability of Sorted Atoms</td>
<td>147</td>
</tr>
<tr>
<td>8.1.2</td>
<td>The Learnability of Simple Constrained Atoms</td>
<td>152</td>
</tr>
<tr>
<td>8.1.3</td>
<td>The Learnability of Conjunctions of Simple Constrained Atoms</td>
<td>155</td>
</tr>
<tr>
<td>8.1.4</td>
<td>The Learnability of Conjunctions of Simple Constrained Atoms Relative to Partitioned Constraint Theories</td>
<td>166</td>
</tr>
<tr>
<td>8.2</td>
<td>Speed-Up Learning: Natarajan’s Framework</td>
<td>178</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Defining the Framework</td>
<td>178</td>
</tr>
<tr>
<td>8.2.2</td>
<td>An Application to Symbolic Integration</td>
<td>181</td>
</tr>
<tr>
<td>8.3</td>
<td>Knowledge Base Vivification</td>
<td>185</td>
</tr>
<tr>
<td>9</td>
<td>Conclusions and Directions for Further Research</td>
<td>189</td>
</tr>
<tr>
<td>Appendix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A Short Glossary</td>
<td>192</td>
</tr>
<tr>
<td>A.1</td>
<td>Restricted Constrained Atoms</td>
<td>192</td>
</tr>
<tr>
<td>A.2</td>
<td>Anti-Unification Operations</td>
<td>193</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>194</td>
</tr>
<tr>
<td>Vita</td>
<td></td>
<td>201</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

1.1 Ordinary Anti-Unification (OA) Algorithm .................. 8

5.1 The Sorted Anti-Unification (SA) Algorithm .................. 102
5.2 The Two Sources of Multiple Minimal Sorted Generalizations .... 107

6.1 The Simple Constrained Anti-Unification (SCA) Algorithm ........ 118
6.2 The Four Anti-Unification Problem Classes .................. 125

8.1 A Portion of the Background Theory Used by LEX ............ 183
Chapter 1

Introduction

1.1 The Thesis

Unification plays an important role in many automated reasoning systems, particularly those that perform deduction. It comes in various forms including term unification, string unification, unification with built-in equality (E-unification), sorted unification, constrained 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 set of syntactic objects that is partially-ordered by instantiation. Turning unification on its head yields an operation called anti-unification, or generalization, that computes minimal upper bounds.\(^1\) As with unification, one can imagine many forms of anti-unification. Ordinary anti-unification is the dual of ordinary unification, and it has been studied by Plotkin [56; 57] and Reynolds [60], and more recently by Lassez, Maher, and Marriott [40; 39]. The thesis is that *anti-unification in its various forms is, like unification, a powerful tool for automated reasoning.*

In defense of the thesis, the dissertation examines the semantic and computational properties of several forms of anti-unification for *constraint logic*, which take into account

\(^1\) While “generalization” actually has a broad range of meanings, anti-unification has only this one. Therefore, we use the term *anti-unification* throughout the dissertation. Nevertheless, we refer to the results of *anti-unification* as *generalizations*. 
background information in the form of a constraint theory. Based on this examination, the dissertation shows that major components of several reasoning systems may be viewed as anti-unification operators, and that this view allows one to predict the behavior of these systems and to evaluate decisions made in their design. Furthermore, the dissertation demonstrates that this view facilitates construction of more powerful reasoning systems, often by extending existing methods. In particular, the dissertation uses the results it presents about anti-unification to provide new results for inductive logic programming, speed-up learning, and knowledge base vivification (an approach to efficient deduction).

1.2 What to Expect

For the reader interested in inductive logic programming, Chapter 8 of the dissertation presents several results on the pac-learnability of logic programs. These results include the following:

- The class of atomic formulas, or atoms, is pac-learnable.
- The class of conjunctions of atoms, which also may be thought of as the class of Prolog databases, is pac-learnable only if the class of DNF formulas is pac-learnable. But with the additional help of subset queries, the class of Prolog databases is pac-learnable even though analogous algorithms fail to pac-learn the class of DNF formulas.
- The class of atoms of sorted logic is not pac-learnable, relative to very simple taxonomic background theories, under a basic complexity-theoretic assumption. The class of atoms of sorted logic may be viewed as a restricted class of Horn clauses that closely resembles languages used by early inductive algorithms such as AQ, INDUCE, and the inductive component of LEX.
- The class of simple constrained atoms is pac-learnable, even relative to rather complex background theories. Interestingly, this class includes the class of atoms of sorted logic. The class of simple constrained atoms may be viewed as the class
of non-recursive Horn clauses where every term in the antecedent appears in the consequent, for example

\[ \text{metal-connection}(x, y) \land \text{higher-potential}(x, y) \rightarrow \text{current-flow}(x, y) \]

- The class of conjunctions of simple constrained atoms is pac-learnable, with the additional help of subset queries, even relative to rather complex background theories. This class may be thought of as the class of non-recursive completions of (possibly recursive) Prolog programs, such that for each clause in the completion, every term in the antecedent of the clause appears in the consequent.

- An extension of the class of conjunctions of simple constrained atoms that allows the representation of structural concepts, such as Blocks World descriptions, is pac-learnable, with the additional help of subset queries, under certain restricted conditions. This learnability result actually implies the pac-learnability of disjunctive concepts in structural domains.

For the reader interested in speed-up learning, the primary result in Chapter 8 applies to B. K. Natarajan’s formalization of the speed-up learning task. Natarajan has shown that if control rules for a problem solving task can be pac-learned, subject to an additional constraint, then successful speed-up learning in his formalization can be achieved. Chapter 8 shows that the pac-learning algorithms presented in this dissertation meet Natarajan’s additional constraint, because they are based on anti-unification. Therefore, these learning algorithms can be used for provably successful speed-up learning.

For the reader interested in automated deduction, Chapter 8 of the dissertation shows that anti-unification can be used for knowledge-base vivification. Knowledge-base vivification attempts to increase the efficiency of deduction by eliminating unnecessary disjunctions from a knowledge base.

Finally, for the reader interested in the use of constraints in programming languages, database systems, or knowledge representation, Chapter 2 presents a general instantiation ordering for formulas of constraint logic. The ordinary instantiation ordering for
formulas of ordinary first-order logic is central to the applications of first-order logic in computer science, primarily because of its semantic and computational properties. The instantiation ordering for formulas of constraint logic has analogous semantic properties and, in restricted cases, has analogous computational properties.

1.3 Outline

The remainder of this chapter reviews results about ordinary anti-unification. Ordinary anti-unification is a purely syntactic operation that operates solely on the basis of expression structure. Chapter 2 defines a language called constraint logic and an instantiation ordering for the language, and it presents valuable semantic properties of this ordering. Anti-unification in constraint logic operates on the atomic formulas, or atoms, in this language, based on this instantiation ordering. Unlike ordinary anti-unification, anti-unification in constraint logic takes into account a background theory, called a constraint theory, in addition to expression structure. Chapter 3 defines several useful subsets of constraint logic and identifies simplifications of the instantiation ordering for these subsets. Each specialized ordering retains all the semantic properties of the general instantiation ordering for constraint logic. Chapter 4 considers algebraic and combinatorial properties of the atoms in these subsets of constraint logic; important algebraic, combinatorial, and computational properties of the various anti-unification operations follow from the properties proven in this chapter. Chapters 5, 6, and 7 examine the computation of the various anti-unification operations that correspond to these subsets of constraint logic. These operations are sorted anti-unification, extended sorted anti-unification, simple constrained anti-unification, E-anti-unification, and partitioned constrained E-anti-unification. (A glossary at the end of the dissertation also provides brief definitions of these operations, for quick reference.) The semantic properties of these operations follow from the semantic properties, proven in Chapter 2, of the instantiation ordering for constraint logic. Chapter 8 discusses the applications of the anti-unification operations to inductive logic programming, speed-up learning, and knowledge base vivification, based on the semantic,
algebraic, combinatorial, and computational properties of these operations. Chapter 9 presents the conclusions of the dissertation and identifies future research directions.

Readers who are much more interested in the applications of anti-unification than in the theoretical properties that make these applications possible may wish to reach Chapter 8 sooner. Even so, these readers should begin with Chapters 1 through 3, in order to understand constraint logic, the instantiation ordering on constrained formulas, and the subsets of constraint logic used in the dissertation, but they may skip the proofs in these chapters. These chapters have a generous supply of examples, so that constraint logic, its subsets, and its instantiation ordering can be understood without reading the proofs. These readers may then proceed to Chapter 8, referring back to Chapters 4 through 7, as needed, for the details of the anti-unification algorithms and the algebraic and combinatorial properties of the subsets of constraint logic.

1.4 Background: Ordinary Anti-Unification

1.4.1 The Definition of Ordinary Anti-Unification

Unification is based on the instantiation ordering defined by ordinary substitutions; one atomic formula, or atom, $e_1$, is an instance of another, $e_2$, if and only if $e_2\theta = e_1$ for some substitution $\theta$. We also say that $e_2$ is more general than $e_1$, and we write $e_2 \geq e_1$. The most general common instance, $e$, of a finite set, $E$, of atoms may be found with unification and characterizes the common instances of $E$: an atom is a common instance of $E$ if and only if it is an instance of $e$. Likewise, the least general generalization (LGG or least generalization), $g$, of $E$ characterizes the atoms that are more general than every atom in $E$: an atom is more general than every member of $E$ if and only if it is more general than $g$. The ordinary anti-unification operation computes this LGG of a finite set of atoms.

Algorithms for ordinary anti-unification, as well as properties of this operation, have appeared elsewhere [56; 60; 39]. The following example illustrates the interesting aspects of computing ordinary anti-unification.
Example 1

Let \( E = \{\text{loves(clyde,mom(clyde)), loves(jumbo,mom(jumbo))}\} \)

Then \( \text{loves}(x, \text{mom}(x)) \) is an LGG of \( E \).

Notice that the common structure of the atoms in \( E \) is preserved in the LGG of \( E \). Notice also that the variable \( x \) repeats in the LGG in the same way that clyde and jumbo repeat in the first and second atoms, respectively, of \( E \).

Although the instantiation ordering on ordinary atomic formulas appears to be a partial ordering, it is not; rather, it is a quasi-ordering, or preorder [56]. It is not a partial-ordering because it is not anti-symmetric: for example, \( \text{loves}(x, \text{mom}(x)) \) and \( \text{loves}(y, \text{mom}(y)) \) are distinct atoms, yet each is more general than the other. A quasi-ordering on a given set is not necessarily anti-symmetric, but instead divides the set into equivalence classes. The quasi-ordering then forms a partial-ordering on these equivalence classes. More concisely, a relation is a quasi-ordering if and only if it is reflexive and transitive. It is straightforward to verify that two atoms are in the same equivalence class in the instantiation ordering if and only if they are variants, that is, they are the same modulo the renaming of their variables.

Because the instantiation ordering is a quasi-ordering, a set of atoms actually may have infinitely-many LGGs, but they are variants. Since the LGGs of a set of atoms are variants, for simplicity we sometimes speak of the LGG of a set of atoms.

LGGs have several interesting properties. One property worth noting is that any set (finite or infinite) consisting of atoms built from the same predicate symbol has an LGG; any set containing atoms built from different predicate symbols has no generalizations, and so has no LGG [39]. At times it is convenient to add to the set of atoms an element \( \top \) that is defined to be more general than every atom, and an element \( \bot \) defined to be an instance of every atom. Having added \( \top \), every set of atoms has an LGG; the LGG of a set of atoms is \( \top \) if and only if the set contains atoms built from different predicate symbols or the set contains \( \top \) itself. Another interesting property of LGGs is that for any set \( E \) of two or more atoms and any nonempty subsets \( E_1 \) and \( E_2 \) of \( E \) such that
\[ E = E_1 \cup E_2, \quad \text{LGG}(E) = \text{LGG}(\text{LGG}(E_1), \text{LGG}(E_2)). \] Thus, for example, an algorithm that computes the \text{LGG} of any pair of atoms may be used repeatedly to compute the \text{LGG} of any finite set of atoms [56; 60; 39].

Another property of \text{LGG}s, or more generally of generalizations, regards their size. We take the size of any atom to be the number of variable and function symbol occurrences in the atom, where constants are understood to be nullary function symbols.\(^2\) Because each function symbol or variable in an atom is the first symbol of exactly one term in the atom, the size of an atom is the number of term occurrences it contains. Notice that a substitution can never decrease the size of an atom, since it replaces variables (size 1) with terms (size at least 1). Because \( e_1 \geq e_2 \) only if some substitution maps \( e_1 \) to \( e_2 \), the size of \( e_1 \) is no greater than the size of \( e_2 \). Because the \text{LGG} of a set \( E \) is an atom that is more general than every atom in \( E \), the size of the \text{LGG} is no greater than the size of the smallest atom in \( E \).

Typically, the numbers of variables and function symbols of any given arity in first-order logic are each taken to be countably infinite. Therefore, as a technical note, the size of the actual representation of a logical expression, or set of logical expressions, can be larger, by a logarithmic factor, than the size as we have defined it. For example, if every symbol in an expression of size \( n \) is distinct, then \( \log_2 n \) bits are required to represent each symbol, so the representation of the entire expression requires \( n \log_2 n \) bits. In the analysis of algorithms, the size measure we have defined yields a \textit{unit cost measure} of complexity, whereas a measure that takes into account the size of the representation of variables and function symbols yields a \textit{logarithmic cost measure} of complexity [44]. The algorithm analyses in the dissertation use the unit cost measure. Alternatively, for the algorithms and complexity results in the dissertation, the \( O(\ ) \) notation may be read as \( \tilde{O}(\ ) \) instead. The \( \tilde{O}(\ ) \) notation differs from \( O(\ ) \) only in that it ignores logarithmic factors [5].

\(^2\)More generally, we take the size of any logical expression to be the number of occurrences of variables, function symbols, and logical connectives in the expression. We take the size of a set \( E \) of logical expressions to be the sum of the sizes of the expressions in \( E \).
Ordinary Anti-Unification (OA) Algorithm

Input: A tuple $E$ of terms or atoms.
Output: The LGG of $E$.

1. If $E$ contains $\top$, contains both terms and atoms, or contains atoms built from different predicate symbols, return $\top$.

2. If $E$ is of the form \{p(t_{1,1}, ..., t_{1,n}), ..., p(t_{m,1}, ..., t_{m,n})\}, that is, if all members of $E$ are built from the same predicate symbol or primary function symbol, $p$, then return $p(r_1, ..., r_n)$ where $r_i = LGG(t_{1,i}, ..., t_{m,i})$, for all $1 \leq i \leq n$.

3. If $E = \{t_1, ..., t_m\}$ where $t_1, ..., t_m$ are terms that are not all built from the same primary function symbol, return $\phi(t_1, ..., t_m)$.

Figure 1.1: Ordinary Anti-Unification (OA) Algorithm

We have stated that a set of atoms has an LGG, other than $\top$, if and only if all the atoms are built from the same predicate. Ordinary anti-unification can be applied to literals, more generally. A set of literals has an LGG, other than $\top$, if and only if all the literals have the same predicate and sign (all are atoms or all are negated atoms). The LGG of a set of negated atoms is simply the negation of the LGG of the atoms (literals with the negation sign removed) in the set.

1.4.2 Computing Ordinary Anti-Unification

Ordinary anti-unification can be computed efficiently. More specifically, existing algorithms for ordinary anti-unification can be implemented to compute the LGG of a finite set of atoms in time quadratic in the sum of the atom sizes. For example, consider the algorithm for ordinary anti-unification in Figure 1.1 (modified from [39]),\footnote{Algorithms provided by Plotkin [56] and Reynolds [60] also run in quadratic time.} it was originally presented to generalize two terms at a time, but it is presented here to generalize an arbitrary number of terms or atoms at a time. The function $\phi$ is a bijection used by the algorithm to map tuples of terms to variables.
The complexity of the algorithm depends upon our implementation of the function $\phi$. One way to compute the function $\phi$ during any given run of the ordinary anti-unification algorithm is as follows. When the function is called with a tuple of terms that has not been used in a previous call to $\phi$, a new variable is generated, and an association between the tuple and the variable is recorded. When $\phi$ is called with a tuple of terms that has already been used, the variable associated with that tuple is returned. The time required by such an implementation of $\phi$ is bounded above by the greater of the maximum time required for storing an association and the maximum time required for retrieving an association. Using simple linear storage, associations can be stored and retrieved in time $O(|E|)$, since the sum of the sizes of all the tuples that might be stored is $|E|$. It follows that, using this implementation of the function $\phi$, the ordinary anti-unification algorithm runs in time $O(|E|^2)$.

It is worth at least noting that if we are unconcerned about the length of variable names in the computed $LGG$s, we can implement the function $\phi$ to take time linear in the size of its input (rather than in the size of $E$). This yields an implementation of the ordinary anti-unification algorithm above that runs in linear time. The method for computing $\phi$ is as follows. For any given tuple of terms $\langle t_1, \ldots, t_n \rangle$, let $\phi$ return the variable $x_{\langle t_1, \ldots, t_n \rangle}$. Of course, such a method may produce extremely long variable names, particularly if the anti-unification algorithm is repeatedly applied. Therefore, in analyzing the algorithm’s complexity it is more reasonable to count characters or bits used in representing an atom, rather than taking the size of an atom to be the number of variable and function symbols. In other words, a logarithmic cost measure is more appropriate. Even with a logarithmic cost measure of $|E|$, the ordinary anti-unification algorithm runs in time linear in $|E|$ (the sum of the number of characters or bits used to represent the atoms in $E$). The disadvantage is that while the $LGG$ of $E$ has no more variable and function symbols than any member of $E$, its representation may be longer. In fact, the representation of the $LGG$ of $E$ may be as long as the representation of $E$ itself.
1.4.3 Semantic Properties of Ordinary Anti-Unification

Ordinary unification and ordinary anti-unification are useful because of the following semantic properties of the instantiation ordering on ordinary atoms. These properties are well-known and are relatively straightforward to verify. Let $\alpha_1$ and $\alpha_2$ be atoms. First, if $\alpha_1 \geq \alpha_2$ then $\alpha_{1gr} \supseteq \alpha_{2gr}$ (every ground instance of $\alpha_2$ is a ground instance of $\alpha_1$). Second, all three of the statements below are equivalent:\textsuperscript{4}

- $\alpha_1 \geq \alpha_2$
- $\forall \alpha_1 \models \forall \alpha_2$
- $\exists \alpha_2 \models \exists \alpha_1$

We will seek similar semantic properties for the instantiation ordering on atoms of constraint logic, which is the basis for anti-unification in constraint logic.

\textsuperscript{4}Throughout the dissertation, where $\phi$ is an expression we use $\forall \phi$ to denote the \textit{universal closure} of $\phi$—the result of universally quantifying all free variables in $\phi$. Similarly we use $\exists \phi$ to denote the \textit{existential closure} of $\phi$. 

10


Chapter 2

Constraint Logic and a Corresponding Instantiation Ordering

Beginning with Plotkin [57], researchers have considered generalization (in a loose sense of the word) relative to background information. One advantage of this approach is that the background information often allows more specific generalizations to be obtained. For example, generalizing “Clyde is gray” and “Jumbo is gray” yields the very general assertion, “Everything is gray,” while generalization with respect to the background information that Clyde and Jumbo are elephants yields the more specific result, “Elephants are gray.” This advantage motivates the consideration of anti-unification relative to background information.

Ordinary anti-unification is defined by a language of expressions, the atoms, together with the instantiation ordering based on ordinary substitutions. Various forms of anti-unification for constraint logic, operations that act relative to background information, can be defined in a similar fashion. All are based on the definitions of constrained formulas and an instantiation ordering on them, which we now present. Although our anti-unification operations act on constrained atoms only, we define, more generally, arbitrary constrained formulas and the instantiation ordering on these.
2.1 Constrained Formulas and Constraint Theories

Roughly stated, a constrained formula is a quantifier-free formula, built from ordinary predicates, whose terms have their denotations restricted by constraints. A constraint may be any logical formula built from constraint predicates. The set of constraint predicates is disjoint from the set of ordinary predicates. Any predicate except the interpreted equality predicate ("=") may serve as an ordinary predicate; any predicate not serving as an ordinary predicate may be used as a constraint predicate. For clarity, constraint predicates (except equality) are written in small capital letters, e.g. “ELEPHANT.” Following work in constraint logic programming, we represent a constrained formula by an ordinary formula, called the head, followed by a slash and then the constraint. For example, the following is a constrained formula.

\[ \text{eats}(x, \text{peanuts})/\text{ELEPHANT}(x) \]

Notice that the head of this constrained formula is an atomic formula, or atom; therefore, we also call this constrained formula, more specifically, a constrained atom. If the head of a constrained formula is a literal, we also refer to the constrained formula as a constrained literal.

But what does a constrained formula mean? In all the applications of constrained formulas that this author knows about, the variables in the formulas are either all universally quantified or all existentially quantified.\(^1\) Where \( \phi \) is any formula (ordinary or constrained), we say that the universal closure of \( \phi \) is the result of universally quantifying all free variables in \( \phi \), and we denote the universal closure of \( \phi \) by \( \forall \phi \). Similarly, the existential closure of \( \phi \) is the result of existentially quantifying all free variables in \( \phi \), and it is denoted by \( \exists \phi \). If \( \phi/C \) is a constrained formula, then \( \forall(\phi/C) \) is defined to be \( \forall(C \rightarrow \phi) \), and \( \exists(\phi/C) \) is defined to be \( \exists(C \land \phi) \). Thus, for example, the universal closure

\(^1\)In fact, substantial difficulties confront attempts to mix the quantifiers on constrained formulas.
of \textit{eats} (x, \textit{peanuts}) / \textit{ELEPHANT} (x) says, “all elephants eat peanuts”, while the existential closure says, “some elephant eats peanuts”.

Operations on constrained formulas act with respect to background information about the constraint predicates. This background information takes the form of a \textit{constraint theory}. A constraint theory is any (possibly infinite) satisfiable theory, expressed in first-order logic with equality, whose only predicates are constraint predicates. The following are two examples of constraint theories. Notice that the first contains unary constraint predicates only. Such constraint theories have been used frequently with \textit{sorted logics}, where they are called \textit{sort theories} and the unary predicates are called \textit{sorts} [15; 16; 17; 14; 23; 25; 24; 66; 67; 68; 65]. Therefore, we also call this theory a \textit{sort theory}.

$$\Sigma_1 = \{ \forall x \ \text{UNIV} (x), \ \text{ELEPHANT} (clyde), \ \text{IN-CIRCUS} (clyde), \ \text{GRAY} (mom(clyde)),$$
$$\text{ELEPHANT} (jumbo), \ \text{IN-CIRCUS} (jumbo), \ \text{GRAY} (mom(jumbo)),$$
$$\forall x \ \text{ELEPHANT} (x) \rightarrow \text{ELEPHANT} (mom(x)),$$
$$\forall x \ \text{ELEPHANT} (x) \rightarrow \text{MAMMAL} (x) \}.$$ 

$$\Sigma_2 = \{ \text{political-party} (arnold) = \text{republican-party},$$
$$\text{spouse} (maria) = \text{arnold}, \ \text{spouse} (\text{arnold}) = \text{maria},$$
$$\text{BIGGER} (\text{arnold}, \text{maria}), \ \text{KENNEDY} (\text{maria}),$$
$$\forall x \ \text{KENNEDY} (x) \rightarrow (\text{political-party} (x) = \text{democratic-party}),$$
$$\forall x \ (\text{political-party} (x) = \text{political-party} (\text{spouse} (x))) \rightarrow$$
$$\text{DIPLOMATIC} (x) \land \text{DIPLOMATIC} (\text{spouse} (x)),$$
$$\forall x \ (\text{political-party} (x) = \text{political-party} (\text{spouse} (x))) \land$$
$$\text{BIGGER} (\text{spouse} (x), x) \rightarrow \text{TOUGH} (x) \}.$$ 

\section{2.2 Additional Basic Definitions}

The following definitions are used throughout this chapter and the rest of the dissertation. As is standard, logical expressions may be terms, formulas, or sentences, where a sentence is a formula with no free variables. A theory is a set, or conjunction, of sentences, so any finite theory also may be viewed as a single sentence. A \textit{value assignment} is an
assignment of individuals from the domain of a given model to variables. Where \( g \) is any value assignment, \( g[x \mapsto d] \) denotes the value assignment that agrees with \( g \) on the assignment to all variables except (possibly) \( x \) and that assigns \( d \) to \( x \). Where \( M \) is a model, \( D_M \) denotes the domain of \( M \). Where \( \phi \) is any logical expression that contains no free variables, and \( M \) is any model, we take \( \llbracket \phi \rrbracket^M \) to be the semantic value of \( \phi \) with respect to \( M \); \( \llbracket \phi \rrbracket^M \) is an individual in \( D_M \) if \( \phi \) is a ground term, and it is a truth value (True or False) if \( \phi \) is a sentence. If \( \phi \) contains free variables, then its semantic value must be taken with respect to both a model and a value assignment. Where \( \phi \) is any logical expression, and \( M \) and \( g \) are any model and value assignment, respectively, we take \( \llbracket \phi \rrbracket^{M,g} \) to be the semantic value of \( \phi \) with respect to \( M \) and \( g \); \( \llbracket \phi \rrbracket^{M,g} \) is an individual in \( D_M \) if \( \phi \) is a term, and it is a truth value if \( \phi \) is a formula. (If \( \phi \) has no free variables, then for any value assignment \( g \) we have \( \llbracket \phi \rrbracket^{M,g} = \llbracket \phi \rrbracket^M \), making \( g \) superfluous.) An argument or sequent is a pair \( (\Sigma, \psi) \), where \( \Sigma \) is any theory and \( \psi \) is any logical sentence; such a sequent is true, or valid, if and only if any model that satisfies \( \Sigma \) also satisfies \( \psi \). We will often write such a sequent as \( \Sigma \Rightarrow \psi \). If such a sequent is valid, we say that \( \Sigma \) entails \( \psi \), and we write \( \Sigma \models \psi \). Where \( \phi_1 \) and \( \phi_2 \) are formulas, we also write \( \phi_1 \models \phi_2 \) to say that any model and value assignment that together satisfy \( \phi_1 \) also satisfy \( \phi_2 \).

Let \( \Sigma \) be a constraint theory, and let \( M \) be a model. We say that \( M \) is a \( \Sigma \)-model if and only if \( M \) satisfies \( \Sigma \). We say that a constraint that is satisfied by some \( \Sigma \)-model and some value assignment is \( \Sigma \)-satisfiable. A constrained formula is said to be admissible with respect to \( \Sigma \), or \( \Sigma \)-admissible, if its constraint is \( \Sigma \)-satisfiable. Otherwise, it is \( \Sigma \)-inadmissible. Where \( \Sigma \) is, more generally, any (possibly infinite) first-order theory, and \( \psi_1 \) and \( \psi_2 \) are logical sentences, we say that \( \psi_1 \Sigma \)-entails \( \psi_2 \) if and only if \( \Sigma \cup \{ \psi_1 \} \models \psi_2 \), and we write \( \psi_1 \models_\Sigma \psi_2 \). More generally yet, where \( \phi_1 \) and \( \phi_2 \) are logical formulas, we say that \( \phi_1 \Sigma \)-entails \( \phi_2 \) if and only if any \( \Sigma \)-model and value assignment that satisfy \( \phi_1 \) also satisfy \( \phi_2 \). We say that two formulas are \( \Sigma \)-equivalent if and only if each \( \Sigma \)-entails the other. Where \( P \) and \( Q \) are constraint predicates of the same arity, \( n \), and \( \Sigma \) is a constraint theory, we write \( P \preceq_\Sigma Q \) if and only if \( P(x_1, ..., x_n) \models_\Sigma Q(x_1, ..., x_n) \), or, equivalently, \( \Sigma \models \forall(P(x_1, ..., x_n) \rightarrow Q(x_1, ..., x_n)) \). More generally, we may apply the ordering \( \preceq_\Sigma \) to
arbitrary constraints. Let $C_1$ and $C_2$ be constraints. Then we write $C_1 \preceq \Sigma C_2$ if and only if $C_1 \models_\Sigma C_2$, or, equivalently, $\Sigma \models \forall(C_1 \rightarrow C_2)$. It is also straightforward to verify that $\preceq_\Sigma$ is a quasi-ordering on constraint predicates, and more generally on constraints, given any choice of $\Sigma$.

At times it is also useful to speak of constrained terms. A constrained term, in contrast with a constrained formula, has a term for its head. The meaning of a constrained term $t/C$ is taken with respect to any given $\Sigma$-model $M$, as follows:

$$[[t/C]]^M = \{d \in D_M \mid [[t]]^{M,g} = d \text{ for some value assignment } g \text{ such that } [[C]]^{M,g} = True\}$$

Thus a constrained term denotes some subset of the domain of a given model. For example, the constrained term $x/ELEPHANT(x)$ denotes the set of all elephants in a given model. A constrained expression is a constrained term or constrained formula.

Where $\phi$ is any logical expression, $\operatorname{VARS}(\phi)$ denotes the free variables in $\phi$. Where $\theta$ is any substitution, $\operatorname{DOM}(\theta)$ denotes the set of all variables $x$ such that $x \theta \neq x$. As is standard, “$\cdot$” denotes function composition. More specifically, for any substitutions $\theta$ and $\sigma$, $\theta \cdot \sigma$ denotes the substitution that is the composition of $\theta$ and $\sigma$, defined as follows. For any expression $e$, $e(\theta \cdot \sigma) = (e\theta)\sigma$, which we sometimes write as simply $e\theta\sigma$.

Finally, a review of the following basic definitions about first-order logic may be helpful. The Herbrand Universe for a given alphabet (set of function symbols and predicate symbols) is the set of ground terms that can be built from the function symbols (including constants) in the alphabet. The Herbrand Base is the set of ground atoms that can be built from the alphabet, that is, from the predicate symbols in the alphabet and from the terms in the Herbrand Universe. A Herbrand model is a model whose domain is the Herbrand Universe; alternatively, it may be viewed as a truth assignment over the Herbrand Base. A sentence is in Prenex Form if no logical connective precedes a quantifier, that is, the quantifiers all appear at the start of the sentence. A theory $\Sigma$ is in Skolem Normal Form if its sentences are in Prenex Form and contain no existential quantifiers. A sequent $\Sigma \Rightarrow \psi$ is in Skolem Normal Form if $\Sigma$ is in Skolem Normal Form, $\psi$ is in Prenex
Form, and $\psi$ contains no universal quantifiers. Herbrand's Theorem states that a sequent $\Sigma \Rightarrow \psi$ in Skolem Normal Form is valid ($\Sigma \models \psi$) if and only if every Herbrand model that satisfies $\Sigma$ also satisfies $\psi$. The Skolem Normal Transform $\Sigma \Rightarrow \psi$ of a sequent $\Sigma' \Rightarrow \psi'$ is defined by a Skolemization procedure, which rewrites $\Sigma'$ and $\psi'$ into Prenex Form, removes existential quantifiers from $\Sigma'$, and removes universal quantifiers from $\psi'$. The full procedure can be found in most texts on theorem proving, first-order logic, or foundations of artificial intelligence, for example [27; 9]. In the simplest case, once $\Sigma'$ and $\psi'$ are written in Prenex Form, all existential quantifiers precede the universal quantifiers in each sentence in $\Sigma'$, and all universal quantifiers precede the existential quantifiers in $\psi'$. In this case, Skolemization replaces existentially-quantified variables in $\Sigma'$ and universally-quantified variables in $\psi'$ by distinct constants, called Skolem constants. In general, a sequent $\Sigma' \Rightarrow \psi'$ is valid ($\Sigma' \models \psi'$) if and only if its Skolem Normal Transform is valid ($\Sigma \models \psi$).

2.3 The General Instantiation Ordering

2.3.1 Variable Abstraction

We begin this section by defining the variable abstractions of constrained formulas. This definition is central to the definition of the instantiation ordering on constrained formulas. It relies on the following notation, which is also used throughout the rest of the chapter. Let $\phi$ be any formula with $n$ top-level term occurrences, that is, $n$ occurrences of terms as arguments to predicates. Number the occurrences from 1 to $n$ in left-to-right order, as they appear in $\phi$. Let $t_i$ be the $i$th top-level term occurrence of $\phi$, for all $1 \leq i \leq n$. Then we also use $\phi[t_1, \ldots, t_n]$ to denote $\phi$. Subsequently we use $\phi[t'_1, \ldots, t'_n]$ to denote the formula that results from replacing the $i$th term occurrence in $\phi[t_1, \ldots, t_n]$ with $t'_i$ for all $1 \leq i \leq n$. For example, if $\phi[f(a), f(a), g(c, x)]$ denotes the formula $p(f(a)) \land q(f(a), g(c, x))$ then $\phi[g(a, b), c, f(x)]$ denotes $p(g(a, b)) \land q(c, f(x))$.

\footnote{Herbrand's Theorem has other, equivalent statements, such as, “A theory in Skolem Normal Form has a countermodel (a falsifying model) if and only if it has a Herbrand countermodel.”}
Definition 2 (Variable-Abstract Form) A constrained formula \( \phi[t_1, \ldots, t_n]/C \) is in variable-abstract form if and only if \( t_1, \ldots, t_n \) are distinct variables.

Definition 3 (Closure-equivalence) Two constrained formulas \( \phi_1/C_1 \) and \( \phi_2/C_2 \) are closure-equivalent with respect to a constraint theory \( \Sigma \) if and only if the following are true: \( \forall(\phi_1/C_1) \) and \( \forall(\phi_2/C_2) \) are \( \Sigma \)-equivalent, and \( \exists(\phi_1/C_1) \) and \( \exists(\phi_2/C_2) \) are \( \Sigma \)-equivalent. We say that \( \phi_1/C_1 \) and \( \phi_2/C_2 \) are closure-equivalent (without regard for the choice of \( \Sigma \)) if and only if: \( \forall(\phi_1/C_1) \) and \( \forall(\phi_2/C_2) \) are logically equivalent, and \( \exists(\phi_1/C_1) \) and \( \exists(\phi_2/C_2) \) are logically equivalent.

For example,

\[
p(x)/\text{HUMAN}(x) \land \text{MALE}(x) \land \text{ADULT}(x)
\]

and

\[
p(x)/\text{MAN}(x)
\]

are closure-equivalent with respect to \( \Sigma \) if and only if

\[
\Sigma \models \forall x ((\text{MAN}(x)) \leftrightarrow (\text{HUMAN}(x) \land \text{MALE}(x) \land \text{ADULT}(x)))
\]

As another example,

\[
p(x)/x = f(a)
\]

and

\[
p(f(y))/y = a
\]

are closure-equivalent, regardless of the choice of \( \Sigma \). Clearly, any two constrained formulas that are closure equivalent are in fact closure equivalent with respect to any constraint theory \( \Sigma \).
Definition 4 (Variable Abstractions) Let \( \phi/C \) be any constrained formula, and let \( \Sigma \) be any constraint theory. Then \( \phi'/C' \) is a (\( \Sigma \)-)variable abstraction of \( \phi/C \) if and only if: 

- \( \phi'/C' \) is in variable-abstract form, 
- \( \phi \) is an instance of \( \phi' \), and 
- \( \phi'/C' \) is closure-equivalent to \( \phi/C \) (with respect to \( \Sigma \)).

For example,

\[
p(x, y)/x = f(z) \land y = g(a, z)
\]

is a variable abstraction of

\[
p(f(z), g(a, z))
\]

Note that any two variable abstractions of a given constrained formula are themselves closure-equivalent, and are variable abstractions of one another, and of themselves. We now show that every constrained formula has a variable abstraction, and we present an Abstraction Algorithm that computes a variable abstraction for any constrained formula. The algorithm simply rewrites any constrained formula \( \phi[t_1,\ldots,t_n]/C \) to \( \phi[x_1,\ldots,x_n]/x_1 = t_1 \land \ldots \land x_n = t_n \land C \), where \( x_1,\ldots,x_n \) are distinct variables that do not appear in \( t_1,\ldots,t_n, C \). In order to prove easily the correctness of the algorithm (that the resulting constrained formula is a variable abstraction of the original one), we describe the Abstraction Algorithm as the repeated application of a simpler transformation. The following lemma motivates this simpler transformation and is useful elsewhere as well.

Lemma 5 Let \( \phi \) be any quantifier-free formula, and let \( t \) be any term in which the variable \( x \) does not occur. Then the following formulas are logically equivalent, that is, one formula is true in a given model and with a given value assignment if and only if all three are.

1. \( \phi \{ x \rightarrow t \} \)

2. \( \forall x \ (x = t \rightarrow \phi) \)
3. \( \exists x \ (x = t \land \phi) \).

\textbf{Proof:}

\([2 \Rightarrow 1]: \forall x \ (x = t \rightarrow \phi) \models t = t \rightarrow \phi \{x \mapsto t\}, \) by the rule of universal instantiation. Because \( t = t \) is trivially true, \( t = t \rightarrow \phi \{x \mapsto t\} \models \phi \{x \mapsto t\}. \)

\([1 \Rightarrow 3]: \phi \{x \mapsto t\} \models t = t \land \phi \{x \mapsto t\}, \) since \( t = t \) is trivially true. \( t = t \land \phi \{x \mapsto t\} \models \exists x \ (x = t \land \phi), \) by the rule of existential generalization.

\([3 \Rightarrow 2]: \) We assume that an arbitrary model \( M \) and value assignment \( g \) satisfy \( \exists x \ (x = t \land \phi) \), and we show that \( M \) and \( g \) also satisfy \( \forall x \ (x = t \rightarrow \phi) \). Since \( M \) and \( g \) satisfy \( \exists x \ (x = t \land \phi) \), there exists some individual \( d \) in the domain of \( M \) such that \( [x = t \land \phi]^{M,g[x \mapsto d]} = \text{True}. \) Hence, \( [x = t]^{M,g[x \mapsto d]} = \text{True}, \) and \( [\phi]^{M,g[x \mapsto d]} = \text{True}. \) Furthermore, by the definition of equality, \( d \) is the unique individual such that \( [x = t]^{M,g[x \mapsto d]} = \text{True}. \) Therefore, for any domain individual \( d' \), if \( [x = t]^{M,g[x \mapsto d']} = \text{True} \) then \( d' = d \), so \( [\phi]^{M,g[x \mapsto d']} = \text{True} \) as well. It follows, then, from the definition of universal quantification that \( M \) satisfies \( \forall x \ (x = t \rightarrow \phi) \). \qed

The \( i \)-th \textbf{Abstraction-Transform}, defined below, is the primary component of the Abstraction Algorithm.

\textbf{Definition 6 (i-th Abstraction-Transform)} Let \( \phi[t_1, \ldots, t_n]/C \) be a constrained formula, and let \( 1 \leq i \leq n \). Then the \( i \)-th Abstraction-Transform of \( \phi[t_1, \ldots, t_n]/C \) is

\[ \phi[t_1, \ldots, t_{i-1}, x_i, t_{i+1}, \ldots, t_n]/x_i = t_i \land C \]

where \( x_i \) is an arbitrary variable not appearing in \( t_1, \ldots, t_n \), or \( C \).

Obviously, the head \( \phi \) of a constrained formula \( \phi/C \) is an instance of the head of the formula’s \( i \)-th Abstraction-Transform. We now show that the \( i \)-th Abstraction Transform of \( \phi/C \) is closure-equivalent to \( \phi/C \).

\textbf{Lemma 7 (Correctness of the i-th Abstraction-Transform)} Any constrained formula \( \phi[t_1, \ldots, t_n]/C \) is closure-equivalent to its \( i \)-th Abstraction-Transform, for any \( 1 \leq i \leq n \).
Proof: Let \( \phi[t_1, \ldots, t_n]/C \) be a constrained formula, let \( 1 \leq i \leq n \), and let the \( i \)-th Abstraction-Transform of \( \phi[t_1, \ldots, t_i]/C \) be \( \phi[t_1, \ldots, t_{i-1}, x_i, t_{i+1}, \ldots, t_n]/x_i = t_i \land C \). We wish to show that (1) \( \nabla(x_i = t_i \land C) \to \phi[t_1, \ldots, t_{i-1}, x_i, t_{i+1}, \ldots, t_n]) \), which can be rewritten as \( \nabla(x_i = t_i \to (C \to \phi[t_1, \ldots, t_{i-1}, x_i, t_{i+1}, \ldots, t_n])) \), is logically equivalent to \( \nabla(C \to \phi[t_1, \ldots, t_n]) \), and (2) \( \exists(x_i = t_i \land (C \land \phi[t_1, \ldots, t_{i-1}, x_i, t_{i+1}, \ldots, t_n])) \) is logically equivalent to \( \exists(C \land \phi[t_1, \ldots, t_n]) \). To see that (1) holds, let \( \phi' \) be \( C \to \phi[t_1, \ldots, t_{i-1}, x_i, t_{i+1}, \ldots, t_n]. \) Because \( x_i \) does not appear in \( C \) or \( t_1, \ldots, t_n \), we know that \( x_i \) appears only once in \( \phi' \). Therefore, we may rewrite \( C \to \phi[t_1, \ldots, t_n] \) as \( \phi'[x_i \mapsto t_i]. \) Then (1) may be restated as: \( \nabla \phi'[x_i \mapsto t_i] \) is logically equivalent to \( \nabla(x_i = t_i \to \phi') \), where \( x_i \) does not occur in \( t_i \). Then Lemma 5 proves (1). Similarly, to see that (2) holds, let \( \phi' \) be \( C \land \phi[t_1, \ldots, t_{i-1}, x_i, t_{i+1}, \ldots, t_n]. \) By the same reasoning used for (1), we may rewrite \( C \land \phi[t_1, \ldots, t_n] \) as \( \phi'[x_i \mapsto t_i]. \) Then (2) may be restated as: \( \exists \phi'[x_i \mapsto t_i] \) is logically equivalent to \( \exists(x_i = t_i \land \phi') \), where \( x_i \) does not occur in \( t_i \). Then Lemma 5 proves (2).

\[ \square \]

The Abstraction Algorithm, to be applied to any constrained formula \( \phi[t_1, \ldots, t_n]/C \), may be defined as follows: beginning with \( \phi[t_1, \ldots, t_n]/C \), for \( i = 1 \) to \( n \) replace the current constrained formula with its \( i \)-th Abstraction-Transform.

**Theorem 8 (Abstraction Algorithm Correctness)** The Abstraction Algorithm can be implemented such that it terminates in time linear in the size of the input,\(^3\) and it returns a variable abstraction of the input.

**Proof:** The loop of the algorithm is repeated \( n \) times, where \( n \) is the number of top-level term occurrences in the head of the input constrained formula. It is straightforward to verify that the body of the loop on iteration \( i \)—computing the \( i \)-th Abstraction-Transform—can be performed in time linear in the size of \( t_i \). Thus the algorithm can be implemented to take time linear in the size of the head of the input constrained formula. The resulting constrained formula is in variable-abstract form, because its head is

\(^3\)We take the size of any constrained formula to be the sum of the size of the head and the size of the constraint, each of which is an ordinary logical formula.
of the form $\phi[x_1, \ldots, x_n]$ where $x_1, \ldots, x_n$ are distinct variables. The resulting constrained formula is closure-equivalent to the original constrained formula because, according to Lemma 7, each iteration preserves closure-equivalence. The head of the original constrained formula is an instance of the head of the resulting constrained formula because each iteration replaces a term in the head with a new variable.

Each choice of names for the new variables $x_1, \ldots, x_n$ in the Abstraction Algorithm yields a different variable abstraction, but all these variable abstractions are the same modulo variable renaming. Thus we have the following observation.

**Observation 1** Any two constrained formulas $\phi_1/C_1$ and $\phi_2/C_2$ have variable abstractions $\phi/C_1'$ and $\phi/C_2'$ with the same head if and only if $\phi_1$ and $\phi_2$ are both instances of some formula $\phi[x_1, \ldots, x_n]$. (In this case, we also say that $\phi_1$ and $\phi_2$ agree on structure and predicates.) As a special case, note that any two constrained literals $\phi_1/C_1$ and $\phi_2/C_2$ have variable abstractions $\phi/C_1'$ and $\phi/C_2'$ with the same head if and only if $\phi_1$ and $\phi_2$ have the same sign (both are atoms or both are negated atoms) and predicate.

As an example of variable abstraction, consider these two constrained formulas:

$$controls(\text{political-party}(\text{spouse(arnold)}), x, y)/\text{LEGISLATURE}(x) \land (y = 1993) \land (\forall w \text{ US-STATE}(w) \rightarrow (\exists z \text{ REPRESENTS}(z, w, x)))$$

(2.1)

$$controls(\text{democratic-party}, x, 1993)/(x = \text{us-house-of-representatives})$$

(2.2)

Constrained formulas (2.1) and (2.2) have variable abstractions (2.3) and (2.4), respectively, as would be produced by the Abstraction Algorithm for a particular choice of names for new variables. Note that formulas (2.3) and (2.4) have the same head.

$$controls(x_1, x_2, x_3) / (x_1 = \text{political-party}(\text{spouse(arnold)})) \land (x_2 = x) \land (x_3 = y) \land \text{LEGISLATURE}(x) \land (y = 1993) \land$$

(2.3)
\[ (\forall w \text{ US-STATE}(w) \rightarrow (\exists z \text{ REPRESENTS}(z, w, x))) \]  

(2.3)

\[ \text{controls}(x_1, x_2, x_3) \quad / \quad (x_1 = \text{democratic-party}) \land (x_2 = x) \land (x_3 = 1993) \land \]  

(\[ x = \text{us-house-of-representatives} \]) \]  

(2.4)

It is worth noting that any variable abstraction of a constrained formula is \( \Sigma \)-admissible if and only if the original constrained formula is itself \( \Sigma \)-admissible.

### 2.3.2 The Ordering

The ordinary instantiation ordering on ordinary formulas (not just ordinary atomic formulas) is particularly useful for three reasons. First, if \( \phi_1 \geq \phi_2 \) then the set of ground instances of \( \phi_1 \) is a superset of the set of ground instances of \( \phi_2 \). Second, if \( \phi_1 \geq \phi_2 \) then \( \forall \phi_1 \models \forall \phi_2 \) and \( \exists \phi_2 \models \exists \phi_1 \). Third, for atomic formulas (atoms), this second statement is an if and only if, that is, the following three statements are equivalent: (1) \( \phi_1 \geq \phi_2 \), (2) \( \forall \phi_1 \models \forall \phi_2 \), and (3) \( \exists \phi_2 \models \exists \phi_1 \). We seek an instantiation ordering for constrained formulas that has analogous properties, where we replace entailment with \( \Sigma \)-entailment and ground instances with ground “\( \Sigma \)-instances”, or ground instances under the new ordering.

**Definition 9 (\( \Sigma \)-more general)** Let \( \Sigma \) be a constraint theory. Given constrained formulas \( \phi_1/C_1 \) and \( \phi_2/C_2 \), we say that \( \phi_1/C_1 \) is \( \Sigma \)-more general than \( \phi_2/C_2 \) (written \( \phi_1/C_1 \geq \Sigma \phi_2/C_2 \)) if and only if either: (1) \( \phi_2/C_2 \) is \( \Sigma \)-inadmissible, or (2) \( \phi_1/C_1 \) and \( \phi_2/C_2 \) have variable abstractions \( \phi/C_1' \) and \( \phi/C_2' \), respectively, such that \( \Sigma \models \forall((\exists z C_2') \rightarrow (\exists y' C_1')) \), where \( y' \) denotes the free variables in \( C_1' \) but not \( \phi \) and \( z \) denotes the free variables in \( C_2' \) but not \( \phi \).

For one example of the \( \Sigma \)-more general ordering, let \( \Sigma_3 \) be the constraint theory that results from adding to \( \Sigma_2 \), given earlier, information about U. S. states and their congressmen, such as the following.
Let $C_1$ denote the constraint in constrained formula (2.3), given earlier, and let $C_2$ denote the constraint in constrained formula (2.4). Then constrained formula (2.1) is $\Sigma_3$-more general than constrained formula (2.2) because: formulas (2.1) and (2.2) have variable abstractions (2.3) and (2.4), respectively, with the same head, and

$$\Sigma_3 \models \forall x_1 \forall x_2 \forall x_3 ((\exists x \ C_2) \rightarrow (\exists x \forall y \exists z \ C_1))$$

For another example, let $\Sigma_4$ be $\{\forall x \forall y \ (f(x, y) = f(y, x), (c = a) \lor (c = b)\}$. The atoms $p(f(a, b))$ and $p(f(c, x))$ have variable abstractions

$$p(z)/z = f(a, b)$$

and

$$p(z)/z = f(c, x)$$

respectively. Because $\Sigma_4 \models \forall (z = f(a, b)) \rightarrow (\exists x \ z = f(c, x))$, we have that $p(f(c, x))$ is $\Sigma_4$-more general than $p(f(a, b))$.

These two examples rely on condition (2) of the definition of $\Sigma$-more general. For an example that uses condition (1), consider a pair of constrained formulas whose heads have different logical structures.

$$\text{loves}(x, \text{mom}(x)) / \text{ELEPHANT}(x) \quad (2.5)$$

$$\text{eats}(x, \text{peanuts}) \land \text{lives-at}(x, \text{zoo}) / \text{ELEPHANT}(x) \land \neg \text{ELEPHANT}(\text{mom}(x)) \quad (2.6)$$
Because the heads of formulas (2.5) and (2.6) have different logical structures, there exist no variable abstractions of these formulas with the same head. Hence the only way formula (2.5) can be $\Sigma$-more general than formula (2.6) is if formula (2.6) is $\Sigma$-inadmissible. Formula (2.6) is in fact $\Sigma_1$-inadmissible, since $\Sigma_1$ entails that the mother of any elephant is an elephant; thus formula (2.5) is $\Sigma_1$-more general then formula (2.6). On the other hand, formula (2.6) is not $\Sigma_2$-inadmissible, so formula (2.5) is not $\Sigma_2$-more general than formula (2.6).

Like the instantiation ordering for ordinary formulas that is based on substitutions, the $\geq_\Sigma$ ordering on constrained formulas is not a partial ordering but a quasi-ordering, or preorder, for any choice of $\Sigma$.

**Theorem 10** Let $\Sigma$ be any constraint theory. The $\Sigma$-more general relation is a quasi-ordering, that is, for all constrained formulas $\phi_1/C_1$, $\phi_2/C_2$, and $\phi_3/C_3$:

1. $\phi_1/C_1 \geq_\Sigma \phi_1/C_1$

2. $\phi_1/C_1 \geq_\Sigma \phi_2/C_2$ and $\phi_2/C_2 \geq_\Sigma \phi_3/C_3$ implies $\phi_1/C_1 \geq_\Sigma \phi_3/C_3$

**Proof:**

**Reflexivity (1):** The result holds by definition if $\phi_1/C_1$ is a $\Sigma$-inadmissible constrained formula. Assume $\phi_1/C_1$ is a $\Sigma$-admissible constrained formula. Then it has a $\Sigma$-admissible variable abstraction $\phi/C$. We wish to determine whether $\phi/C \geq_\Sigma \phi/C$. Let $\bar{y}$ be the variables in $C$ but not in $\phi$. For any $\Sigma$, obviously $\Sigma \models \forall((\exists \bar{y} C) \rightarrow (\exists \bar{y} C))$, so $\phi/C \geq_\Sigma \phi/C$.

**Transitivity (2):** The result is obvious if $\phi_3/C_3$ is a $\Sigma$-inadmissible constrained formula. Otherwise, all three constrained formulas must be $\Sigma$-admissible. It follows (since $\phi_1/C_1 \geq_\Sigma \phi_2/C_2$ and $\phi_2/C_2 \geq_\Sigma \phi_3/C_3$) that $\phi_1/C_1$, $\phi_2/C_2$, and $\phi_3/C_3$ must have variable abstractions with the same head: $\phi/C_1'$, $\phi/C_2'$, and $\phi/C_3'$, respectively, where $\phi/C_1' \geq_\Sigma \phi/C_2'$ and $\phi/C_2' \geq_\Sigma \phi/C_3'$. Let $\bar{x}_1$ be the variables in $C_1'$ but not $\phi$, $\bar{x}_2$ the variables in $C_2'$ but not $\phi$, and $\bar{x}_3$ the variables in $C_3'$ but not $\phi$. Then $\Sigma \models \forall((\exists \bar{x}_1 C_1') \rightarrow (\exists \bar{x}_2 C_2'))$, and $\Sigma \models \forall((\exists \bar{x}_2 C_2') \rightarrow (\exists \bar{x}_3 C_3'))$. Therefore $\Sigma \models \forall((\exists \bar{x}_1 C_1') \rightarrow (\exists \bar{x}_3 C_3'))$, so $\phi_1/C_1 \geq_\Sigma \phi_3/C_3$. \qed
Because $\geq_{\Sigma}$ is a quasi-ordering, if $\phi_1/C_1 \geq_{\Sigma} \phi_2/C_2$ then every ground instance of $\phi_2/C_2$ is also a ground instance of $\phi_1/C_1$. Thus the $\geq_{\Sigma}$ ordering is analogous to the $\geq$ ordering on ordinary atoms in at least one of the three ways we wanted it to be. What about the other two? Theorems 11 and 14, which follow, reveal that the ordering $\geq_{\Sigma}$ for constrained formulas is analogous to the ordering $\geq$ for ordinary formulas in these ways as well.

**Theorem 11** Let $\phi_1/C_1$ and $\phi_2/C_2$ be constrained formulas, and let $\Sigma$ be a constraint theory. If $\phi_1/C_1 \geq_{\Sigma} \phi_2/C_2$ then $\nabla(\phi_1/C_1) \models_{\Sigma} \nabla(\phi_2/C_2)$ and $\exists(\phi_2/C_2) \models_{\Sigma} \exists(\phi_1/C_1)$.

**Proof:** If either $\phi_1/C_1$ or $\phi_2/C_2$ is $\Sigma$-inadmissible, then the result is obvious: if $\phi_2/C_2$ is $\Sigma$-inadmissible, then all three relationships hold trivially, and if $\phi_1/C_1$ is $\Sigma$-inadmissible while $\phi_2/C_2$ is $\Sigma$-admissible, then none of the three relationships holds. Otherwise, if $\phi_1/C_1 \geq_{\Sigma} \phi_2/C_2$ then $\phi_1/C_1$ and $\phi_2/C_2$ can be written as $\Sigma$-admissible constrained formulas with the same head, $\phi_1/C_1$ and $\phi_2/C_2$. Let $\overline{y}$ denote the variables in $C_1'$ but not $\phi$, and let $\overline{z}$ denote the variables in $C_2'$ but not $\phi$. By definition, $\phi_1/C_1 \geq_{\Sigma} \phi_2/C_2$ if and only if $\Sigma \models \nabla((\exists \overline{z} C_1') \rightarrow (\exists \overline{y} C_1'))$.

We first show that if $\Sigma \models \nabla((\exists \overline{z} C_1') \rightarrow (\exists \overline{y} C_1'))$ then $\nabla(C_1' \rightarrow \phi) \models_{\Sigma} \nabla(C_2' \rightarrow \phi).$ We prove the contrapositive of our goal by contradiction. Assume that some $\Sigma$-model, $M$, satisfies $\nabla(C_1' \rightarrow \phi)$ and falsifies $\nabla(C_2' \rightarrow \phi).$ Since $M$ falsifies $\nabla(C_2' \rightarrow \phi)$, there exists a value assignment, $g$, such that $[C_2']_{M,g}^{\Sigma} = \text{True}$ and $[\phi]_{M,g}^{\Sigma} = \text{False}.$ Since $[C_2']_{M,g}^{\Sigma} = \text{True}$, we know $[\exists \overline{z} C_2']_{M,g}^{\Sigma} = \text{True}.$ Since $\Sigma \models \nabla((\exists \overline{z} C_2') \rightarrow (\exists \overline{y} C_1'))$, and $M$ is a $\Sigma$-model, we have $[\exists \overline{y} C_1']_{M,g}^{\Sigma} = \text{True}.$ Then there exists a value assignment $g'$ that agrees with $g$ on all variables except those in $\overline{y}$, such that $[C_1']_{M,g'}^{\Sigma} = \text{True}.$ But because (1) $[\phi]_{M,g}^{\Sigma} = \text{False}$, (2) $g'$ agrees with $g$ on all variables except those in $\overline{y}$, and (3) the variables in $\overline{y}$ do not appear in $\phi$, we know that $[\phi]_{M,g'}^{\Sigma} = \text{False}.$ Then since $[C_1']_{M,g'}^{\Sigma} = \text{True}$ and $[\phi]_{M,g'}^{\Sigma} = \text{False},$ $M$ does not satisfy $\nabla(C_1' \rightarrow \phi)$, contradicting to our original assumption.

We next show that if $\Sigma \models \nabla((\exists \overline{z} C_1') \rightarrow (\exists \overline{y} C_1'))$ then $\exists(C_2' \land \phi) \models_{\Sigma} \exists(C_1' \land \phi).$ If $M$ satisfies $\exists(C_2' \land \phi)$, there exists a value assignment, $g$, such that $[C_2']_{M,g}^{\Sigma} = \text{True}$ and $[\phi]_{M,g}^{\Sigma} = \text{True}.$ Since $[C_2']_{M,g}^{\Sigma} = \text{True},$ we know $[\exists \overline{z} C_1']_{M,g}^{\Sigma} = \text{True}.$ Since $M$ is a
\(\Sigma\)-model, \(\left[\exists z' C_2' \rightarrow (\exists y' C_1')\right]^{M, g} = \text{True}\), so \(\left[\exists y' C_1'\right]^{M, g} = \text{True}\). Then for some value assignment \(g'\) that agrees with \(g\) on all variables except, possibly, those of \(y'\), we have \(\left[C_1'\right]^{M, g'} = \text{True}\). Because the variables in \(y'\) do not appear in \(\phi\), and \(\left[\phi\right]^{M, g} = \text{True}\), we have \(\left[\phi\right]^{M, g'} = \text{True}\). Therefore, \(M\) satisfies \(\exists(C_1' \land \phi)\). \(\square\)

It remains to show (Theorem 14) that for constrained atoms \(\phi_1/C_1\) and \(\phi_2/C_2\) the following three statements are equivalent:

1. \(\phi_1 \geq \phi_2\)
2. \(\neg\phi_1 \models \neg\phi_2\)
3. \(\exists\phi_2 \models \exists\phi_1\)

Theorem 14 actually shows this, more generally, for constrained literals. Lemma 12 and Lemma 13, below, are central to the proof of Theorem 14.

**Lemma 12** Let \(\phi_1/C_1\) and \(\phi_2/C_2\) be constrained literals, and let \(\Sigma\) be a constraint theory. Then \(\neg(\phi_1/C_1) \models_{\Sigma} \neg(\phi_2/C_2)\) if and only if \(\exists(\phi_2/C_2) \models_{\Sigma} \exists(\phi_1/C_1)\).

**Proof:** We have

\[
\neg(\phi_1 \rightarrow \phi_1) \models_{\Sigma} \neg(\phi_2 \rightarrow \phi_2)
\]

if, and only if, \(\neg(\phi_1 \rightarrow \neg\phi_1) \models_{\Sigma} \neg(\phi_2 \rightarrow \neg\phi_2)\) Renaming

if, and only if, \(\neg(\neg\phi_2 \rightarrow \neg\phi_2) \models_{\Sigma} \neg\neg(\phi_1 \rightarrow \neg\phi_1)\) Contraposition

if, and only if, \(\exists(\neg\phi_2 \land \phi_2) \models_{\Sigma} \exists(\neg\phi_1 \land \phi_1)\) Rewriting \(\square\)

**Lemma 13** Let \(\phi/C_1\) and \(\phi/C_2\) be constrained literals, and let \(\Sigma\) be a constraint theory. Let \(y'\) be the variables in \(C_1\) but not \(\phi\), and let \(z'\) be the variables in \(C_2\) but not \(\phi\). If \(\neg(\phi/C_1) \models_{\Sigma} \neg(\phi/C_2)\) then \(\Sigma \models \neg((\exists z' C_2) \rightarrow (\exists y' C_1))\).

**Proof:** We again prove the contrapositive. Note that \(\neg(\phi/C_1)\) is logically equivalent to \(\neg((\exists y' C_1) \rightarrow \phi)\), and \(\neg(\phi/C_2)\) is logically equivalent to \(\neg((\exists z' C_2) \rightarrow \phi)\). Assume that \(\Sigma \not\models \neg((\exists z' C_2) \rightarrow (\exists y' C_1))\); we build a \(\Sigma\)-model \(M'\) that satisfies \(\neg((\exists y' C_1) \rightarrow \phi)\)
and falsifies \( \neg((\exists \bar{z} \ C_2) \to \phi) \). Since \( \Sigma \models \neg((\exists \bar{z} \ C_2) \to (\exists \bar{y} \ C_1)) \), there exists a \( \Sigma \)-model \( M \) and a value assignment \( g \) such that \( [(\exists \bar{z} \ C_2)]^{M,g} = \text{True} \) and \( [(\exists \bar{y} \ C_1)]^{M,g} = \text{False} \). Let \( p \) be the predicate from which \( \phi \) is built; \( p \) must be an ordinary predicate, so it cannot appear in \( C_1 \), \( C_2 \), or \( \Sigma \). Therefore, there exists a \( \Sigma \)-model \( M' \) that agrees with \( M \) on every predicate except (possibly) \( p \), such that for any value assignment \( e \) : \( [\phi]^{M',e} = \text{True} \) if, and only if, \( [(\exists \bar{y} \ C_1)]^{M',e} = \text{True} \). Then \( [(\neg((\exists \bar{z} \ C_2) \to \phi))]^{M'} = \text{True} \). Since \( [(\exists \bar{y} \ C_1)]^{M',g} = \text{False} \), we know \( [\phi]^{M',g} = \text{False} \); since \( [(\exists \bar{z} \ C_2)]^{M',g} = \text{True} \), we have \( [(\exists \bar{z} \ C_2) \to \phi]^{M',g} = \text{False} \), so \( [(\neg((\exists \bar{z} \ C_2) \to \phi))]^{M'} = \text{False} \). \( \square \)

**Theorem 14 (\( \Sigma \)-Subsumption/\( \Sigma \)-Entailment Equivalence)** Let \( \phi_1/C_1 \) and \( \phi_2/C_2 \) be constrained literals, and let \( \Sigma \) be a constraint theory. Then the following three statements are equivalent.

1. \( \phi_1/C_1 \geq \Sigma \phi_2/C_2 \)
2. \( \neg(\phi_1/C_1) \models_{\Sigma} \neg(\phi_2/C_2) \)
3. \( \exists(\phi_2/C_2) \models_{\Sigma} \exists(\phi_1/C_1) \)

**Proof:** Lemma 12 asserts the equivalence of (2) and (3). Theorem 11 states that (1) implies (2) (and (1) implies (3), but only one of these is needed here). We now show that (2) implies (1).

If \( \phi_1 \) and \( \phi_2 \) have different signs or different predicates, then \( \neg(\phi_1/C_1) \models_{\Sigma} \neg(\phi_2/C_2) \) if and only if \( C_2 \) is not \( \Sigma \)-satisfiable (\( \phi_2/C_2 \) is \( \Sigma \)-inadmissible). But if \( \phi_1 \) and \( \phi_2 \) have different signs or different predicates, then by definition \( \phi_1/C_1 \geq \Sigma \phi_2/C_2 \) if and only if \( C_2 \) is not \( \Sigma \)-satisfiable. Therefore, we may restrict our attention to the case in which \( C_2 \) is \( \Sigma \)-satisfiable.

Since \( \phi_1 \) and \( \phi_2 \) have the same sign and predicate, \( \phi_1/C_1 \) and \( \phi_2/C_2 \) have variable abstractions \( \phi/C'_1 \) and \( \phi/C'_2 \), respectively. We wish to show that \( \Sigma \models \neg((\exists \bar{z} \ C'_2) \to (\exists \bar{y} \ C'_1)) \), where \( \bar{y} \) denotes the variables in \( C_1 \) but not \( \phi \), and \( \bar{z} \) denotes the variables in \( C_2 \) but not \( \phi \). Because \( \neg(\phi_1/C_1) \models_{\Sigma} \neg(\phi_2/C_2) \), and \( \phi/C'_1 \) and \( \phi/C'_2 \) are closure-equivalent
to \(\phi_1/C_1\) and \(\phi_2/C_2\), respectively, we know \(\overline{\forall}(\phi/C_1) \models_\Sigma \overline{\forall}(\phi/C_2)\). Then by Lemma 13, we have \(\Sigma \models \overline{\forall}(\exists \vec{z}' C'_2) \rightarrow (\exists \vec{y}' C'_1')\).

So we see that the ordering \(\geq_\Sigma\) for constrained formulas is analogous to the ordering \(\geq\) for ordinary formulas in exactly the ways we wanted it to be. But the ordering \(\geq\) is useful as well because we can determine efficiently whether one ordinary formula is more general than another. Can we determine efficiently whether one constrained formula is \(\Sigma\)-more general than another? Theorem 16, below, tells us that efficient determination of the consequences of \(\Sigma\) is necessary and sufficient for efficiently determining whether one constrained formula is \(\Sigma\)-more general than another. The proof of Theorem 16 uses the following lemma.\(^4\)

**Lemma 15** Let \(\Sigma\) be a constraint theory, and let \(\phi/C'_1\) and \(\phi/C'_2\) be \(\Sigma\)-variable abstractions of \(\phi_1/C_1\) and \(\phi_2/C_2\), respectively. If \(\phi_1/C_1 \geq_\Sigma \phi_2/C_2\) then \(\Sigma \models \overline{\forall}(\exists \vec{z}' C'_2) \rightarrow (\exists \vec{y}' C'_1')\), where \(\vec{y}\) denotes the variables in \(C'_1\) but not \(\phi\), and \(\vec{z}'\) denotes the variables in \(C'_2\) but not \(\phi\).

**Proof:** Since \(\phi_1/C_1\) and \(\phi_2/C_2\) have \(\Sigma\)-variable abstractions with the same head, \(\phi_1\) and \(\phi_2\) must have the same structure and predicates. Therefore, the heads of the variable abstractions of \(\phi_1\) and \(\phi_2\) are all variants (they all have the same structure and predicates, and the terms of each are distinct variables). Then since \(\phi_1/C_1 \geq_\Sigma \phi_2/C_2\), there exist variable abstractions \(\phi/C''_1\) and \(\phi/C''_2\) of \(\phi_1/C_1\) and \(\phi_2/C_2\), respectively, such that: where \(\vec{y}\) denotes the variables in \(C''_1\) but not \(\phi\) and \(\vec{z}'\) denotes the variables in \(C''_2\) but not \(\phi\), we have \(\Sigma \models \overline{\forall}(\exists \vec{z}' C''_2) \rightarrow (\exists \vec{y}' C''_1')\). But since \(\phi/C'_1\) and \(\phi/C''_1\) are both \(\Sigma\)-variable abstractions of \(\phi_1/C_1\), they are closure-equivalent with respect to \(\Sigma\), so \(\exists \vec{y}' C'_1\) is \(\Sigma\)-equivalent to \(\exists \vec{y}' C''_1\). By a similar argument \(\exists \vec{z}' C'_2\) is \(\Sigma\)-equivalent to \(\exists \vec{y}' C''_2\). Therefore, since \(\Sigma \models \overline{\forall}(\exists \vec{z}' C'_2) \rightarrow (\exists \vec{y}' C''_1')\), it follows that \(\Sigma \models \overline{\forall}(\exists \vec{z}' C'_2) \rightarrow (\exists \vec{y}' C'_1')\). \(\Box\)

The point of Lemma 15 is that to determine whether one constrained formula is \(\Sigma\)-more general than a second, we need only consider any one pair of variable abstractions.

\(^4\)Notice the subtle distinction between this lemma and Lemma 13. This lemma begins by rewriting two constrained formulas \((\phi_1/C_1\) and \(\phi_2/C_2)\) that might not have the same head to arbitrary \(\Sigma\)-variable abstractions \((\phi/C'_1\) and \(\phi/C'_2)\) with the same head.
(or even Σ-variable abstractions), with the same head, of the constrained formulas rather than all such pairs of variable abstractions.

**Theorem 16** Let $P(x)$ be any polynomial function of $x$. An algorithm exists that determines for any pair of constrained formulas $\phi_1/C_1$ and $\phi_2/C_2$ whether $\phi_1/C_1 \succeq_\Sigma \phi_2/C_2$, for a constraint theory $\Sigma$, and does so in time $O(P(|\phi_1/C_1| + |\phi_2/C_2|))$ if, and only if: an algorithm exists for determining, for any sentence $\psi$, whether $\Sigma \models \psi$, and does so in time $O(P(|\psi|))$.

**Proof:**

**Sufficiency:** Recall that two constrained formulas have variable abstractions with the same head if and only if the heads of the constrained formulas agree on structure and predicates. Thus if two constrained formulas have heads with different structure or predicates, neither is $\Sigma$-more general than the other. Otherwise, by Lemma 15 we can use any pair of variable abstractions with the same head to determine if one constrained formula is $\Sigma$-more general than the other. We have seen that, based on Lemma 5, we can compute variable abstractions $\phi/C_4$ and $\phi/C_2$ in time linear in the sizes of the constrained formulas (and so we obtain variable abstractions whose sizes are linear in the sizes of the original constrained formulas). All that remains is to check whether $\Sigma \models \psi$ where $\psi$ is $\forall(\exists \bar{z} C_2) \rightarrow (\exists \bar{z} C_1)$; $|\psi|$ is linear in the sizes of the original constrained formulas, so the query is answered in time $O(P(|\phi_1/C_1| + |\phi_2/C_2|))$.

**Necessity:** Assume an algorithm $A$ can determine whether $\phi_1/C_1 \succeq_\Sigma \phi_2/C_2$, for a constraint theory $\Sigma$, and does so in time $O(P(|\phi_1/C_1| + |\phi_2/C_2|))$. Let $\psi$ be any logical sentence. Let $x$ be a variable that does not appear in $\psi$, and let $p$ be a predicate symbol that does not appear in $\Sigma$ or $\psi$. Take $\psi$ to be a constraint, and ask $A$ whether $p(x)/\psi$ is $\Sigma$-more general than $p(x)$; if so, then $\Sigma \models \psi$, and if not, then $\Sigma \not\models \psi$. Because $p(x)/\psi$ and $p(x)$ are together just a constant size larger than $\psi$, for any $\psi$, the determination is made in time $O(|\psi|)$.

Of course, in general no constraint theory $\Sigma$ exists such that an algorithm exists for determining, for any sentence $\psi$, whether $\Sigma \models \psi$, let alone an algorithm that does so.
in polynomial time. (The only such theory is an inconsistent one, and we do not allow constraint theories to be inconsistent.) The form of \( \psi \) must be restricted, as well as the form of \( \Sigma \), for such an algorithm to exist. Restricting the form of \( \psi \) corresponds to restricting the form of the constraints on constrained formulas. The next chapter and the remainder of the dissertation consider constrained formulas with various restrictions on the form of the constraints.

## 2.4 Initial Models and Strong Compactness

This section identifies an additional semantic property of constrained atoms, which we call strong compactness (a term applied to ordinary literals by Lassez, Maher, and Marriott [39]), that is central to some applications of anti-unification in constraint logic. Strong compactness says that for constrained atoms \( \phi_1/C_1, \ldots, \phi_n/C_n \), and \( \phi/C \), and a constraint theory \( \Sigma: \nabla(\phi_1/C_1) \land \ldots \land \nabla(\phi_n/C_n) \models_{\Sigma} \nabla(\phi/C) \) if and only if for some \( 1 \leq i \leq n \) we have \( \phi_i/C_i \geq_{\Sigma} \phi/C \). But, as we might suspect, in general this property does not hold. For example,\(^5\) let \( \Sigma_A \) be the constraint theory \( \{ \text{BOY}(ralph) \lor \text{GIRL}(ralph) \} \). Then

\[
\nabla(p(x)/\text{BOY}(x)) \land \nabla(p(x)/\text{GIRL}(x)) \models_{\Sigma} p(ralph)
\]

yet neither \( p(x)/\text{BOY}(x) \) nor \( p(x)/\text{GIRL}(x) \) is \( \Sigma_A \)-more general than \( p(\text{ralph}) \). For a related example, let \( \Sigma_B \) be the empty constraint theory. Then

\[
\nabla(p(x)/\text{BOY}(x)) \land \nabla(p(x)/\text{GIRL}(x)) \models_{\Sigma} \nabla(p(x)/\text{BOY}(x) \lor \text{GIRL}(x))
\]

yet neither \( p(x)/\text{BOY}(x) \) nor \( p(x)/\text{GIRL}(x) \) is \( \Sigma_A \)-more general than \( p(x)/\text{BOY}(x) \lor \text{GIRL}(x) \).

In both examples, the failure of strong compactness is caused by disjunctive information. In the first example the disjunction is in the theory, while in the second it appears in a constraint. This observation leads us to a pair of restrictions, one on the form of con-

\(^5\)This example is motivated by a counterexample to what seems, on the surface, to be a very different proposition—the existence of a fully-general Herbrand Theorem for sorted logic without equality [24].
straints and one on the form of $\Sigma$, that guarantees strong compactness. Specifically, any constraint must be a conjunction of atoms built from constraint predicates, and $\Sigma$ must be a Skolem Normal Form theory that has an initial model. For example, both constraint theories $\Sigma_1$ and $\Sigma_2$ are in Skolem Normal Form and have initial models. We now provide several definitions needed for the strong compactness result, which are also used in the chapters that follow. The definitions of homomorphism and isomorphism between models are standard. The definition of initial model, taking into account the truth values of predicates, is taken from Goguen and Meseguer [28]. The definition of Herbrand-quotient model comes from the standard definition of quotient model, and we simply draw attention to the Herbrand Universe as the domain over which the quotient model is taken. Since Herbrand models are also called term algebras, what we call Herbrand-quotient models are also called quotients of term algebras.

**Definition 17 (Homomorphism between models)** Let $M$ and $M'$ be models. Then a function $h : D_M \rightarrow D_{M'}$ is a homomorphism from $M$ to $M'$ if and only if:

- for every $n$-ary function $f$ $(n \geq 0)$, $h([f]^M(d_1, \ldots, d_n)) = [f]^{M'}(h(d_1), \ldots, h(d_n))$
- for every $n$-ary predicate $p$ $(n \geq 0)$, if $[p]^M(d_1, \ldots, d_n) = True$ then we also have $[p]^{M'}(h(d_1), \ldots, h(d_n)) = True$

where $\langle d_1, \ldots, d_n \rangle$ is any $n$-tuple of individuals in $D_M$ [28].

For example, consider an alphabet that has only one constant, one unary function symbol, and one binary predicate symbol. Let $M$ be a model such that: $D_M = \{1, 2, 3\}$; the constant is mapped to 1; the function symbol is mapped to a function $f$, for which $f(1) = 2$, $f(2) = 3$, and $f(3) = 1$; the predicate symbol is mapped to a binary relation $R$ that contains (is true of) only the tuples $\langle 1, 1 \rangle$, $\langle 2, 3 \rangle$, and $\langle 3, 2 \rangle$. Let $M'$ be a model

---

<sup>6</sup>Some authors define homomorphisms relative to an explicitly-stated alphabet—a particular set of function symbols and predicate symbols. This approach could be taken in defining and ordering Herbrand models as well, though typically it is not. We usually take the alphabet to be, implicitly, the alphabet of a particular theory under consideration, as is standard practice for Herbrand models. We occasionally identify the alphabet explicitly.
such that: $D_{M'} = \{4, 5, 6, 7\}$; the constant is mapped to 4; the unary function symbol is mapped to the function $g$, for which $g(4) = 7, g(5) = 7, g(6) = 4, \text{and } g(7) = 6$; and the predicate symbol is mapped to the relation $Q$, which contains all tuples except $\langle 6, 6 \rangle$ and $\langle 7, 7 \rangle$. Let $h(1) = 4, h(2) = 7, \text{and } h(3) = 6$. Then $h$ is a homomorphism from $M$ to $M'$.

**Definition 18 (Isomorphic Models)** Two models $M \text{ and } M'$ are isomorphic if and only if there exists a homomorphism $h$ from $M$ to $M'$ such that: $h$ has an inverse $h'$, and $h'$ is a homomorphism from $M'$ to $M$.

**Definition 19 (Initial Model)** Let $\Sigma$ be any theory of first-order logic with equality. A model $M$ is an initial model of $\Sigma$ if and only if: $M$ satisfies $\Sigma$, and there exists a unique homomorphism from $M$ to any other model that satisfies $\Sigma$ [28].

Let’s consider several sample constraint theories in Skolem Normal Form and determine whether they have initial models. First consider the very simple theory $\Sigma_A = \{BOY(ralphp)\}$. Any model $M$ that has only one individual $d$, that maps the constant $ralph$ to $d$, and that maps BOY to a unary relation that contains (is true of) $d$, is an initial model of $\Sigma_A$. To see this, consider any other model $M'$ that satisfies $\Sigma_A$. $M'$ must map $ralph$ to some individual $d'$, and it must map BOY to a unary relation containing $d'$. Then the homomorphism that maps $d$ in $D_M$ to $d'$ in $D_{M'}$ is the unique homomorphism from $M$ to $M'$. Are there other initial models of $\Sigma_A$ that do not “look like” $M$? There are none, as we shall see now. For a model $M'$ to differ from $M$ yet be based on the same alphabet and satisfy $\Sigma_A$, $D_{M'}$ must contain additional individuals (sometimes called junk) to which no ground terms ($ralph$ is the only ground term in the alphabet) are mapped. But there exist multiple homomorphisms from $M'$ to any other such model that satisfies $\Sigma_A$ (for example, to itself). In fact, initial models are unique up to isomorphism [29].

Now consider the theory $\Sigma_B = \{BOY(ralphp) \lor GIRL(ralphp)\}$. We shall see that $\Sigma_B$ has no initial model. Let $M_1$ be the model that results from extending the model, $M$, from the previous paragraph so that $M_1$ maps the predicate symbol GIRL to the empty unary relation. Clearly $M_1$ satisfies $\Sigma_B$. Let $M_2$ be the result of “reversing” the mappings of $M_1$ for the predicate symbols BOY and GIRL, so that BOY is mapped to the empty
unary relation and GIRL is mapped to the unary relation containing the individual $d$ (the only individual in $D_{M_1}$). Then $M_2$ also satisfies $\Sigma_B$. Now assume some model $M'$ is an initial model of $\Sigma_B$. Let $d'$ be the individual in $D_{M'}$ to which $M'$ maps ralph, and let $R_1$ and $R_2$ be the unary relations to which $M'$ maps BOY and GIRL, respectively. Unique homomorphism from $M'$ to $M_1$ and to $M_2$ must exist. The homomorphism from $M'$ to $M_1$ must map $d'$ to $d$, and the same is true of the homomorphism from $M'$ to $M_2$. Then neither $R_1$ nor $R_2$ can contain $d'$: (1) if $R_1$ contains $d'$, then since $M_2$ maps BOY to the empty relation, no homomorphism exists from $M'$ to $M_2$, and (2) if $R_2$ contains $d'$, then since $M_1$ maps GIRL to the empty relation, no homomorphism exists from $M'$ to $M_1$.

The preceding example illustrates how the requirement that an initial model exists rules out disjunctive theories. But might we not simply require that a theory contain no disjunction? This approach is insufficient because disjunction may be built using other logical connectives, or even using existential quantifiers and the primitive equality predicate. The restriction must rule out such “implicit” disjunctions as well.

We now consider an example of an initial model for a theory that involves the equality predicate and an alphabet from which infinitely many ground terms can be built. This example foreshadows the definition of Herbrand-quotient models, which follows. Let $\Sigma_C$ be

$$\{\text{BABY}(\text{ralph}), \forall x \forall y (f(x,y) = f(y,x))\}$$

There are infinitely many models of $\Sigma_C$. Let’s consider a “Herbrand-like” model $H$ of $\Sigma_A$ whose domain consists of sets of terms. Specifically, let $D_H$ consist of the equivalence classes of ground terms under the following equivalence relation: two terms are equivalent if one can be rewritten to the other by repeated application of the rewrite rule $f(t_1, t_2) \rightarrow f(t_2, t_1)$. (The rule applies for any terms $t_1$ and $t_2$.) Thus, for example, one equivalence class is the set whose elements are $f(f(\text{ralph}, \text{ralph}), \text{ralph})$ and $f(\text{ralph}, f(\text{ralph}, \text{ralph}))$, while another equivalence class is simply the set $\{\text{ralph}\}$. Let $H$ map the constant ralph to this latter equivalence class. In analogy to Herbrand models, let $H$ assign $f$ to a

33
binary function that maps a pair of equivalence classes containing the ground terms $t_1$ and $t_2$, respectively, to the equivalence class containing $f(t_1, t_2)$; it is straightforward to verify that such a function exists. Then $H$ maps any ground term $t$ to the equivalence class containing $t$. It is also straightforward to verify that any two ground terms $t_1$ and $t_2$ are in the same equivalence class if and only if $\Sigma_A \models t_1 = t_2$. Let $H$ map the predicate symbol BABY to a unary relation that contains (is true of) only the equivalence class \{ralph\}. $H$ is an initial model of $\Sigma$. To see this, consider any model $M$ that satisfies $\Sigma$. We show there is a unique homomorphism from $H$ to $M$. Let $h$ be a function that maps an equivalence class $\epsilon$ in $D_H$ to an individual $d$ in $D_M$ if and only if $M$ maps some ground term in $\epsilon$ to $d$. The function $h$ is well-defined: if $M$ maps two ground terms in an equivalence class $\epsilon \in D_H$ to two distinct individuals $d_1$ and $d_2$ in $D_M$, then $M$ falsifies the sentence $\forall x \forall y (f(x, y) = f(y, x))$ and so does not satisfy $\Sigma$. The function $h$ is a homomorphism from $H$ to $M$ because:

- $h([ralph]^H) = [ralph]^M$, and $h([f]^H(\epsilon_1, \epsilon_2)) = [f]^M(h(\epsilon_1), h(\epsilon_2))$ for any $\epsilon_1$ and $\epsilon_2$ in $D_H$\(^7\)

- $[BABY]^H(\epsilon_1) = \text{True}$ implies $[BABY]^M(h(\epsilon_1)) = \text{True}$: if this is not the case, then $M$ falsifies BABY(ralph), so $M$ does not satisfy $\Sigma$\(^8\)

where $\epsilon_1$ and $\epsilon_2$ are any individuals (equivalence classes) in $D_H$. It is easy to verify that $h$ is the unique homomorphism from $H$ to $M$: any function $h' : D_H \to D_M$ distinct from $h$ cannot satisfy condition (1) of a homomorphism from $H$ to $M$.

Notice that $\Sigma_A$ and $\Sigma_C$ in the examples above are very simple Horn clause theories ($\Sigma_C$ with equality). More generally, any Horn clause theory, with or without the equality predicate, has an initial model [28]. Nevertheless, we have already seen ($\Sigma_B$) that some

\(^7\)We have $[f]^M(h(\epsilon_1), h(\epsilon_2)) = d$ for some $d \in D_M$. Let $t_1$ and $t_2$ be arbitrary ground terms in $\epsilon_1$ and $\epsilon_2$, respectively. $M$ maps $t_1$ to some $d_1 \in D_M$ and $t_2$ to some $d_2 \in D_M$, so $h(\epsilon_1) = d_1$ and $h(\epsilon_2) = d_2$. $M$ assigns $f$ to some function that maps $\langle d_1, d_2 \rangle$ to $d$. Therefore $M$ maps the ground term $f(t_1, t_2)$ to $d$. Hence $h([f]^H(\epsilon_1, \epsilon_2)) = d$.

\(^8\)We need not consider the only other predicate involved—the interpreted equality predicate—since by definition the condition is satisfied for this predicate. This predicate has the same definition for every model and every domain: for any model and any two individuals $d_1$ and $d_2$ in the domain of that model, the equality predicate is true of $\langle d_1, d_2 \rangle$ if and only if $d_1$ and $d_2$ in fact are the same individual.
Skolem Normal Form theories do not have initial models. Notice also that, from the
definition of an initial model, if a theory $\Sigma$ has an initial model $M$, then $\Sigma$ entails a given
sentence if and only if $M$ satisfies that sentence.

Before presenting the strong compactness result, we present an additional definition
and two additional results related to the concept of initial models that are useful in the
following chapter.

**Definition 20 (Herbrand-Quotient Model)** A model $M$ is a Herbrand-quotient model if and only if:

- $D_M$ consists of disjoint sets (intuitively, equivalence classes) of ground terms
- for any constant $c$ we have $\llbracket c \rrbracket^M = c$ where $c \in D_M$ contains $c$
- for each $n$-ary function symbol $f$ and each $n$-tuple of ground terms $\langle t_1, \ldots, t_n \rangle$, we
  have $\llbracket f \rrbracket^M(\llbracket t_1 \rrbracket^M, \ldots, \llbracket t_n \rrbracket^M) = c$ where $c \in D_M$ contains $f(t_1, \ldots, t_n)$

For example, the initial model $H$ of $\Sigma_C$ is a Herbrand-quotient model. If a Herbrand-quotient model $M$ is an initial model of a theory $\Sigma$, then we say that $M$ is an initial Herbrand-quotient model of $\Sigma$.

**Lemma 21** Any Skolem Normal Form theory $\Sigma$ that has an initial model has an initial
Herbrand-quotient model.

**Proof:** Suppose $\Sigma$ has an initial model $M$. Let $H$ be a Herbrand-quotient model such
that two ground terms $t_1$ and $t_2$ are in the same equivalence class in $D_H$ if and only if
$\llbracket t_1 \rrbracket^M = \llbracket t_2 \rrbracket^M$. For any $n$-ary predicate symbol $p$ and any $n$-tuple of equivalence classes
in $D_H \langle e_1, \ldots, e_n \rangle$, let $\llbracket p \rrbracket^H(e_1, \ldots, e_n) = \text{True}$ if and only if for all $1 \leq i \leq n$ there exist
ground terms $t_i \in e_i$ such that $\llbracket p(t_1, \ldots, t_n) \rrbracket^M = \text{True}$. We show that $H$ satisfies $\Sigma$ and
that there exists a unique homomorphism $h$ from $H$ to $M$; from the assumption that $M$
is an initial model of $\Sigma$, it follows that $H$ is also an initial model of $\Sigma$.

We first show the existence of a unique homomorphism $h$ from $H$ to $M$. Let $h$ map
$e \in D_H$ to $d \in D_M$ if and only if for some ground term $t \in e$, $\llbracket t \rrbracket^M = d$. From the
definition of $H$, it follows that for every ground term $t \in e$ we have $\llbracket t \rrbracket^M = d = h(e)$. This implies that for every $n$-ary function $f$ ($n \geq 0$) and every $n$-tuple of individuals in $D_H \langle e_1, ..., e_n \rangle$, we have $h(\llbracket f \rrbracket^H(e_1, ..., e_n)) = \llbracket f \rrbracket^M(h(e_1), ..., h(e_n))$. It also implies that for every $n$-ary predicate $p$ ($n \geq 0$) and every $n$-tuple of individuals in $D_H \langle e_1, ..., e_n \rangle$, $\llbracket p \rrbracket^H(e_1, ..., e_n) = \text{True}$ implies $\llbracket p \rrbracket^M(h(e_1), ..., h(e_n)) = \text{True}$. Hence $h$ is a homomorphism from $H$ to $M$. Clearly any function $h' : D_H \to D_M$ that is distinct from $h$ violates the first requirement of a homomorphism; it is not the case that for every $n$-ary function $f$ ($n \geq 0$) and every $n$-tuple of individuals in $D_H \langle e_1, ..., e_n \rangle$, we have $h(\llbracket f \rrbracket^H(e_1, ..., e_n)) = \llbracket f \rrbracket^M(h'(e_1), ..., h'(e_n))$. Therefore, $h$ is unique.

It remains to show that $H$ satisfies $\Sigma$. Suppose $H$ does not. Then because $\Sigma$ is in Skolem Normal Form, and $M$ satisfies $\Sigma$, there must be some ground atom $p(t_1, ..., t_n)$ to which $M$ and $H$ assign different truth values. But by the construction of $H$ this is not possible. \hfill \square

It is worth noting that if $\Sigma$ is a Skolem Normal Form theory that has an initial model, and $C$ is a conjunction of ground atoms such that $\Sigma \cup \{C\}$ is satisfiable, then $\Sigma \cup \{C\}$ has an initial model. It follows that $\Sigma \cup \{C\}$ has an initial Herbrand-quotient model. This observation is useful in the proof of Theorem 33 in the next chapter. The following, related lemma is also useful in the proof.\footnote{This lemma is a simple generalization of Corollary 2 of Goguen and Meseguer [28].} The proof of this lemma uses the following notation: where $H$ is a Herbrand-quotient model and $t$ is a term, $\llbracket t \rrbracket_H$ denotes the equivalence class in $D_H$ to which $t$ belongs.

**Lemma 22 (Generalized Herbrand Theorem)** Let $\Sigma$ be any theory that has an initial Herbrand-quotient model $H$, and let $C$ be a conjunction of atoms. Then $\Sigma \models \exists C$ if and only if there exists a grounding substitution $\sigma$ for $C$ such that $H$ satisfies $C \sigma$.

**Proof:** From the definition of initial model, it follows that $\Sigma \models \exists C$ if and only if $H$ satisfies $\exists C$. We now show that $H$ satisfies $\exists C$ if and only if $H$ satisfies $C \sigma$. If $H$ satisfies $C \sigma$, for some substitution $\sigma$ that grounds $C$, then repeated application of existential generalization verifies that $H$ satisfies $\exists C$. Conversely, if $H$ satisfies $\exists C$, then there
exists an assignment \( g \) of individuals in the domain of \( D_H \) to the variables in \( C \) such that \([C]^H_\emptyset = \text{True}\). For each variable \( x \) in \( C \), let \( \sigma \) map \( x \) to a term \( t \) for which \( g \) maps \( x \) to \([t]_H\). Then \( H \) satisfies \( C\sigma \). \( \Box \)

We now return to the question of strong compactness. Because of Theorem 13, proving strong compactness requires only proving the following lemma.

**Lemma 23** Let \( \phi_1/C_1, ..., \phi_n/C_n \), and \( \phi/C \) be constrained atoms whose constraints are conjunctions of atoms, and let \( \Sigma \) be a Skolem Normal Form constraint theory that has an initial model. If \( \forall(\phi_1/C_1) \land ... \land \forall(\phi_n/C_n) \models \Sigma \forall(\phi/C) \) then for some \( 1 \leq i \leq n \) we have \( \forall(\phi_i/C_i) \models \Sigma \forall(\phi/C) \).

**Proof:** If \( C \) is not \( \Sigma \)-satisfiable then the result holds trivially. Otherwise, without loss of generality we may rewrite \( \phi_1/C_1, ..., \phi_n/C_n \), and \( \phi/C \) to constrained atoms with the same head, say \( \phi'/C'_1, ..., \phi'/C'_n \), and \( \phi'/C' \), respectively. (Any of \( \phi_1/C_1, ..., \phi_n/C_n \) whose heads do not have the same sign and predicate as \( \phi \) can be removed without loss of generality; if all are removed, the result holds trivially.) Notice that since \( C_1, ..., C_n \), and \( C \) are conjunctions of atoms, we may assume this rewriting is done in such a way that \( C'_1, ..., C'_n \), and \( C' \) are conjunctions of atoms (that may contain equality) as well. Furthermore, we may rewrite
\[
\forall(\phi'/C'_1) \land ... \land \forall(\phi'/C'_n),
\]
which is to say
\[
\forall(C'_1 \rightarrow \phi') \land ... \land \forall(C'_n \rightarrow \phi'),
\]
as \(
\forall((C'_1 \lor ... \lor C'_n) \rightarrow \phi').
\)
Therefore, we wish to show that if \( \forall((C'_1 \lor ... \lor C'_n) \rightarrow \phi') \models \Sigma \forall(C' \rightarrow \phi') \) then for some \( 1 \leq i \leq n \) we have \( \forall(C'_i \rightarrow \phi') \models \Sigma \forall(C' \rightarrow \phi') \). This is equivalent to showing that if \( \Sigma \models \forall(C' \rightarrow (C'_1 \lor ... \lor C'_n)) \) then for some \( 1 \leq i \leq n \) we have \( \Sigma \models \forall(C' \rightarrow C'_i) \). But since \( \Sigma \) is in Skolem Normal Form, and \( C'_1, ..., C'_n \), and \( C' \) are conjunctions of atoms, the statement is true if and only if the following statement is true, where \( \lambda \) is a (Skolemization) substitution that maps the variables in \( C'_1, ..., C'_n, C' \) to distinct constants not appearing in \( \Sigma, C'_1, ..., C'_n, \) or \( C' \): if \( \Sigma \models C'\lambda \rightarrow (C'_1\lambda \lor ... \lor C'_n\lambda) \) then for some \( 1 \leq i \leq n \) we have \( \Sigma \models C'\lambda \rightarrow C'_i\lambda \). This may be rewritten as: if \( \Sigma \cup \{C'\lambda\} \models C'_i\lambda \lor ... \lor C'_n\lambda \) then for some \( 1 \leq i \leq n \) we have \( \Sigma \cup \{C'\lambda\} \models C'_i\lambda \). Because \( \Sigma \) has an initial model, \( \Sigma \cup \{C'\lambda\} \) has an initial model as well—call it \( M \). If \( \Sigma \cup \{C'\lambda\} \models C'_i\lambda \lor ... \lor C'_n\lambda \) then \( M \) satisfies \( C'_i\lambda \lor ... \lor C'_n\lambda \). Therefore, since \( C'_i\lambda, ...,
$C'_{n}\lambda$ are ground atoms, $M$ must satisfy $C'_{i}\lambda$ for some $1 \leq i \leq n$. Then, because $M$ is an initial model of $\Sigma \cup \{C'\lambda\}$, we have $\Sigma \cup \{C'\lambda\} \models C'_{i}\lambda$. \hfill \qed

An analogous proof shows the following.

**Lemma 24** Let $\phi_1/C_1$, ..., $\phi_n/C_n$, and $\phi/C$ be constrained atoms whose constraints are conjunctions of atoms, and let $\Sigma$ be a Skolem Normal Form constraint theory that has an initial model. If $\exists(\phi/C) \models \Sigma \exists(\phi_1/C_1) \lor ... \lor \exists(\phi_n/C_n)$ then for some $1 \leq i \leq n$ we have $\exists(\phi/C) \models \Sigma \exists(\phi_i/C_i)$.

The preceding arguments yield the following general rendition of the strong compactness result.

**Theorem 25** (Strong Compactness for Constraint Logic) Let $\phi_1/C_1$, ..., $\phi_n/C_n$, and $\phi/C$ be constrained atoms whose constraints are conjunctions of atoms. Let $\Sigma$ be a Skolem Normal Form constraint theory that has an initial model. Then the following three statements are equivalent:

1. for some $1 \leq i \leq n$ we have $\phi_i/C_i \models_{\Sigma} \phi/C$

2. $\forall(\phi_1/C_1) \land ... \land \forall(\phi_n/C_n) \models_{\Sigma} \forall(\phi/C)$

3. $\exists(\phi/C) \models_{\Sigma} \exists(\phi_1/C_1) \lor ... \lor \exists(\phi_n/C_n)$

38
Chapter 3

Special Classes of Constrained Formulas

In this chapter we show that other, established instantiation orderings for various restricted classes of constrained formulas and constraint theories are equivalent to the \( \Sigma \)-more general ordering (given the appropriate restrictions). Our analysis is based in large part on Theorem 29 (Section 3.1) and Theorem 33 (Section 3.2), which under certain conditions provide alternative characterizations of the \( \geq \Sigma \) ordering, or more precisely, of condition (2) in the definition of that ordering. Using these theorems it is easy to equate the \( \geq \Sigma \) ordering with various established orderings.

3.1 A Characterization in the Absence of Equality

The first characterization we consider applies when no equality appears in the constraint theory or the constraints. No other restrictions are required. To present this characterization, we first require some additional definitions.

**Definition 26 (Weak Equivalence of Models Modulo Equality)** Two models \( M \) and \( M' \) are weakly-equivalent modulo equality if for any formula \( \phi \) that does not contain the equality predicate: (1) \( \semantics{\phi}{M,g} = \text{True} \) for some value assignment \( g \) if and only if \( \semantics{\phi}{M',g'} = \text{True} \) for some value assignment \( g' \), and (2) \( \semantics{\phi}{M,g} = \text{False} \) for some value
assignment \( g \) if and only if \( [\phi]^{M'}_{g'} = \text{False} \) for some value assignment \( g' \). Note that this implies that for any sentence \( \psi \) that does not contain the equality predicate: \( [\psi]^M = [\psi]^{M'} \).

We now define the construction of a Herbrand Image \( M' \) of a model \( M \), and we show that \( M' \) is weakly equivalent to \( M \) modulo equality. Let \( D_M \) and \( D_{M'} \) denote the domains of \( M \) and \( M' \), respectively. Let \( A_M \) and \( A_{M'} \) denote the mappings of \( M \) and \( M' \), respectively, that take predicate symbols to predicates and function symbols to functions. For each \( d \in D_M \) to which \( M \) maps no ground term, associate some distinct constant \( c \) that is not in the range of \( A_M \). We refer to \( d \) as an unnamed individual in \( M \), and we refer to \( c \) as an auxiliary constant. Let \( M' \) be a Herbrand model over these auxiliary constants, as well as the function symbols in the range of \( A_M \). We define a special function \( F: D_{M'} \rightarrow D_M \) as follows. For each auxiliary constant \( c \), \( F(c) = d \) where \( d \) is the unnamed individual with which \( c \) is associated. For any other ground term \( f(t_1, ..., t_n) \), \( n \geq 0 \), \( F(f(t_1, ..., t_n)) \) is defined recursively as the result of applying the function \( [f]^M \) to the tuple \( \langle F(t_1), ..., F(t_n) \rangle \). It is worth noting that \( F \) is a homomorphism from \( M' \) to \( M \). We now use the function \( F \) to define the mapping of \( A_{M'} \) to predicates, that is, to define the truth assignments of \( M' \). For any predicate symbol \( p \) denoting an \( n \)-ary predicate, \( [p]^{M'} \) maps a given tuple of ground terms \( \langle t_1, ..., t_n \rangle \) to \( \text{True} \) if and only if \( [p]^M \) maps \( \langle F(t_1), ..., F(t_n) \rangle \) to \( \text{True} \).

Based on the function \( F \), we define another function \( \mathcal{F} \) that maps the value assignments with the range \( D_{M'} \) onto the value assignments with the range \( D_M \). \( \mathcal{F} \) is defined as follows: for any variable \( x \), if a value assignment \( g \) maps \( x \) to the ground term \( t \) then \( \mathcal{F}(g) \) maps \( x \) to \( F(t) \). Because the function \( F \) maps \( D_{M'} \) onto \( D_M \), the function \( \mathcal{F} \) maps the value assignments with the range \( D_{M'} \) onto the value assignments with the range \( D_M \).

**Lemma 27** Let \( M' \) be any Herbrand Image of a model \( M \). Then \( M' \) is weakly equivalent to \( M \) modulo equality. More specifically, for any formula \( \phi \) that does not contain equality, \( [\phi]^{M'}_{g'} = [\phi]^{M,F(g')} \). (This implies the previous statement, since the function \( \mathcal{F} \) is onto.)
Proof: The proof is by induction on the complexity of \( \phi \). For the base case, \( \phi \) is an atomic formula. The result follows directly from the definitions of \( M' \) and \( \mathcal{F} \). For the inductive step, \( \phi \) has the form of either \( \forall x \phi' \), \( \exists x \phi' \), or \( \phi_1 \circ \phi_2 \), where \( \circ \) is some binary logical connective. Given the inductive hypothesis, the result is straightforward to verify for the last two cases. Furthermore, the proof for the second case is analogous to the proof for the first, so we present only the proof for the first case here.

First, assume \( \llbracket \forall x \phi \rrbracket^{M',\sigma}_{\mathcal{F}[x \mapsto t]} = \text{True} \). Then for every ground term \( t \) in \( D_{M'} \), \( \llbracket \phi' \rrbracket^{M',\sigma}_{\mathcal{F}[x \mapsto d]} = \text{True} \). Let \( t \) be an arbitrary ground term in \( D_{M'} \), and let \( F(t) = d \). By the inductive hypothesis, \( \llbracket \phi' \rrbracket^{M,\mathcal{F}(s[x \mapsto d])} = \llbracket \phi' \rrbracket^{M,\mathcal{F}(s[x \mapsto d])} = \text{True} \). Because \( t \) was arbitrary and \( \mathcal{F} \) is onto, \( \llbracket \phi' \rrbracket^{M,\mathcal{F}(s)} = \text{True} \) for all \( d \in D_M \), so \( \llbracket \forall x \phi \rrbracket^{M,\mathcal{F}(s)} = \text{True} \).

Now assume instead that \( \llbracket \forall x \phi \rrbracket^{M',\sigma} = \text{False} \). Then for some ground term \( t \) in \( D_{M'} \), \( \llbracket \phi' \rrbracket^{M',\sigma}_{\mathcal{F}[x \mapsto t]} = \text{False} \). Let \( F(t) = d \). By the inductive hypothesis, \( \llbracket \phi' \rrbracket^{M,\mathcal{F}(s[x \mapsto d])} = \llbracket \phi' \rrbracket^{M,\mathcal{F}(s[x \mapsto d])} = \text{False} \). Hence \( \llbracket \forall x \phi \rrbracket^{M,\mathcal{F}(s)} = \text{False} \). \( \square \)

One more lemma, below, is helpful in presenting the characterization.

**Lemma 28** Let \( M \) be a Herbrand model, and let \( t \) and \( s \) be terms such that for some pair of value assignments \( g \) and \( g' \) we have \( \llbracket t \rrbracket^M = [s]^M \). Furthermore, let \( \text{VAR}(t) = \{w_1, ..., w_m\} \), and let \( \theta = \{w_1 \mapsto r_1, ..., w_m \mapsto r_m\} \) be a substitution such that \( t\theta = s \). Then for all \( 1 \leq i \leq m \) we have \( [w_i]^M = [r_i]^M \).

Proof: Since \( M \) is a Herbrand model, any value assignment \( g \) has a corresponding grounding substitution \( \sigma \) such that, for every term \( t \), \( \llbracket t \rrbracket^M = [t]^M \). Let \( \sigma \) and \( \sigma' \) be the grounding substitutions that correspond to \( g \) and \( g' \), respectively. Then \( \llbracket t \rrbracket^M = [t]^M \) and \( \llbracket [s]^M \rrbracket = s \sigma' \). Because \( t \theta = s \), we know \( s \sigma' = t \theta \sigma' \). Therefore, since \( \llbracket t \rrbracket^M = [s]^M \rho' \), we have \( t \sigma = t \theta \sigma' \), so \( \sigma \) and \( \theta \cdot \sigma' \) must agree on the ground terms to which they map each \( w_i \). Therefore, for all \( 1 \leq i \leq m \), \( w_i \sigma = w_i \theta \sigma' = r_i \sigma' \). Since \( [w_i]^M = w_i \sigma \) and \( [r_i]^M = r_i \sigma' \), we have \( [w_i]^M = [r_i]^M \). \( \square \)

**Theorem 29** Let \( \phi_1/C_1 \) and \( \phi_2/C_2 \) be constrained formulas and let \( \Sigma \) be a constraint theory, none of which contains the equality predicate. Let \( \mathcal{F} \) denote the free variables
that appear in \( C_1 \) but not \( \phi_1 \), and let \( \bar{z} \) denote the free variables that appear in \( C_2 \) but not \( \phi_2 \). Let \( \phi_2/C_2 \) be \( \Sigma \)-admissible. Then \( \phi_1/C_1 \models \Sigma \phi_2/C_2 \) if and only if there exists a substitution \( \theta \) such that \( \phi_1 \theta = \phi_2 \) and \( \Sigma \models \overline{\forall}((\exists \bar{z} C_2) \rightarrow ((\exists \bar{y} C_1) \theta)) \).

**Proof:** The result is obvious if the constrained formulas do not have the same structure and predicates. Therefore, we need only consider the case where \( \phi_1/C_1 \) and \( \phi_2/C_2 \) are constrained formulas \( \phi[t_1, ..., t_n]/C_1 \) and \( \phi[s_1, ..., s_n]/C_2 \), respectively.

Let \( x_1, ..., x_n \) be variables that do not appear in \( \phi[t_1, ..., t_n]/C_1 \) or \( \phi[s_1, ..., s_n]/C_2 \). Then \( \phi[x_1, ..., x_n]/x_1 = t_1 \land ... \land x_n = t_n \land C_1 \) and \( \phi[x_1, ..., x_n]/x_1 = s_1 \land ... \land x_n = s_n \land C_2 \) are variable abstractions of \( \phi[t_1, ..., t_n]/C_1 \) and \( \phi[s_1, ..., s_n]/C_2 \), respectively. Recall that \( \bar{y} \) denotes the variables in \( C_1 \) but not \( t_1, ..., t_n \), and \( \bar{z} \) denotes the variables in \( C_2 \) but not \( s_1, ..., s_n \). Let \( \bar{u} \) denote the variables in \( t_1, ..., t_n, C_1 \), and let \( \bar{v} \) denote the variables in \( s_1, ..., s_n, C_2 \) (note that neither \( \bar{u} \) nor \( \bar{v} \) contains any of \( x_1, ..., x_n \)). We show that: (1) there exists a substitution \( \theta \) such that

\[
\phi[t_1, ..., t_n] \theta = \phi[s_1, ..., s_n]
\]

and

\[
\Sigma \models \overline{\forall}((\exists \bar{z} C_2) \rightarrow (\exists \bar{y} C_1) \theta)
\]

if and only if (2) we have

\[
\Sigma \models \overline{\forall}((\exists \bar{u} (x_1 = s_1 \land ... \land x_n = s_n \land C_2)) \rightarrow (\exists \bar{u} (x_1 = t_1 \land ... \land x_n = t_n \land C_1)))
\]

Because of the existential quantifiers, we may assume without loss of generality that the variables in \( \bar{y} \) are distinct from those in \( \phi_2/C_2 \) and the variables in \( \bar{z} \) are distinct from those in \( \phi_1/C_1 \).

(1) \( \Rightarrow \) (2): Assume (1) holds but (2) does not. We may further assume, without loss of generality, that \( \text{DOM}(\theta) \subseteq \text{VARS}(t_1, ..., t_n) \), so we may write \( \Sigma \models \overline{\forall}((\exists \bar{z} C_2) \rightarrow (\exists \bar{y} C_1) \theta) \)—the variables in \( \text{DOM}(\theta) \) are distinct from those in \( \bar{y} \), so we can remove the
parentheses around \( \exists \vec{y} C_1 \) without introducing ambiguity. Since (2) fails to hold, there exist a \( \Sigma \)-model \( M \) and a value assignment \( g \) such that \( [x_1 = s_1 \land \ldots \land x_n = s_n \land C_2]^{M,\sigma} = \text{True} \), yet for any value assignment \( \epsilon \) that agrees with \( g \) on \( x_1, \ldots, x_n \), \([x_1 = t_1 \land \ldots \land x_n = t_n \land C_1]^{M,\epsilon} = \text{False} \). Some useful results of the first part of this statement are: \([C_2]^{M,\sigma} = \text{True} \), and \([x_i = s_i]^{M,\sigma} = \text{True} \) for all \( 1 \leq i \leq n \).

Since (1) holds and \( M \) is a \( \Sigma \)-model, \( M \) satisfies \( \forall((\exists \vec{z} C_2) \rightarrow (\exists \vec{y} C_1)) \). Therefore, since \([C_2]^{M,\sigma} = \text{True} \), there exists a value assignment \( \vec{g}' \) that agrees with \( g \) on the assignments to all variables, except those in \( \vec{y} \), such that \([C_1]^{M,\vec{g}'} = \text{True} \). (Notice that \( \vec{g}' \) agrees with \( g \) on \( x_1, \ldots, x_n \), since the variables in \( \vec{y} \) appear in \( C_1 \), and the variables \( x_1, \ldots, x_n \) do not appear in \( C_1 \).) Let \( \vec{g}'' \) be a value assignment that maps each variable \( x \) to \([x]^{M,\vec{g}'} \). (Notice that \( \vec{g}'' \) agrees with \( \vec{g}' \) over all variables not in \( \text{DOM}(\theta) \). In particular, \( \vec{g}'' \) agrees with \( \vec{g}' \) over \( x_1, \ldots, x_n \), since \( \text{DOM}(\theta) \subseteq \text{VARS}(\phi_1) \), and \( x_1, \ldots, x_n \) do not appear in \( \phi_1 \). Furthermore, since \( \vec{g}' \) agrees with \( g \) over \( x_1, \ldots, x_n \), we know that \( \vec{g}'' \) agrees with \( g \) over \( x_1, \ldots, x_n \).) Then \([C_1]^{M,\vec{g}''} = \text{True} \). Since \( \text{DOM}(\theta) \subseteq \text{VARS}(t_1, \ldots, t_n) \), which omits \( x_1, \ldots, x_n \), it follows that for all \( 1 \leq i \leq n \), 

\[
[x_i = t_i]^{M,\vec{g}''} = [x_i = t_i\theta]^{M,\vec{g}'} = [x_i = s_i]^{M,\vec{g}'}
\]

Since \( \vec{g}' \) agrees with \( g \) on all but \( \vec{y} \), and no variable in \( \vec{y} \) appears in \( C_2 \), \( p(s_1, \ldots, s_n) \), or \( x_1, \ldots, x_n \), we have \([x_i = s_i]^{M,\vec{g}'} = [x_i = s_i]^{M,\sigma} \) for all \( 1 \leq i \leq n \). Therefore, since \([x_i = s_i]^{M,\sigma} = \text{True} \) for all \( 1 \leq i \leq n \), we have \([x_i = s_i]^{M,\vec{g}'} = \text{True} \) for all \( 1 \leq i \leq n \). Hence \([x_i = t_i]^{M,\vec{g}''} = \text{True} \) for all \( 1 \leq i \leq n \), so \([x_1 = t_1 \land \ldots \land x_n = t_n \land C_1]^{M,\vec{g}''} = \text{True} \). But because \( \vec{g}'' \) agrees with \( g \) over \( x_1, \ldots, x_n \), this statement contradicts the earlier statement that for any value assignment \( \epsilon \) that agrees with \( g \) on \( x_1, \ldots, x_n \), \([x_1 = t_1 \land \ldots \land x_n = t_n \land C_1]^{M,\epsilon} = \text{False} \).

(2) \( \Rightarrow \) (1): Assume (2). We first show that there exists a substitution \( \theta \) for which \( \phi[t_1, \ldots, t_n]\theta = \phi[s_1, \ldots, s_n] \). If

\[
\Sigma \models \forall((\exists \vec{z} (x_1 = s_1 \land \ldots \land x_n = s_n \land C_2)) \rightarrow (\exists \vec{u} (x_1 = t_1 \land \ldots \land x_n = t_n \land C_1)))
\]

then because \( \Sigma, C_1, \) and \( C_2 \) do not contain equality, it must be the case that

43
\[\nabla(\exists \bar{v} \ (x_1 = s_1 \land \ldots \land x_n = s_n)) \rightarrow (\exists \bar{u} \ (x_1 = t_1 \land \ldots \land x_n = t_n))\]

is valid. (No model \(M\) and value assignment \(g\) exist such that \([\exists \bar{v} \ (x_1 = s_1 \land \ldots \land x_n = s_n)]^{M,g} = \text{True} \) and \([\exists \bar{u} \ (x_1 = t_1 \land \ldots \land x_n = t_n)]^{M,g} = \text{False}\).) If it is valid then, in particular, it is true in any Herbrand model. But this is the case only if for any ground terms \(u_1, \ldots, u_n\): if for some substitution \(\theta_1, \ s_\theta_1 = u_i\) for all \(1 \leq i \leq n\) then, for some substitution \(\theta_2, \ t_\theta_2 = u_i\) for all \(1 \leq i \leq n\). Then any ground instance of \(\phi[s_1, \ldots, s_n]\) is a ground instance of \(\phi[t_1, \ldots, t_n]\), so \(\phi[t_1, \ldots, t_n] \geq \phi[s_1, \ldots, s_n]\)—that is, for some substitution \(\theta\) we have \(\phi[t_1, \ldots, t_n] \theta = \phi[s_1, \ldots, s_n]\). In particular, where \(\bar{u} = w_1, \ldots, w_m\) are the variables in \(t_1, \ldots, t_n\), there exists a substitution \(\theta = \{w_1 \mapsto r_1, \ldots, w_m \mapsto r_m\}\) such that \(\phi[t_1, \ldots, t_n] \theta = \phi[s_1, \ldots, s_n]\).

We now show that \(\Sigma \models \nabla(\exists \bar{v} \ C_2) \rightarrow (\exists \bar{y} \ C_1 \theta))\). Assume that for an arbitrary \(\Sigma\)-model \(M\) there exists a value assignment \(g_M\) such that \([C_2]^{M, g_M} = \text{True}\). We show that \([\exists \bar{y} \ C_1 \theta]^{M, g_M} = \text{True}\). Let \(M'\) be a Herbrand Image of \(M\). Because \(M'\) is weakly-equivalent to \(M\) modulo equality, and \(\Sigma\) contains no equality, \(M'\) is a \(\Sigma\)-model. Also, because \(M'\) is weakly-equivalent to \(M\), and \(C_2\) contains no equality, there exists a value assignment \(g\) such that \([C_2]^{M', g} = \text{True}\); more specifically, let \(g\) be such that \(\mathcal{F}(g) = g_M\)—such a \(g\) exists since \(\mathcal{F}\) is onto. Let \(g'\) be a value assignment that agrees with \(g\) on the values of all variables except (possibly) \(x_1, \ldots, x_n\), the values of which are chosen such that \([x_i]^{M', g'} = [s_i]^{M', g'}\) (\(= [s_i]^{M, g}\)) for all \(1 \leq i \leq n\). Then \([x_1 = s_1 \land \ldots \land x_n = s_n \land C_1]^{M', g'} = \text{True}\). Hence, from (2), there exists a value assignment \(g''\) that agrees with \(g'\) on \(x_1, \ldots, x_n\) such that \([x_1 = t_1 \land \ldots \land x_n = t_n \land C_1]^{M', g''} = \text{True}\). Then for all \(1 \leq i \leq n\), \([t_i]^{M', g''} = [x_i]^{M', g'} = [s_i]^{M', g'}\). Therefore by Lemma 28, since \(M'\) is a Herbrand model, for all \(1 \leq j \leq m\) we have \([w_j]^{M', g''} = [r_j]^{M', g''}\).

Now since \([C_1]^{M', g''} = \text{True}\), and every variable in \(C_1\) is in \(\bar{u}\) (the variables in \(\phi_1\)) or \(\bar{y}\) (the variables in \(C_1\) but not \(\phi_1\)), we know that \([\exists \bar{y} \ C_1]^{M', e} = \text{True}\) for any value assignment \(e\) that agrees with \(g''\) on \(\bar{u}\). Since for all \(1 \leq j \leq m\) we have \([w_j]^{M', g''} = [r_j]^{M', g''}\)
we know that \([\exists \vec{y} \ \exists \vec{w} \ (w_1 = r_1 \land \ldots \land w_m = r_m \land C_1)]^{M, \vec{\delta}'} = \text{True}\). Therefore, it follows from repeated application of Lemma 5 that \([\exists \vec{y} C_1 \theta]^{M, \vec{\delta}'} = \text{True}\). Since \(g'\) differs from \(g\) on only the assignments of \(x_1, \ldots, x_n\), which do not appear in \(C_1 \theta\), \([\exists \vec{y} C_1 \theta]^{M', \vec{\delta}} = \text{True}\). Since \(M'\) is a Herbrand Image of \(M\), \([\exists \vec{y} C_1 \theta]^{M, \vec{\delta} M} = \text{True}\). Finally, recall that we chose \(g\) such that \(\mathcal{F}(g) = g_M\), so \([\exists \vec{y} C_1 \theta]^{M, \vec{\delta} M} = \text{True}\). \(\square\)

It is worth noting that if there exists a substitution \(\theta\) such that \(\phi_1 \theta = \phi_2\) and also \(\Sigma \models \neg((\exists z C_2) \rightarrow ((\exists y C_1) \theta))\), then for any substitution \(\theta\) such that \(\phi_1 \theta = \phi_2\) we have \(\Sigma \models \neg((\exists z C_2) \rightarrow ((\exists y C_1) \theta))\). This is because the only variables on which such substitutions may differ are those that do not appear in \(\phi_1\), and any variable in \(C_1\) that does not appear in \(\phi_1\) is already existentially quantified, so the substitution is not applied to these variables. The same reasoning may be applied to the characterization in the next section (Theorem 33).

The following is an example of this characterization.

### Example 30

\[
\text{loves}(x, y) / \text{Mammal}(x) \land \text{Mammal}(y) \geq_{\Sigma_1} \text{loves}(z, \text{mom}(z)) / \text{Elephant}(z)
\]

Notice first that for any substitution \(\theta\) that maps \(x\) to \(z\) and \(y\) to \(\text{mom}(z)\):

\[
\text{loves}(x, y) \theta = \text{loves}(z, \text{mom}(z))
\]

Second, note that

\[
(\text{Mammal}(x) \land \text{Mammal}(y)) \theta = \text{Mammal}(z) \land \text{Mammal}(\text{mom}(z))
\]

Because \(\Sigma_1\) entails that the mom of an elephant is an elephant and that all elephants are mammals, we have

\[
\Sigma_1 \models \forall z (\text{Elephant}(z) \rightarrow (\text{Mammal}(z) \land \text{Mammal}(\text{mom}(z)))
\]

45
Notice that the lone restriction on constrained formulas and constraint theories in Theorem 29 is the complete absence of equality. To see why equality is forbidden, consider an example where $\Sigma = \{\forall x \forall y (f(x,y) = f(y,x))\}$. Then $p(f(b,x)) \geq \Sigma p(f(a,b))$, but not according to the characterization. Similarly, consider a case where $\Sigma$ is empty but equality appears in a constraint: $p(a) \geq \Sigma p(b)/a = b$, but not according to the characterization.

### 3.2 A Characterization that Allows Limited Equality

Theorem 33, which follows, allows equality in the constraint theory, $\Sigma$, though not in the constrained formulas. In exchange for equality in $\Sigma$, some additional restrictions on $\Sigma$ and the form of the constraints are required. Specifically, $\Sigma$ must a Skolem Normal Form theory that has an initial model, and constraints must be conjunctions of atoms built from constraint predicates. After Theorem 33 we will see the reason for using these restrictions. Notice that these are precisely the restrictions used in Theorem 25 about strong compactness.

The presentation of Theorem 33 requires one more definition and a related lemma.

**Definition 31 (E-Transform)** Let $\Sigma$ be a constraint theory, let $\phi/C$ be a constrained formula, and let $t_1$ and $t_2$ be terms. If $\Sigma \models \forall(C \rightarrow (t_1 = t_2))$, and $t_1$ occurs in $\phi$, then $\phi/C$ can be rewritten to $\phi'/C$, where $\phi'$ is the result of replacing some occurrence of $t_1$ in $\phi$ by $t_2$. If $\phi''/C$ results from applying a sequence of such rewriting steps to $\phi/C$, then we say that $\phi''/C$ is an E-transform of $\phi/C$ with respect to $\Sigma$.

Note that if $\phi'/C$ is a $\Sigma$-transform of $\phi/C$ then obviously $\phi/C$ is a $\Sigma$-transform of $\phi'/C$.

**Lemma 32 (Correctness of E-Transformation)** If $\phi/C$ and $\phi'/C$ are E-transforms of one another with respect to $\Sigma$, then $\Sigma \models \forall(C \rightarrow (\phi \leftrightarrow \phi'))$, which implies in particular that $\phi/C$ and $\phi'/C$ are closure-equivalent with respect to $\Sigma$. 

46
Proof: It suffices to prove the result when only one application of the E-transformation rule for constrained formulas is needed to obtain $\phi'/C$ from $\phi/C$. Assume the application replaces some occurrence of a term $t_1$ in $\phi$ with a term $t_2$ to yield $\phi'$. We prove the lemma by proving its contrapositive. Suppose that for some $\Sigma$-model $M$ and some value assignment $g$, $[C \rightarrow \phi]^{M_g}$ is True but $[C \rightarrow \phi']^{M_g}$ is False. Then $[C]^{M_g}$ is True, $[\phi]^{M_g}$ is True, and $[\phi']^{M_g}$ is False. Since $[\phi]^{M_g}$ and $[\phi']^{M_g}$ have different truth values, $[t_1]^{M_g}$ and $[t_2]^{M_g}$ must be different domain individuals. But this cannot be the case: since $[C]^{M_g}$ is True, and $\Sigma \models \forall(C \rightarrow (t_1 = t_2))$, and $M$ is a $\Sigma$-model, we know $[t_1 = t_2]^{M_g}$ is True, which means $[t_1]^{M_g}$ and $[t_2]^{M_g}$ are the same individual. \hfill \Box

**Theorem 33** Let $\phi_1/C_1$ and $\phi_2/C_2$ be constrained formulas, where $C_1$ and $C_2$ are conjunctions of atoms that do not contain the equality predicate. Let $\Sigma$ be a constraint theory in Skolem Normal Form that has an initial model, and let $\phi_2/C_2$ be $\Sigma$-admissible. Then $\phi_1/C_1 \geq \Sigma \phi_2/C_2$ if and only if for some E-transform $\phi_2'/C_2$ of $\phi_2/C_2$: there exists a substitution $\theta$ such that $\phi_1 \theta = \phi_2'$ and $\Sigma \models \forall((\exists \bar{z} C_2) \rightarrow ((\exists \bar{y} C_1) \theta))$, where $\bar{y}$ denotes the free variables that appear in $C_1$ but not $\phi_1$, and $\bar{z}$ denotes the free variables that appear in $C_2$ but not $\phi_2'$.

Proof: The result is obvious if $\phi_1/C_1$ and $\phi_2/C_2$ are constrained formulas whose heads have different structure or predicates. Otherwise $\phi_1/C_1$ and $\phi_2/C_2$ may be written as $\phi[t_1, \ldots, t_n]/C_1$ and $\phi[s_1, \ldots, s_n]/C_2$, where $C_1$ and $C_2$ are $\Sigma$-satisfiable, equality-free conjunctions of atoms built from constraint predicates. Let $\bar{y}$ denote the variables in $C_1$ but not $t_1, \ldots, t_n$, and let $\bar{z}$ denote the variables in $C_2$ but not $s_1, \ldots, s_n$. Let $\bar{u}$ denote the variables in $t_1, \ldots, t_n, C_1$, let $\bar{v}$ denote the variables in $s_1, \ldots, s_n, C_2$, and let $x_1, \ldots, x_n$ be distinct variables that appear in neither $\bar{u}$ nor $\bar{v}$. We show that

$$\Sigma \models \forall((\exists \bar{v}(x_1 = s_1 \land \ldots \land x_n = s_n \land C_2)) \rightarrow (\exists \bar{u}(x_1 = t_1 \land \ldots \land x_n = t_n \land C_1)))$$

if and only if there exist terms $s_1', \ldots, s_n'$ and a substitution $\theta$ such that:

1. $\Sigma \models \forall(C_2 \rightarrow (s_i = s_i'))$ for all $1 \leq i \leq n$, 

47
2. $\phi[t_1, ..., t_n]\theta = \phi[s_1', ..., s_n']$, and

3. $\Sigma \models \forall(\exists \vec{z} \ C_2 \rightarrow (\exists \vec{y} \ C_1\theta))$

Because of the existential quantifiers, we may assume without loss of generality that the variables in $\vec{y}$ do not appear in $C_2$.

Let $\vec{v} = \langle v_1, ..., v_m \rangle$, let $\vec{v'} = \langle v'_1, ..., v'_m \rangle$ be a vector of variables that do not appear in $C_1$, $C_2$, $s_1$, ..., $s_n$, $t_1$, ..., $t_n$, $x_1$, ..., $x_n$, and let $\varphi = \{v_1 \mapsto v'_1, ..., v_m \mapsto v'_m\}$ be a substitution. Notice that

$$\Sigma \models \forall(\exists \vec{u}(x_1 = s_1 \wedge ... \wedge x_n = s_n \wedge C_2) \rightarrow (\exists \vec{u}(x_1 = t_1 \wedge ... \wedge x_n = t_n \wedge C_1)))$$

if and only if

$$\Sigma \models \forall(\exists \vec{v}(x_1 = s_1 \varphi \wedge ... \wedge x_n = s_n \varphi \wedge C_2 \varphi) \rightarrow (\exists \vec{u}(x_1 = t_1 \wedge ... \wedge x_n = t_n \wedge C_1)))$$

because of the existential quantifiers on $\vec{v}$ and $\vec{v'}$. Because each variable in $\vec{u}$ appears in $t_1$, ..., $t_n$, or $C_1$, we know that $\vec{v'}$ and $\vec{v}$ are disjoint. Because $\vec{v'}$ and $\vec{u}$ are disjoint, and $\vec{v'}$ contains all the variables in $s_1 \varphi$, ..., $s_n \varphi$, $\vec{u}$ contains no variables of $s_1 \varphi$, ..., $s_n \varphi$. We will use this fact in a moment, after noting the following. We may rewrite

$$\Sigma \models \forall(\exists \vec{v}(x_1 = s_1 \varphi \wedge ... \wedge x_n = s_n \varphi \wedge C_2 \varphi) \rightarrow (\exists \vec{u}(x_1 = t_1 \wedge ... \wedge x_n = t_n \wedge C_1)))$$

as

$$\Sigma \models \forall((x_1 = s_1 \varphi \wedge ... \wedge x_n = s_n \varphi \wedge C_2 \varphi) \rightarrow (\exists \vec{u}(x_1 = t_1 \wedge ... \wedge x_n = t_n \wedge C_1)))$$

because the variables of $\vec{v'}$ do not occur free in the formula $\exists \vec{u}(x_1 = t_1 \wedge ... \wedge x_n = t_n \wedge C_1)$ (only $x_1, ..., x_n$ occur free in that formula). Furthermore, we may rewrite

$$\Sigma \models \forall((x_1 = s_1 \varphi \wedge ... \wedge x_n = s_n \varphi \wedge C_2 \varphi) \rightarrow (\exists \vec{u}(x_1 = t_1 \wedge ... \wedge x_n = t_n \wedge C_1)))$$

as

48
$$\Sigma \models \forall C \varrho \rightarrow ((x_1 = s_1 \varrho \land \ldots \land x_n = s_n \varrho) \rightarrow (\exists \bar{u}(x_1 = t_1 \land \ldots \land x_n = t_n \land C_1)))$$

Now, since $\bar{u}$ contains no variables of $s_1 \varrho$, ..., $s_n \varrho$, nor $x_1$, ..., $x_n$, we have

$$\Sigma \models \forall C \varrho \rightarrow ((x_1 = s_1 \varrho \land \ldots \land x_n = s_n \varrho) \rightarrow (\exists \bar{u}(x_1 = t_1 \land \ldots \land x_n = t_n \land C_1)))$$

if and only if

$$\Sigma \models \forall C \varrho \rightarrow (\exists \bar{u}(s_1 \varrho = t_1 \land \ldots \land s_n \varrho = t_n \land C_1))$$

Let $\lambda$ be a substitution that, for all $1 \leq i \leq n$, maps $v'_i$ to a constant $v''_i$ that does not appear in $\Sigma$, $C_1$, $C_2$, $s_1$, ..., $s_n$, $t_1$, ..., $t_n$. Then

$$\Sigma \models \forall C \varrho \rightarrow (\exists \bar{u}(s_1 \varrho = t_1 \land \ldots \land s_n \varrho = t_n \land C_1))$$

if and only if

$$\Sigma \models C_2 \varrho \lambda \rightarrow (\exists \bar{u}(s_1 \varrho \lambda = t_1 \land \ldots \land s_n \varrho \lambda = t_n \land C_1))$$

because

$$\Sigma \Rightarrow C_2 \varrho \lambda \rightarrow (\exists \bar{u}(s_1 \varrho \lambda = t_1 \land \ldots \land s_n \varrho \lambda = t_n \land C_1))$$

is a Skolem Normal Transform of

$$\Sigma \Rightarrow \forall (C \varrho \rightarrow (\exists \bar{u}(s_1 \varrho = t_1 \land \ldots \land s_n \varrho = t_n \land C_1)))$$

Let $\lambda' = \{v_1 \mapsto v''_1, \ldots, v_m \mapsto v''_m\}$ be a substitution. Then $\lambda' = \varrho \cdot \lambda$. Therefore we may rewrite

$$\Sigma \models C_2 \varrho \lambda \rightarrow (\exists \bar{u}(s_1 \varrho \lambda = t_1 \land \ldots \land s_n \varrho \lambda = t_n \land C_1))$$

49
as

$$\Sigma \models C_2 \chi \rightarrow (\exists \bar{u}(s_1 \chi = t_1 \land \ldots \land s_n \chi = t_n \land C_1))$$

We know that \( \Sigma \) has an initial model. Since \( C_2 \) is \( \Sigma \)-satisfiable, \( \Sigma \not\models \neg C_2 \). Since \( \Sigma \Rightarrow \neg C_2 \chi \) is a Skolem Normal Transform of \( \Sigma \Rightarrow \neg C_2 \), \( \Sigma \not\models \neg C_2 \chi \). Thus \( C_2 \chi \) is \( \Sigma \)-satisfiable. Therefore, since \( C_2 \chi \) is a conjunction of ground atoms, and \( \Sigma \) has an initial model, \( \Sigma \cup \{C_2 \chi\} \) has an initial model—more specifically, it has an initial Herbrand-quotient model, \( H \). By Lemma 22, we know that

$$\Sigma \models C_2 \chi \rightarrow (\exists \bar{u}(s_1 \chi = t_1 \land \ldots \land s_n \chi = t_n \land C_1))$$

if and only if there exists a grounding substitution (for \( t_1, \ldots, t_n, C_1 \)) \( \sigma \) such that \( H \) satisfies \( s_1 \chi = t_1 \sigma \land \ldots \land s_n \chi = t_n \sigma \land C_1 \sigma \). But this is true if and only if (1) \( \Sigma \models C_2 \chi \rightarrow (s_1 \chi = t_1 \sigma \land \ldots \land s_n \chi = t_n \sigma) \) and (2) \( \Sigma \models C_2 \chi \rightarrow C_1 \sigma \).

Let \( \sigma' \) be the result of replacing every occurrence of \( v_i'' \), for all \( 1 \leq i \leq n \), in the range of \( \sigma \) with \( v_i \). Then (1) \( \Sigma \Rightarrow C_2 \chi \rightarrow (s_1 \chi = t_1 \sigma \land \ldots \land s_n \chi = t_n \sigma) \) is a Skolem Normal Transform of \( \Sigma \Rightarrow \neg(C_2 \rightarrow (s_1 = t_1 \sigma' \land \ldots \land s_n = t_n \sigma')) \), and (2) \( \Sigma \models C_2 \chi \rightarrow C_1 \sigma \) is a Skolem Normal Transform of \( \Sigma \models \neg(C_2 \rightarrow C_1 \sigma) \). Therefore (1) \( \Sigma \models C_2 \chi \rightarrow (s_1 \chi = t_1 \sigma \land \ldots \land s_n \chi = t_n \sigma) \) and (2) \( \Sigma \models C_2 \chi \rightarrow C_1 \sigma \) if and only if: (1) \( \Sigma \models \neg(C_2 \rightarrow (s_1 = t_1 \sigma' \land \ldots \land s_n = t_n \sigma')) \) and (2) \( \Sigma \models \neg(C_2 \rightarrow C_1 \sigma) \).

Let \( s_i' \) be \( t_i \sigma' \), for all \( 1 \leq i \leq n \), and let \( \theta \) be \( \sigma' \). Then (1) \( \Sigma \models \neg(C_2 \rightarrow (s_1 = t_1 \sigma' \land \ldots \land s_n = t_n \sigma')) \) and (2) \( \Sigma \models \neg(C_2 \rightarrow C_1 \sigma) \) if and only if:

1. \( \Sigma \models \neg(C_2 \rightarrow (s_i = s_i')) \) for all \( 1 \leq i \leq n \),

2. \( \phi[t_1, \ldots, t_n] \theta = \phi[s_1', \ldots, s_n'], \) and

3. \( \Sigma \models \neg((\exists \bar{y} C_2) \rightarrow ((\exists \bar{y'} C_1) \theta)) \) (since the variables in \( \bar{y} \) are disjoint from those in \( C_2 \)).

\( \square \)
Let’s return to an earlier example to illustrate this characterization. Recall the constraint theory \( \Sigma_3 \), and consider the following pair of constrained formulas.

\[
\text{controls}(\text{political-party}(\text{spouse(arnold)}), x, 1993) / \text{legislature}(x) \land \\
(\forall w \ \text{us-state}(w) \rightarrow (\exists z \ \text{represents}(z, w, x)))
\]

\[
\text{controls}(\text{democratic-party, us-house-of-representatives,1993})
\]

Notice first that \( \Sigma_3 \) sanctions the rewriting of

\[
\text{controls}(\text{democratic-party, us-house-of-representatives,1993})
\]

to

\[
\text{controls}(\text{political-party}(\text{spouse(arnold)}), \text{us-house-of-representatives,1993})
\]

Then the substitution \( \theta = \{ x \mapsto \text{us-house-of-representatives} \} \) maps

\[
\text{controls}(\text{political-party}(\text{spouse(arnold)}), x, 1993)
\]

to

\[
\text{controls}(\text{political-party}(\text{spouse(arnold)}), \text{us-house-of-representatives,1993})
\]

Finally, the constraint of the second constrained formula is empty, and \( \Sigma_3 \) entails

\[
(\text{legislature}(x) \land (\forall w \ \text{us-state}(w) \rightarrow (\exists z \ \text{represents}(z, w, x))))\theta
\]

The following examples show that without the additional restrictions the characterization of Theorem 33 is incomplete. In the first example, \( \Sigma \) does not have an initial
model, because of the disjunction, and in the second, the constraint of one constrained formula contains disjunction. For the first example, again let \( \Sigma_D \) be

\[
\{ \text{BABY}(ralph) \lor \text{BOY}(ralph), \forall x \forall y(f(x, y) = f(y, x)) \}
\]

Then

\[
p(f(x, y))/\text{BABY}(x) \land \text{BOY}(y) \geq_{\Sigma_D} p(f(\text{ralph}, z))/\text{BABY}(z) \land \text{BOY}(z)
\]

but not according to the characterization. For the second example, let \( \Sigma_E \) be the theory

\[
\{ \forall x \forall y(f(x, y) = f(y, x)) \}. \text{ Then}
\]

\[
p(f(x, y))/\text{BABY}(x) \land \text{BOY}(y) \geq_{\Sigma_E} p(f(\text{ralph}, z))/\text{BABY}(z) \land \text{BOY}(z)
\]

but not according to the characterization.

We are now prepared to relate the \( \geq_{\Sigma} \) ordering to four orderings that form the basis of the anti-unification operations studied in the remainder of the thesis. The first two orderings have been well-studied and are the established orderings for sorted logic and for reasoning with equality by \( E \)-unification, respectively.

### 3.3 The Instantiation Ordering for Sorted Logic

As noted in the Section 1, if the predicates in a constraint theory are unary, the theory is called a sort theory. (Notice that a sort theory cannot contain the equality predicate, which is binary.) In sorted logic, the constraint theory is a sort theory, and the formulas are sorted formulas. A sorted formula is a constrained formula whose constraint is a conjunction of atoms built from unary constraint predicates and variables that appear in the head, such that each variable occurs at most once in the constraint. Because of these additional restrictions on the constraints of sorted formulas, sorted formulas are
often represented in an alternative, *in-line* syntax, in which the constraints are attached directly to the variables. Thus, for example, the sorted formula

\[ \text{eats}(x, y) / \text{ELEPHANT}(x) \land \text{VEGETABLE}(y) \]

may be expressed as

\[ \text{eats}(x: \text{ELEPHANT}, y: \text{VEGETABLE}) \]

**Definition 34 (In-line Representation)** Let \( \phi / \tau_1(x_1) \land \ldots \land \tau_n(x_n) \) be a sorted formula. Then \( \alpha \) is the in-line representation of \( \phi / C \) if and only if \( \alpha \) results from replacing every occurrence of the variable \( x_i \) in \( \phi \) with an occurrence of the sorted variable \( x_i: \tau_i \), for all \( 1 \leq i \leq n \).

The additional restrictions on sorted formulas make possible another definition of an instantiation ordering on sorted formulas (called \( S \)-more general, or \( \geq_S \), in Definition 35), in which substitutions are central; this is the established instantiation ordering for sorted formulas. Theorem 36 states that this established ordering on sorted formulas is the same as the \( \geq_{\Sigma} \) ordering, provided a small additional assumption is made. The assumption is that every sort contains at least one individual, and it is made for the established ordering anyway [26]. The definition of the established ordering for sorted formulas, given below, is based on the in-line syntax for sorted formulas.

**Definition 35 (\( S \)-more general for Sorted Formulas)** Let \( S \) be a sort theory such that for every sort \( \tau \) we have \( S \models \exists x \, \tau(x) \) (according to \( S \) every sort contains some individual). A substitution \( \theta \) is a well-sorted substitution (with respect to \( S \)) if and only if for any variable \( x: \tau \), \( x: \tau \theta = t \) where \( S \models \overline{\tau}(t) \). A sorted formula \( \alpha_1 \) is \( S \)-more general than another, \( \alpha_2 \), if and only if \( \alpha_1 \theta = \alpha_2 \) for some well-sorted substitution \( \theta \).

**Theorem 36 (\( S \)-more general and \( \geq_{\Sigma} \) Equivalence)** Let \( \alpha_1 \) and \( \alpha_2 \) be sorted formulas, where \( \alpha_1 \) can be written as \( \phi_1 / C_1 \), and \( \alpha_2 \) can be written as \( \phi_2 / C_2 \). Let \( S = \Sigma \)}

53
be a sort theory such that for every sort \( \tau \), \( S \models \exists x \tau(x) \). Then \( \phi_1/C_1 \geq \Sigma \phi_2/C_2 \) if and only if \( \alpha_1 \) is \( S \)-more general than \( \alpha_2 \).

Proof: Let \( y_1 : \tau_1, \ldots, y_n : \tau_n \) be the variables in \( \alpha_1 \) and \( z_1 : \omega_1, \ldots, z_m : \omega_m \) the variables in \( \alpha_2 \). Therefore \( C_1 \) is \( \tau_1(y_1) \land \ldots \land \tau_n(y_n) \), and \( C_2 \) is \( \omega_1(z_1) \land \ldots \land \omega_m(z_m) \). If no substitution \( \theta \) exists such that \( \alpha_1 \theta = \alpha_2 \), then no substitution \( \theta' \) exists such that \( \phi_1 \theta' = \phi_2 \), so the result holds. Otherwise, a substitution \( \theta \) exists such that \( \text{DOM}(\theta) \subseteq \text{VARS}(\alpha_1) \) and \( \alpha_1 \theta = \alpha_2 \).

Let \( \theta' = \{ y_1 \mapsto t_1, \ldots, y_m \mapsto t_m \} \) if and only if \( \theta = \{ y_1 : \tau_1 \mapsto t'_1, \ldots, y_m : \tau_m \mapsto t'_m \} \), where \( t_i \) for all \( 1 \leq i \leq n \), is the result of removing all sorts that are attached to variables in \( t'_i \). Then \( \alpha_1 \theta = \alpha_2 \) if and only if \( \phi_1 \theta' = \phi_2 \). By Theorem 29 it remains only to show that \( \theta \) is well-sorted with respect to \( S \) if and only if \( \Sigma \models \forall(\exists z C_2) \rightarrow ((\exists y C_1)\theta') \). This follows directly once we note that, because any variable that appears in the constraint of a sorted formula also appears in the head, we can simplify \( \Sigma \models \forall(\exists z C_2) \rightarrow ((\exists y C_1)\theta') \) in Theorem 29 to \( \Sigma \models \forall(C_2 \rightarrow (C_1\theta')) \). \( \square \)

Because sorted anti-unification, defined in Chapter 5, is based on the instantiation ordering applied to sorted atoms, we now give several examples of the \( \geq \Sigma \) ordering on sorted atoms (or of the \( S \)-more general ordering on sorted atoms, since the two orderings are the same over all sorted formulas). We use the characterization of \( \geq \Sigma \) given in Theorem 29. The examples involve the sort theory \( \Sigma_1 \) given in Chapter 2, which we repeat below.

\[
\Sigma_1 = \{ \forall x \text{ univ}(x), \text{ elephant}(clyde), \text{ in-circus}(clyde), \text{ gray}(mom(clyde)), \\
\text{ elephant(jumbo), in-circus(jumbo), gray(mom(jumbo))}, \\
\forall x \text{ elephant}(x) \rightarrow \text{ elephant}(mom(x)), \\
\forall x \text{ elephant}(x) \rightarrow \text{ mammal}(x) \}.
\]

**Example 37**

\[
eats(x : \text{ elephant}, nuls) \geq \Sigma_1 \eats(clyde, nuls)
\]

Using the original syntax of constrained formulas, this relationship is expressed as follows.
$eats(x, \text{nuts}) / \text{ELEPHANT}(x) \geq_{\Sigma_1} eats(\text{clyde, nuts})$

To see that the relationship holds, first notice that for any substitution $\theta$ that maps $x$ to $\text{clyde},$

$eats(x, \text{nuts})\theta = eats(\text{clyde, nuts})$

Thus $\theta$ satisfies the first requirement of the definition of the $\geq_{\Sigma}$ ordering. Second, note that because the constraint of the second atom is empty (since it has no sorted variables), the second part of the definition of $\geq_{\Sigma}$ requires that $\Sigma_1 \models \text{ELEPHANT}(\text{clyde}),$ which clearly holds. No universal quantifiers appear here simply because no variables are present to be quantified.

On the other hand, the following is a negative example of the ordering.

**Example 38**

$eats(x: \text{ELEPHANT, nuts}) \not\geq_{\Sigma_1} eats(\text{fido, nuts})$

Any substitution that maps $eats(x, \text{nuts})$ to $eats(\text{fido, nuts})$ must map $x$ to $\text{fido}$. But $\Sigma_1$ does not entail that $\text{fido}$ is an elephant.

Both of the sorted atoms in the next example have sorts attached.

**Example 39**

$eats(y: \text{MAMMAL, nuts}) \geq \Sigma eats(x: \text{ELEPHANT, nuts})$

In the syntax of constrained formulas, this example is written as

$eats(y, \text{nuts}) / \text{MAMMAL}(y) \geq \Sigma eats(x, \text{nuts}) / \text{ELEPHANT}(x)$
To verify the relationship, let $\theta$ be any substitution that maps $y$ to $x$. Then

$$eals(y,\text{nuts})\theta = eals(x,\text{nuts})$$

The second part of the definition of $\geq \Sigma$ requires that

$$\Sigma_1 \models \forall x (\text{elephant}(x) \rightarrow \text{mammal}(x))$$

This requirement is met because $\Sigma_1$ entails that all elephants are mammals.

**Example 40**

$$\text{loves}(x:\text{mammal}, y:\text{mammal}) \geq\Sigma_1 \text{loves}(z:\text{elephant}, \text{mom}(z:\text{elephant}))$$

This example may also be written as follows.

$$\text{loves}(x,y)/\text{mammal}(x) \land \text{mammal}(y) \geq\Sigma_1 \text{loves}(z,\text{mom}(z))/\text{elephant}(z)$$

From this writing, we see that this is simply Example 30, given earlier as an example of the characterization of the $\geq \Sigma$ ordering in Theorem 29.

The definition of the $\geq \Sigma$ ordering applies equally well to sorted terms as to sorted atoms. We use this application in Chapter 5; here is an example.

**Example 41**

$$z:\text{elephant} \geq\Sigma_1 \text{mom}(\text{mom}(x:\text{elephant}))$$

This example may also be written as

$$z/\text{elephant}(z) \geq\Sigma_1 \text{mom}((\text{mom}(x))/\text{elephant}(x)$$

Let $\theta$ be any substitution that maps $z$ to $\text{mom}(\text{mom}(x))$. Then
\((\text{elephant}(z))\theta = \text{elephant}(\text{mom}(\text{mom}(x)))\)

and

\[\Sigma_1 \models \forall x(\text{elephant}(x) \rightarrow \text{elephant}(\text{mom}(\text{mom}(x))))\]

### 3.4 The Ordering Used in E-Unification and E-Anti-Unification

E-Unification is a form of unification that takes into account a background theory, \(E\), about equality. In Chapter 7 we define the dual operation, E-anti-unification. In this brief section we examine the instantiation ordering on which \(E\)-unification implicitly is based, and we show that the ordering is in fact the \(\geq_\Sigma\) ordering. Chapter 7 bases the definition of \(E\)-anti-unification on this ordering explicitly.

Following Plotkin [58], we say that \(\sigma\) is an \(E\)-unifier of literals \(L\) and \(M\) if and only if \(L\sigma\) and \(M\sigma\) are \(E\)-equivalent. Plotkin requires that \(E\) be a “theory all of whose sentences are universal closures of equations.” Jaffar, Lassez, and Maher [35] relax this restriction to allow \(E\) to be any Horn clause theory whose only predicate is the equality predicate. It is worth noting that any such theory has an initial model. (This is obvious since we have already noted that any Horn clause theory built from the equality predicate and possibly other predicates has an initial model.) We can lighten the restriction still more, and require only that \(E\) be a constraint theory in Skolem Normal Form that has an initial model. We take \(E\)-unification to require only this lighter restriction. (Thus, for example, several constraint predicates other than equality could be used to specify conditions under which various equalities hold). Theorem 42, below, follows from Theorem 33 and shows that \(E\)-unification is based on precisely the \(\geq_\Sigma\) ordering. We take this ordering as the basis of \(E\)-anti-unification in Chapter 7 as well.
Theorem 42 Let \( E = \Sigma \) be a constraint theory. For any two literals \( L \) and \( M \), which share no variables, (1) if \( \sigma \) is an \( E \)-unifier of \( L \) and \( M \) then \( L\sigma \) is a lower bound of \( L \) and \( M \) in the \( \geq_\Sigma \) ordering (\( L \geq_\Sigma L\sigma \) and \( M \geq_\Sigma L\sigma \)), and (2) if \( L\sigma \) is a lower bound of \( L \) and \( M \) in the \( \geq_\Sigma \) ordering then there exists an \( E \)-unifier, \( \theta \), of \( L \) and \( M \) such that \( L\theta = L\sigma \).

Proof: (1) Since \( L \) has no constraints, trivially by Theorem 33 we have \( L \geq_\Sigma L\sigma \). If \( \sigma \) is an \( E \)-unifier of \( L \) and \( M \), then \( L\sigma \) and \( M\sigma \) are \( \Sigma \)-equivalent. Therefore, by Theorem 14, \( M \geq_\Sigma L\sigma \). (2) Let \( \sigma' \) be a substitution such that \( L\sigma' = L\sigma \) and \( \sigma' \) maps any variable \( x \) not in \( L \) to \( x \) itself. Because \( M \geq_\Sigma L\sigma \), we know \( M \geq_\Sigma L\sigma' \); therefore, \( L\sigma' \) is an \( E \)-transform of \( M\theta \), for some substitution \( \theta \). (Without loss of generality, we may assume that \( \theta \) maps any variable \( y \) not in \( M \) to \( y \) itself.) Therefore, \( \Sigma \models \neg(L\sigma' \leftrightarrow M\theta) \), that is, \( L\sigma' \) and \( M\theta \) are \( \Sigma \)-equivalent. Because \( L \) and \( M \) share no variables, \( L\sigma'\theta = L\sigma' \) and \( M\sigma'\theta = M\theta \). Therefore, \( L\sigma' = L\sigma'\theta \) and \( M\sigma'\theta \) are \( \Sigma \)-equivalent, that is, \( E \)-equivalent. Thus \( \sigma'\theta \) is an \( E \)-unifier of \( L \) and \( M \) such that \( L\sigma'\theta = L\sigma' = L\sigma \), as desired. \( \square \)

Let’s consider two examples using \( \Sigma_2 \) (Chapter 2), which we repeat below.

\[
\Sigma_2 = \{ \text{political-party(arnold)} = \text{republican-party}, \\
\text{spouse(maria)} = \text{arnold}, \text{spouse(arnold)} = \text{maria}, \\
\text{bigger(arnold,maria), kennedy(maria)}, \\
\forall x \ \text{kennedy}(x) \rightarrow (\text{political-party}(x) = \text{democratic-party}), \\
\forall x \neg(\text{political-party}(x) = \text{political-party(spouse}(x))) \rightarrow \\
\text{diplomatic}(x) \land \text{diplomatic(spouse}(x)), \\
\forall x \neg(\text{political-party}(x) = \text{political-party(spouse}(x))) \land \\
\text{bigger(spouse}(x),x) \rightarrow \text{tough}(x) \}.
\]

Example 43

\[
\text{loves(arnold,spouse(arnold))} \geq_\Sigma \text{loves(spouse(maria),maria)}
\]

To verify this, we first notice that one \( E \)-transform of
\[ loves(\text{spouse}(\text{maria}),\text{maria}) \]

taking \( E \) to be \( \Sigma_2 \), is

\[ loves(\text{arnold},\text{spouse}(\text{arnold})) \]

Then the empty substitution maps the first atom in the example to this \( E \)-transform of the second atom.

Now consider a negative example.

**Example 44**

\[ taller(\text{lou},x) \nsubseteq^* taller(\text{arnold},\text{maria}) \]

To see this, generate all the \( E \)-transforms of the second atom, such as

\[ taller(\text{spouse}(\text{maria}),\text{spouse}(\text{arnold})) \]

and notice that there is no substitution from the first atom to any of these. In this particular example, it is enough in fact to generate all the \( E \)-transforms of the first argument, \( \text{arnold} \), and verify that none of these is \( \text{lou} \).

### 3.5 Orderings for E-Anti-Unification, Simple Constrained Anti-Unification, and Constrained E-Anti-Unification

Simple constrained anti-unification operates on *simple constrained atoms*, which meet the following restriction: the constraint is a conjunction of atoms built from constraint
predicates and from terms that appear in the head. In addition we require that the constraint theory not contain the equality predicate. (Extended sorted anti-unification is just the version of simple constrained anti-unification where all constraint predicates are unary. It is used only for analysis, in Chapter 5.) Because every term, and more specifically every variable, in the constraint also appears in the head, we can simplify the characterization in Theorem 29. Specifically, we can replace \( \Sigma \models \forall y(C_2 \rightarrow (\exists z C_1 \theta)) \) in Theorem 29 with \( \Sigma \models \forall(C_2 \rightarrow (C_1 \theta)) \).

Here are some examples of the \( \geq_{\Sigma} \) ordering on simple constrained atoms. The analysis of the examples uses the simplified characterization. Some of these examples use the constraint (sort) theory \( \Sigma_1 \), while others use a new constraint theory, \( \Sigma_5 \) (below).

\[
\Sigma_5 = \{ \text{BIGGER}(\text{son}(\text{jumbo}),\text{son}(\text{clyde})), \text{BIGGER}(\text{son}(\text{fred}),\text{son}(\text{joe})), \\
\text{BIGGER}(\text{jumbo},\text{clyde}), \text{BIGGER}(\text{fred},\text{joe}), \\
\text{ELEPHANT}(\text{clyde}), \text{ELEPHANT}(\text{jumbo}), \text{HUMAN}(\text{fred}), \text{HUMAN}(\text{joe}), \\
\forall x \forall y \text{ELEPHANT}(x) \land \text{HUMAN}(y) \rightarrow \text{BIGGER}(x,y) \}
\]

**Example 45**

\[
\text{loves}(x, \text{mom}(x))/\text{ELEPHANT}(x) \land \text{IN-CIRCUS}(x) \land \text{GRAY}(\text{mom}(x))
\]

\( \geq_{\Sigma_1} \text{loves}(\text{clyde}, \text{mom}(\text{clyde})) \).

For any substitution \( \theta \) that maps \( x \) to clyde,

\[
\text{loves}(x, \text{mom}(x))\theta = \text{loves}(\text{clyde}, \text{mom}(\text{clyde}))
\]

Note further that

\[
(\text{ELEPHANT}(x) \land \text{IN-CIRCUS}(x) \land \text{GRAY}(\text{mom}(x)))\theta = \\
\text{ELEPHANT}(\text{clyde}) \land \text{IN-CIRCUS}(\text{clyde}) \land \text{GRAY}(\text{mom}(\text{clyde}))
\]

We can easily verify that

\[
\Sigma_1 \models \text{ELEPHANT}(\text{clyde}) \land \text{IN-CIRCUS}(\text{clyde}) \land \text{GRAY}(\text{mom}(\text{clyde}))
\]
Example 46

\[
\text{chases}(x_1, y_1) / \text{ELEPHANT}(x_1) \land \text{BIGGER}(x_1, y_1)
\]
\[
\geq_{\Sigma_5} \text{chases}(x_2, y_2) / \text{ELEPHANT}(x_2) \land \text{HUMAN}(y_2).
\]

To verify this, let \( \theta \) be a substitution that maps \( x_1 \) to \( x_2 \) and \( y_1 \) to \( y_2 \). Observe that \( \theta \) maps \( \text{chases}(x_1, y_1) \) to \( \text{chases}(x_2, y_2) \) and that

\[
(\text{ELEPHANT}(x_1) \land \text{BIGGER}(x_1, y_1))\theta = \text{ELEPHANT}(x_2) \land \text{BIGGER}(x_2, y_2)
\]

Observe in addition that

\[
\Sigma_5 \models \forall x_2 \forall y_2 (\text{ELEPHANT}(x_2) \land \text{HUMAN}(y_2) \rightarrow \text{ELEPHANT}(x_2) \land \text{BIGGER}(x_2, y_2))
\]

Here are two negative examples of the \( \geq_{\Sigma} \) relation.

Example 47

\[
\text{loves}(x, \text{mom}(x)) / \text{ELEPHANT}(x) \land \text{IN-CIRCUS}(x)
\]
\[
\not\geq_{\Sigma_1} \text{loves}(x, \text{mom}(x)) / \text{MAMMAL}(x) \land \text{GRAY(mom(x))}.
\]

The relation fails to hold because

\[
\Sigma_1 \not\models \forall x((\text{MAMMAL}(x) \land \text{GRAY(mom(x)))} \rightarrow (\text{ELEPHANT}(x) \land \text{IN-CIRCUS}(x)))
\]

Example 48

\[
\text{intimidates}(\text{son}(x_1), \text{son}(y_1)) / \text{BIGGER}(\text{son}(x_1), \text{son}(y_1))
\]
\[
\not\geq_{\Sigma_1} \text{intimidates}(\text{son}(x_2), \text{son}(y_2)) / \text{BIGGER}(x_2, y_2)
\]

To see this, first note that any substitution \( \theta \) for which
intimidates(son(x_1), son(y_1)) \theta = intimidates(son(x_2), son(y_2))

must map \( x_1 \) to \( x_2 \) and \( y_1 \) to \( y_2 \). Second, notice that

\[
\Sigma_3 \not\models \forall x_2 \forall y_2 (\text{BIGGER}(x_2, y_2) \to \text{BIGGER}(\text{son}(x_2), \text{son}(y_2)))
\]

Constrained \( E \)-anti-unification also operates on simple constrained atoms, but it allows equality in the constraint theory \( \Sigma \) and requires that \( \Sigma \) be a Skolem Normal Form theory that has an initial model. Below are some examples of the \( \geq_\Sigma \) ordering on simple constrained atoms with respect to the constraint theory \( \Sigma_2 \), which uses the equality predicate, is in Skolem Normal Form, and has an initial model. We use the characterization in Theorem 33 in the analysis.

**Example 49**

\[
\textbf{loves}(\text{arnold}, x) / \textbf{TOUGH}(x) \geq_{\Sigma_2} \textbf{loves}(\text{spouse}(\text{maria}), \text{maria})
\]

To verify this, we first notice that one \( E \)-transform of \( \text{loves}(\text{spouse}(\text{maria}), \text{maria}) \), taking \( E \) to be \( \Sigma_2 \), is \( \text{loves}(\text{arnold}, \text{maria}) \). Then the substitution \( \theta = \{ x \mapsto \text{maria} \} \) maps the head of the first constrained atom to this \( E \)-transform. Moreover, \( \text{TOUGH}(x) \theta = \text{TOUGH}(\text{maria}) \), and \( \Sigma_2 \models \text{TOUGH}(\text{maria}) \), so the relation holds.

For a negative example, consider

**Example 50**

\[
\textbf{loves}(\text{arnold}, x) / \textbf{TOUGH}(x) \not\geq_{\Sigma_2} \textbf{loves}(\text{spouse}(\text{maria}), y) / \textbf{KENNEDY}(y)
\]

To see this, generate all the \( E \)-transforms (up to the renaming of variables) of the second atom. Then \( \text{loves}(\text{arnold}, y) / \text{KENNEDY}(y) \) is the only \( E \)-transform to which
some substitution maps $\text{loves}(\text{arnold}, x) / \text{Tough}(x)$. Any substitution $\theta$ that does so must map $x$ to $y$. But then $\Sigma_2 \not\models \overline{\nabla}(\text{Kennedy}(y) \rightarrow (\text{Tough}(x) \theta))$, which is to say, $\Sigma_2 \not\models \overline{\nabla}(\text{Kennedy}(y) \rightarrow (\text{Tough}(y)))$.

Finally, notice that the restrictions on a constraint theory for constrained $E$-anti-unification are precisely the restrictions required for the strong compactness property of the $\geq_{\Sigma}$ ordering. Therefore, Theorem 25 (Strong Compactness) and Theorem 33 together give the following more specific result for sets of simple constrained atoms.

**Corollary 51** Let $\phi_1 / C_1$, ..., $\phi_n / C_n$, and $\phi / C$ be simple constrained atoms. Let $\Sigma$ be a Skolem Normal Form constraint theory that has an initial model, and let $\phi / C$ be $\Sigma$-admissible. Then the following three statements are equivalent:

1. for some $1 \leq i \leq n$ we have: (1) $\phi_i / C_i \theta = \phi' / C$, for some substitution $\theta$ and some $E$-transform $\phi' / C$ of $\phi / C$, and (2) $\Sigma \models \overline{\nabla}(C \rightarrow (C_i \theta))$

2. $\overline{\nabla}(\phi_1 / C_1) \land \ldots \land \overline{\nabla}(\phi_n / C_n) \models_{\Sigma} \overline{\nabla}(\phi / C)$

3. $\exists(\phi / C) \models_{\Sigma} \exists(\phi_1 / C_1) \lor \ldots \lor \exists(\phi_n / C_n)$

This result proves useful in Chapter 7, when we consider properties of constrained $E$-anti-unification, and in Chapter 8, when we consider applications to inductive learning and speed-up learning. We end this chapter with one example of this property. We would like to know whether

**Example 52**

\[
\overline{\nabla}(\text{loves}(\text{maria}, \text{spouse}(x))) / \text{Kennedy}(x) \land \\
\overline{\nabla}(\text{loves}(\text{arnold}, x) / \text{Tough}(x)) \land \\
\overline{\nabla}(\text{loves}(\text{spouse}(\text{maria}), x) / \text{Diplomatic}(x)) \models_{\Sigma_2} \\
\text{loves}(\text{spouse}(\text{maria}), \text{maria})
\]

From Corollary 51, we know that this is true, since we have already seen that
loves(arnold, x) / tough(x) ≥ₚ loves(spouse(maria), maria)

In fact, we could also verify its truth using the third conjunct in the antecedent, instead of the second. Notice that the obvious method for showing that such an entailment does not hold is to show that none of the constrained atoms in the antecedent conjunction is Σ-more general than the constrained atom in the consequent.
Chapter 4

Algebraic and Combinatorial Properties of the Instantiation Ordering on Constrained Formulas

In Chapter 2 we developed an instantiation ordering, \( \Sigma \)-more general (\( \geq \Sigma \)), on constrained formulas. This ordering is in some sense the *right* instantiation ordering for constrained formulas, because it shares with the \( \geq \) ordering on ordinary formulas the semantic properties that we consider important. In Chapter 3 we examined various restrictions on the class of constrained formulas and constraint theories, and various special definitions or characterizations of the \( \geq \Sigma \) ordering for those classes. But because these special definitions gave us the same ordering, on smaller classes of constrained formulas, all of the general semantic properties of the \( \geq \Sigma \) ordering remained. Nevertheless, the algebraic and combinatorial properties do not necessarily remain when we restrict the class of formulas. In this chapter we study the basic algebraic and combinatorial properties of constrained formulas, ordinary formulas, sorted formulas, and simple constrained formulas ordered by \( \geq \Sigma \). (Notice that, from Theorem 29, the \( \geq \Sigma \) ordering is equivalent to the \( \geq \) ordering on ordinary formulas, provided \( \Sigma \) is empty. Therefore, we may think of \( \geq \) as denoting \( \geq \Sigma \) where \( \Sigma \) is empty.) One property, the number of minimal upper bounds, is closely tied to the computation of various anti-unification operations. Therefore the current chapter
presents a general discussion of this property but leaves more precise results for the chapters about the computation of anti-unification, which follow. Because the remainder of the dissertation deals with constrained \textit{atomic} formulas, this chapter focuses on atoms, constrained atoms, sorted atoms, and simple constrained atoms.

4.1 Definitions

We have stated that any quasi-ordering $\geq$ on a set $S$ may be viewed as a partial ordering on the set of equivalence classes $S'$ that $\geq$ induces on $S$. We now make this statement more precise. If $e$ is an element of $S$, $[e]_{\geq}$ denotes the equivalence class to which $e$ belongs under the $\geq$ ordering. If $e_1$ and $e_2$ are elements of $S$ then: $[e_1]_{\geq} \geq [e_2]_{\geq}$ if and only if $e_1 \geq e_2$. In this manner $\geq$ is a partial-ordering on the equivalence classes.

Many concepts and results exist regarding algebraic and combinatorial properties of quasi-ordered sets, or \textit{quasets}, and partially-ordered sets, or \textit{posets}. These concepts and results are useful in understanding the properties of the $\geq_\Sigma$ ordering. (In addition the $\models_\Sigma$ and $\preceq_\Sigma$ relations are also quasi-orderings, and they may also be viewed as partial orderings on the equivalence classes they induce.) Therefore we now provide a review of some basic definitions regarding posets [69]. Where possible, the definitions are provided, more generally, for quasets.

Let $\geq$ be any quasi-ordering on a set $S$. Two elements $x$ and $y$ of $S$ are \textit{comparable} with respect to $\geq$ if $x \geq y$ or $y \geq x$; otherwise, $x$ and $y$ are \textit{incomparable}. In the remaining definitions, and throughout the dissertation, we omit the phrase “with respect to $\geq$” if the ordering in use is obvious from the context. We also use the terms \textit{greater than} and \textit{less than} in the standard way. An element of $S$ is \textit{maximal} if no element of $S$ is greater than $x$, and \textit{minimal} if no element of $S$ is less than $x$. If $x \geq y$ but $y \not\geq x$, then we say $x$ is \textit{strictly greater than} $y$, or $x$ \textit{dominates} $y$, and we write $x > y$. The set of all elements of $S$ that dominate an element $x$ is called the \textit{dominating set} of $x$. If $x$ dominates $y$, and $x$ dominates no other element that dominates $y$, then we say that $x$ \textit{covers} $y$. The set of all elements in $S$ that cover an element $x$ is called the \textit{cover set} of $x$. 

66
A subquaset $P$ of $S$ is a subset of $S$ quasi-ordered by $\geq$. A quaset $P$, whose elements are ordered by $\geq'$, is isomorphic to a quaset $S$, whose elements are ordered by $\geq$, if there exists a bijection $h$ between the elements of $P$ and $S$ such that: $x \geq' y$ if and only if $h(x) \geq h(y)$ for any $x$ and $y$ in $P$. An embedding of $P$ in $S$ is a subquaset of $S$ that is isomorphic to $P$.

A chain in $S$ is a subquaset of $S$ that is totally ordered (pairwise ordered) by $>$ (this is often called a proper chain). A descending chain is a chain with an associated sequence on its members, such that one element is earlier in the sequence than another if and only if it is strictly greater. Similarly, an ascending chain is a chain with an associated sequence such that one element is earlier in the sequence than another if and only if it is strictly less. The size of a chain is the number of elements in the chain. The length of a chain is one less than the size of the chain, and the height of a poset $S$ is the length of the longest chain in $S$. We can also speak of the height or depth of an element $x$ in a poset $S$. The height of $x$ is the length of the longest chain that has $x$ as its maximal element; the depth of $x$ is the length of the longest chain that has $x$ as its minimal element. A chain is said to be between two elements $x$ and $y$ if $x$ is the maximal element and $y$ is the minimal element, or vice-versa. A chain is a maximal chain between $x$ and $y$ if no other elements can be added to the chain to yield a chain between $x$ and $y$.

An antichain is a subquaset of $S$ that is pairwise unordered by $\geq$. Note that if $S$ is a poset then the cover set of any element $x$ is an antichain. The width of a quaset $S$ is the size of the largest antichain in $S$.

A minimal upper bound, or mub, of a subset $T$ of $S$ (quasi-ordered by $\geq$) is any element $s \in S$ such that: (1) $s \geq s_i$ for all $s_i \in T$, and (2) for any other $s' \in S$, if $s \geq s'$ then for some $s_i \in T$ we have $s' \nleq s_i$. A maximal lower bound, or mlb, of a subset $T$ of $S$ is any element $s \in S$ such that: (1) $s_i \geq s$ for all $s_i \in T$, and (2) for any other $s' \in S$, if $s' \geq s$ then for some $s_i \in T$ we have $s_i \nleq s'$. If a subset of elements of $S$ has exactly one mub, the mub is said to be the least upper bound, or lub, of the set. If a set of elements of $S$ has exactly one mlb, the mlb is said to be the greatest lower bound, or glb, of the set. If every pair of elements in a poset has a glb, the poset is an upper semilattice. If
every pair of elements in a poset has an lub, the poset is a lower semilattice. A poset that is both an upper and lower semilattice is a lattice. In Section 4.3 we examine sufficient conditions for the existence of glbs and lubs.

For our purposes, a Boolean Algebra, $B_n$, is a poset whose elements are the subsets of $\{1, \ldots, n\}$, ordered by inclusion.\(^1\) We say that $n$ is the size of $B_n$. Note that a Boolean Algebra is a lattice whose greatest element is the set $\{1, \ldots, n\}$ and whose least element is the empty set. Notice also that the size of the largest chain in the Boolean Algebra $B_n$ is $n+1$ (the length is $n$, so the height of $B_n$ is $n$). For example, $\{1, 2, 3\} \geq \{2, 3\} \geq \{3\} \geq \{\} \geq$ is a largest chain in $B_3$.

We now define a concept from computational learning theory (which we use in Chapter 8) in the context of posets. The Vapnik-Chervonenkis Dimension, or VC-Dimension, of a poset $S$ is the size of the largest subset $A$ of $S$ such that: for each subset $P$ of $A$, some element of $S$ is greater than the members of $P$ and is not greater than any member of $A - P$.\(^2\) This definition of VC-Dimension is related to the standard definition, as used in computational learning theory, in the following way. The standard definition of VC-Dimension applies to a class $C$ of concepts, where each concept $c \in C$ classifies the members of a given instance space $X$ as positive or negative. The VC-Dimension of $C$ is the size of the largest subset $A$ of $X$ such that: for each subset $P$ of $A$, some concept in $C$ classifies every member of $P$ as positive and classifies every member of $A - P$ as negative. If the concept class $C$ and the instance space $X$ each consist of the members of a poset $S$ (sometimes called the single representation trick), such that one member of $S$ is greater than a second if and only if it classifies the second as positive, then the VC-Dimension of the poset $S$ is the same as the VC-Dimension of the concept class $C$.

We close this section with an observation about VC-Dimension as it applies to posets. First, if $S$ is an upper semi-lattice, the VC-Dimension of $S$ is exactly the size of the largest Boolean Algebra embedded in $S$.$^3$ Clearly if a Boolean Algebra of size $n$ is embedded

---

\(^1\)The traditional definition of Boolean Algebra is more general, but every Boolean Algebra is isomorphic to one of the form described here.

\(^2\) $A - P$ denotes the set difference, that is, the set of elements of $A$ that are not elements of $P$.

\(^3\)I thank Douglas West for a very helpful discussion that led to this observation.
in $S$ then the VC-Dimension of $S$ is at least $n$. If the VC-Dimension of $S$ were $m$ for $m > n$, then some subset $A$ of $S$ exists such that: $A$ has cardinality $m$, and for every subset $P_i$ of $A$ there exists an element in $S$ that is greater than each member of $P_i$ and greater than no member of $A - P_i$. Then, more specifically, the lub $p_i$ of the elements in $P_i$ is an element in $S$ that is greater than each member of $P_i$ and greater than no member of $A - P_i$. (Such lubs must exist since $S$ is an upper semi-lattice.) Then the subposet of $S$ consisting of the elements $p_i$ and the elements in $A$ is a Boolean Algebra of size $m > n$ embedded in $S$. This observation is valuable in Section 4.4. One result of the preceding observation is that the height of $S$ is an upper bound on the VC-Dimension of $S$, provided $S$ is an upper semi-lattice. This follows because the Boolean Algebra $B_n$ has height $n$ (has chains of length $n$).

### 4.2 Cover Sets, Chains, and Antichains

It is well-known that the set of equivalence classes of ordinary atoms has infinite descending chains and infinite antichains (even of equivalence classes of atoms with the same predicates). This implies the same for any restricted class of constrained atoms with respect to some constraint theories (such as the empty constraint theory), provided the class includes at least the ordinary atoms. Nevertheless, for some constraint theories, these properties do not hold. For example, if $\Sigma$ entails that all terms are equal, then the size of the largest chain of equivalence classes of constrained atoms is one, and the size of the largest antichain of equivalence classes of constrained atoms with the same predicate is one. (Nevertheless, with an infinite number of predicates, infinite antichains of atoms with different predicates still exist).

It is also well-known that the set of equivalence classes of ordinary atoms ordered by instantiation has no infinite ascending chain. Nevertheless, Theorem 53 (below) states that for some constraint theories, infinite ascending chains exist. It also states that in some cases infinite chains exist between two constrained atoms. In fact, it states these
results for the more restricted class of sorted atoms, which implies that the results hold for constrained atoms.

**Theorem 53** A constraint theory \( \Sigma \) exists such that equivalence classes of constrained atoms (in fact, even of the more restricted class of sorted atoms) have infinite ascending chains, even if \( \Sigma \) includes only one constraint predicate and the constraints use only that constraint predicate. In addition, such an infinite chain may exist beneath another equivalence class of sorted atoms, so an infinite chain may exist between two equivalence classes.

**Proof:** The motivation for the proof is as follows. An example of an infinite descending chain of equivalence classes of ordinary atoms, ordered by ordinary instantiation, is: 
\( \langle [p(x)]_{\geq 1}, [p(f(x))]_{\geq 1}, [p(f(f(x)))]_{\geq 1}, \ldots \rangle \). We now attach a monadic constraint predicate (or sort) \( \tau \) to \( x \) in each atom, and we design the sort theory \( \Sigma \) so that, roughly stated, adding \( f \)'s makes the (now sorted) atoms greater, or more general, rather than more specific. Let \( \Sigma \) be the following logical sentence: \( \forall x (\tau(x) \rightarrow (\exists y (\tau(y) \land (x = f(y)))) \rangle \). Then an infinite ascending chain of equivalence classes of sorted atoms is: 
\( \langle [p(x)/\tau(x)]_{\geq 1}, [p(f(x))/\tau(x)]_{\geq 1}, [p(f(f(x)))/\tau(x)]_{\geq 1}, \ldots \rangle \), or, writing the constrained atoms in variable-abstract form, 
\( \langle [p(y)/(y = x) \land \tau(x)]_{\geq 1}, [p(y)/(y = f(x)) \land \tau(x)]_{\geq 1}, [p(y)/(y = f(f(x)) \land \tau(x)]_{\geq 1}, \ldots \rangle \). Notice that every atom in the chain 
\( \langle [p(y)/(y = x) \land \tau(x)]_{\geq 1}, [p(y)/(y = f(x)) \land \tau(x)]_{\geq 1}, [p(y)/(y = f(f(x)) \land \tau(x)]_{\geq 1}, \ldots \rangle \) is a \( \Sigma \)-instance of \( p(x) \); hence we have an infinite chain between \( p(x) \) and \( p(x)/\tau(x) \).

Notice, in addition, that an infinite descending chain of equivalence classes of sorted atoms may exist above a given equivalence class. For example, consider the constraint theory

\[
\Sigma_F = \{ \forall x (\tau(x) \rightarrow \tau(f(x))), \forall x (\omega(x) \rightarrow \tau(x)), \\
\forall x (\omega(x) \rightarrow (\exists y (\tau(y) \land (x = f(y))))), \\
\forall y \forall y ((\omega(x) \land (x = f(y))) \rightarrow (x = f(f(y)))) \}
\]
Then \( \{[p(x)/\tau(x)]_{\Sigma_F}, [p(f(x))/\tau(x)]_{\Sigma_F}, [p(f(f(x)))/\tau(x)]_{\Sigma_F}, \ldots \} \) is an infinite descending chain, all of whose members are \( \Sigma_F \)-more general than \( [p(x)/\omega(x)]_{\Sigma} \).

So we see that with unrestricted equality in \( \Sigma \), “almost anything goes” regarding chains and antichains (the same occurs when we allow equality in the constraints but not \( \Sigma \)). Therefore, throughout the rest of this section, we restrict ourselves to the case where \( \Sigma \) and the constraints do not contain equality. Thus we may use exclusively the characterization of \( \geq_\Sigma \) given in Theorem 29. Later, in Chapter 7, we return to a brief consideration of these properties where \( \Sigma \) has a particular kind of very limited equality.

The following observation is useful in studying chains of atoms and constrained atoms in the absence of equality: a substitution can never decrease the size of an atom, since it replaces variables (size 1) with terms (size 1 or greater). Because \( \phi_1 \geq \phi_2 \) only if there exists a substitution \( \theta \) such that \( \phi_1 \theta = \phi_2 \), we know that if \( \phi_1 \geq \phi_2 \) then the size of \( \phi_1 \) can be no greater than the size of \( \phi_2 \). Similarly for constrained atoms (without equality in \( \Sigma \) or the constraints), because \( \phi_1/C_1 \geq_\Sigma \phi_2/C_2 \), where neither \( \Sigma \) nor the constraints contains equality, only if there exists a substitution \( \theta \) such that \( \phi_1 \theta = \phi_2 \), we know that if \( \phi_1/C_1 \geq_\Sigma \phi_2/C_2 \) then the size of \( \phi_1 \) (the head size of \( \phi_1/C_2 \)) can be no greater than the size of \( \phi_2 \) (the head size of \( \phi_2/C_2 \)). Notice also that if \( \Sigma \) and the constraints do not contain equality, the heads of constrained atoms in the same equivalence class according to \( \geq_\Sigma \) must be variants and therefore must have the same size. Therefore, for any chain of equivalence classes of ordinary atoms, the size of each atom in each equivalence class is at most the size of the atoms in the minimal class in the chain. And for any chain of equivalence classes of constrained atoms, where neither \( \Sigma \) nor the constraints contains equality, the head size of each constrained atom in each equivalence class is at most the head size of the constrained atoms in the minimal class in the chain.

We begin with ordinary atoms and empty \( \Sigma \). Then we extend the techniques used for ordinary atoms to apply to simple constrained atoms.
4.2.1 Cover Sets and Chains of Ordinary Atoms

**Theorem 54** Let $\phi$ and $\phi'$ be any pair of atoms such that $\phi > \phi'$, and let $n$ be the size of $\phi'$. Then the largest chain of atoms between $\phi$ and $\phi'$ has size at most $n + 1$.

*Proof:* We associate a parameter $n$-height with each atom that is more general than $\phi'$. Where the size of an atom is $k$, and the number of distinct variables (not variable occurrences) in the atom is $m$, the $n$-height of the atom is $(n - k) + m$. Notice that applying a renaming substitution to an atom does not change the $n$-height. Notice also that applying any other substitution to an atom reduces the $n$-height of the atom, since it must increase $k$ more than it can increase $m$, and it actually may decrease $m$. Therefore, the atoms in a chain between $\phi$ and $\phi'$ must have distinct $n$-heights less than the $n$-height of $\phi$ and greater than the $n$-height of $\phi'$. The $n$-height of $\phi$ is at most $n$, while the $n$-height of $\phi'$ is at least 0. Therefore, the size of any chain of atoms between $\phi$ and $\phi'$ is at most $n + 1$. □

**Corollary 55** The size of the largest chain of ordinary atoms, each atom of size at most $n$, is $n + 1$.

*Proof:* The largest such chain has some atom $\phi$ as its maximal element and some atom $\phi'$, of size $n$, as its minimal element. Therefore, the result follows from Theorem 54. □

By the way we have defined equivalence classes of atoms, a largest chain of equivalence classes of atoms corresponds to a largest chain of atoms. (Take one atom from each equivalence class; conversely, take the equivalence class of each atom.) Therefore, Corollary 55 implies that the poset of equivalence classes of ordinary atoms of size at most $n$ has height $n$.

**Corollary 56** Let $[\phi]_\geq$ be any equivalence class of ordinary atoms that have size $n$. Then the depth of $[\phi]_\geq$ is $n + 1$. In other words, the size of the largest (ascending) chain of equivalence classes of ordinary atoms with $[\phi]_\geq$ as the minimal element is $n + 1$.

*Proof:* Because of the correspondence between chains of ordinary atoms and chains of equivalence classes of ordinary atoms, we need only show that the size of the largest
chain of atoms with \( \phi \) as the minimal element is at most \( n + 1 \). But this follows from Theorem 54. \( \square \)

**Corollary 57**  There exists no infinite descending chain of equivalence classes of atoms whose members are all \( \Sigma \)-more general than some class \( [\phi]_\geq \).

*Proof:* If such an infinite descending chain existed, then an infinite chain of atoms, each atom of size at most \( n \), would exist where \( n \) is the size of \( \phi \). This would contradict Corollary 55. \( \square \)

We will find it helpful to characterize the cover sets of equivalence classes of ordinary atoms. Theorem 58, below, does so.

**Theorem 58** One equivalence class of ordinary atoms, \([\phi_1]_\geq\), covers another, \([\phi_2]_\geq\), if and only if there exist substitutions \( \theta_1 \) and \( \theta_2 \) such that:

- \( \phi_1 \theta_1 \theta_2 = \phi_2 \)
- \( \theta_1 = \{ x \mapsto t \} \), where \( x \) is a variable in \( \phi_1 \), and \( t \) is either (A) another variable in \( \phi_1 \) or (B) a term \( f(z_1, \ldots, z_n) \) built from an \( n \)-ary function \( f \), \( n \geq 0 \), and variables \( z_1, \ldots, z_n \) that do not occur in \( \phi_1 \)
- \( \theta_2 \) is a (possibly empty) renaming substitution for \( \phi_1 \theta_1 \).

*Proof:* If: Because \( \phi_1 \theta_1 \theta_2 = \phi_2 \), by definition we have \( [\phi_1]_\geq \geq [\phi_2]_\geq \). From the specification of \( \theta_1 \), \( \theta_1 \cdot \theta_2 \) is not a renaming substitution, so in fact we have \( [\phi_1]_\geq > [\phi_2]_\geq \). Now suppose there exists some \( [\phi]_\geq \) such that \( [\phi_1]_\geq > [\phi]_\geq > [\phi_2]_\geq \). Then there exists a substitution \( \sigma \), which is not a renaming substitution, such that \( \phi_1 \sigma = \phi \). Notice that since \( [\phi]_\geq > [\phi_2]_\geq \), \( \sigma \) can map \( x \) only to itself or a variable that does not appear in \( \phi_1 \). Furthermore, because \( [\phi_1]_\geq > [\phi]_\geq \), \( \sigma \) cannot map another variable in \( \phi_1 \) to \( x \). Then \( \sigma \) must map some variable \( y \), other than \( x \), in \( \phi_1 \) to a term \( s \) that either is built from some function symbol or is another variable, other than \( x \), in \( \phi_1 \). But then there is no substitution from \( \phi \) to \( \phi_2 \), so \( [\phi]_\geq \neq [\phi_2]_\geq \).
Only if: If \([\phi_1] \geq \) covers \([\phi_2] \geq \) then there exists some substitution \(\theta\) such that \(\phi_1 \theta = \phi_2\). Without loss of generality, we may assume that \(\text{DOM}(\theta) \subseteq \text{VARS}(\phi_1)\). Let \(\theta_1\) be a substitution such that for any variable \(x\): if \(\theta\) maps \(x\) to a term \(t\), where \(t\) is a non-variable term or is another variable that occurs in \(\phi_1\), then \(\theta_1\) maps \(x\) to \(t\); otherwise, \(\theta_1\) maps \(x\) to itself. Let \(\theta_2\) be a substitution such that for any variable \(x\): if \(\theta\) maps \(x\) to a variable \(y\) that does not appear in \(\phi_1\), then \(\theta_2\) maps \(x\) to \(y\); otherwise, \(\theta_2\) maps \(x\) to itself. Then \(\theta_1 \cdot \theta_2 = \theta\), and \(\theta_2\) is a renaming substitution. If \(\theta_1\) maps two or more variables, say \(x_1\) and \(x_2\), to terms other than themselves, say \(t_1\) and \(t_2\) respectively, then \([\phi_1] \geq \) > \([\phi] \geq \) > \([\phi_2] \geq \), where \(\phi\) is \(\phi_1\{x_1 \mapsto t_1\}\). In this case, \([\phi_1] \geq \) does not cover \([\phi_2] \geq \), which is a contradiction. Similarly, if \(\theta_1\) maps some variable \(x\) to a term \(f(t_1, \ldots, t_n)\), where the terms \(t_i\) (\(1 \leq i \leq n\)) are not distinct variables not appearing in \(\phi_1\), then \([\phi_1] \geq \) > \([\phi] \geq \) > \([\phi_2] \geq \), where \(\phi\) is \(\phi_1\{x \mapsto f(z_1, \ldots, z_n)\}\) and \(z_1, \ldots, z_n\) are distinct variables that do not appear in \(\phi_1\). Then again \([\phi_1] \geq \) does not cover \([\phi_2] \geq \). Therefore \(\theta_1\) has the form \(\{x \mapsto t\}\), where \(x\) is a variable in \(\phi_1\), and \(t\) is either \((A)\) another variable in \(\phi_1\) or \((B)\) a term \(f(z_1, \ldots, z_n)\) built from an \(n\)-ary function \(f\), \(n \geq 0\), and variables \(z_1, \ldots, z_n\) that do not occur in \(\phi_1\). \(\square\)

For ordinary atoms, based on Theorem 58 the members of the cover set of an equivalence class of atoms \([\phi] \geq \) are the classes of the form \([\phi_i] \geq \) where \(\phi_i\) is obtained from \(\phi\) by one of:

- replacing all occurrences in \(\phi\) of some term \(f(x_1, \ldots, x_n)\), where \(f\) is an \(n\)-ary function symbol \((n \geq 0)\) and \(x_1, \ldots, x_n\) are distinct variables not occurring outside these term occurrences in \(\phi\), by a variable \(x\) that does not appear in \(\phi\)

- rewriting some nonempty proper subset of the occurrences of some variable \(y\) in \(\phi\) to some variable \(y'\) not in \(\phi\)

It follows that every equivalence class of ordinary atoms has a finite, nonempty cover set. It is worth noting that the size of this cover set may be exponential in the size of the atoms in the equivalence class. For example, consider the class \([\phi] \geq \), where \(\phi\) is built from an \(n\)-ary predicate \(p\) and \(n\) occurrences of some variable \(x\). The atoms in this class all have size
n, and the cover set consists of $\frac{1}{4}(2^n) - 1$ classes: one member of the cover set corresponds to each complementary pair of nonempty proper subsets of the occurrences of x. For example, for $n = 4$ the cover set of $[p(x, x, x, y)] \geq [p(x, y, x, y)] \geq [p(x, y, y, x)] \geq [p(\{x, y, x, y\})]$, and the cover set of any equivalence class of ordinary atoms is finite, we have the following observation.

**Observation 2** The dominating set of any equivalence class of ordinary atoms is finite.

Notice that the mubs of a set $E$ (finite or infinite) of equivalence classes of ordinary atoms are each more general than every class in $E$, and hence in the dominating set of each member of $E$. Therefore, we have the following corollary, which has been proven by Lassez, Maher, and Marriott [39].

**Corollary 59 (Lassez, Maher, and Marriott)** Any set (finite or infinite) of ordinary atoms has only finitely many mubs.

Of course, if $E$ is finite, it has a single mub (an lub), which is $[\phi] \geq$, where $\phi$ is the LGG of a set of atoms consisting of one atom from each class in $E$.

### 4.2.2 Cover Sets and Chains of Constrained Atoms without Equality

We now consider the sizes of chains with respect to any given constraint theory $\Sigma$ without equality. Of course, for any such theory the largest chain is infinite, provided the language allows at least one non-nullary function. But what about infinite ascending chains, or chains between two constrained atoms? Certainly if the constraint theory is infinite then infinite ascending chains can exist. In fact, even if $\Sigma$ is finite, and neither $\Sigma$ nor the constraints contains equality, infinite ascending chains can exist. For a simple example, let $\Sigma_C$ be $\{\forall x \ (c(x) \rightarrow c(f(x)))\}$, where $c$ is a constraint predicate. Then the following is
an infinite ascending chain: \( \langle p(x)/c(x), p(x)/c(f(x)), p(x)/c(f(f(x))), \ldots \rangle \). Moreover, this chain is beneath \( p(x) \), so adding \( p(x) \) yields an infinite chain between two constrained atoms. To eliminate infinite ascending chains such as this, we might require that \( \Sigma \) have no infinite chains of constraints under the \( \prec_\Sigma \) ordering, but no theory \( \Sigma \) without equality meets this restriction. For example, if we conjoin \( \Sigma_C \) to each of the constraints \( c(x), c(f(x)), \ldots \) in the preceding example, we have an infinite chain of constraints in the \( \prec_\Sigma \) ordering even where \( \Sigma \) is empty. Thus even with an empty constraint theory, there exist infinite ascending chains of equivalence classes of constrained atoms, and infinite chains between two equivalence classes of constrained atoms. Therefore, we now restrict our attention to sorted atoms and simple constrained atoms, where \( \Sigma \) uses only finitely-many constraints. Of course, infinite descending chains still exist. Therefore, we further restrict our attention to ascending chains and chains between any two simple constrained atoms. We address these questions in the following section.

Before continuing to the following section, we note that cover sets for constrained atoms have very unrestricted qualities as well, even if \( \Sigma \) contains no equality and is finite. Oddly enough, some equivalence classes of constrained atoms may have an empty cover set, even if there exist equivalence classes that are \( \Sigma \)-more general. This may be the case even if \( \Sigma \) is finite (and so uses a finite set of constraint predicates) and contains no equality, and the constraints of constrained formulas contain no equality. For example, consider the constraint theory

\[
\Sigma_H = \{ \forall z \ (c(a, z) \rightarrow c(f(a), z)), \\
\forall x \forall y \forall z \ ((c(x, z) \rightarrow c(f(y), z)) \rightarrow (c(x, z) \rightarrow c(f(f(y)), z))) \}.
\]

Many equivalence classes are \( \Sigma_H \)-more general than the class \( [p(x)/c(a, x)]_{\Sigma_H} \), but none covers it. Any class \( [p(x)/C]_{\Sigma_H} \), where \( C \) is a finite conjunction of atoms in the set \( S = \{ c(f(a), x), c(f(f(a)), x), c(f(f(f(a))), x), \ldots \} \), is strictly \( \Sigma_H \)-more general than \( [p(x)/c(a, x)]_{\Sigma_H} \). But such a class does not cover \( [p(x)/c(a, x)]_{\Sigma_H} \), because for each such class there exists another class \( [p(x)/C']_{\Sigma_H} \) such that \( [p(x)/C]_{\Sigma_H} \succ_\Sigma \ [p(x)/C']_{\Sigma_H} \succ_\Sigma [p(x)/c(a, x)]_{\Sigma_H} \). To see this, let \( C' \) contain every atom in \( C \) plus at least one other
atom in \( S \). Therefore \([p(x)/C]_{\Sigma_H} \) does not cover \([p(x)/c(a, x)]_{\Sigma_H} \). The culprit—the reason the cover set is empty—in this case is an infinite descending chain whose members are all above, or greater than, \([p(x)/c(a, x)]_{\Sigma_H} \).

For other choices of \( \Sigma \) (again finite and equality-free), some constrained atoms may have infinite cover sets, again even if constraints are equality-free. Let \( \Sigma_I \) be the result of adding the following sentences to \( \Sigma_H \):

\[
\{ c'(f(x), f(f(x))), \forall x \forall y \ (c'(x, y) \rightarrow c'(f(x), f(y))), \forall x \forall y \ (c'(x, y) \rightarrow (\forall z \ (\neg c(x, z) \lor \neg c(y, z)))) \}
\]

The effect of this addition is that any constraint that is a conjunction of two or more atoms from the set \( S \) is not \( \Sigma_I \)-satisfiable. As a result, each equivalence class \([p(x)/C]_{\Sigma_I} \), where \( C \) is one atom from \( S \), covers the class \([p(x)/c(a, x)]_{\Sigma_I} \).

So we see that there are no bounds, upper or lower, on the size of a cover set for equivalence classes of constrained atoms. This is the case even if constraints are equality-free and \( \Sigma \) is finite and equality-free.

Nevertheless, we can partially characterize the cover sets of constrained atoms. Specifically, we can extend Theorem 58 somewhat to apply to the class of constrained atoms. The extension retains only the only if direction of the theorem, but this direction helps us to place bounds on chain sizes in some restricted cases, as we will see afterward. The new theorem is based on the characterization of the cover relation for equivalence classes of ordinary atoms, given in Theorem 58, and on the \( \preceq_{\Sigma} \) ordering on constraints. Recall that for constraints \( C_1 \) and \( C_2 \): \( C_1 \preceq_{\Sigma} C_2 \) if and only if \( \Sigma \models \forall (C_1 \rightarrow C_2) \). If \( C_1 \preceq_{\Sigma} C_2 \) but \( C_2 \npreceq_{\Sigma} C_1 \), then we write \( C_1 \prec_{\Sigma} C_2 \), and we say that \( C_1 \) is tighter than \( C_2 \) with respect to \( \Sigma \). We also say that \( C_2 \) is weaker than \( C_1 \) with respect to \( \Sigma \). (We omit the phrase “with respect to \( \Sigma \)” if the choice of \( \Sigma \) is clear from the context.) If \( C_1 \prec_{\Sigma} C_2 \) and there is no constraint \( C \) such that \( C_1 \prec_{\Sigma} C \prec_{\Sigma} C_2 \), then we say that \( C_1 \) is minimally tighter than \( C_2 \). We also say that \( C_2 \) is minimally weaker than \( C_1 \). In order to avoid explicitly representing existential quantifiers on variables in the constraint of a constrained atom that do not appear in the head, with the notation \( \exists y^* C \) as used in previous chapters, we
will simply assume that the existential quantifiers have been supplied already. We may do so because we now are interested in only the $\geq_{\Sigma}$ ordering, rather than $\Sigma$-entailment as well, and this assumption does not alter the ordering. Therefore, for any constrained atom $\phi/C$, we may assume without loss of generality that every variable with a free occurrence in $C$ appears in $\phi$.

It is worth noting that for some constraints, with respect to some constraint theories, there exist no minimally tighter constraints even though there exist tighter constraints. For example, suppose $\Sigma_J = \{\forall x \ (\tau(f(x)) \rightarrow \tau(x), \forall x \ (\tau(a) \rightarrow \omega(a))\}$. Then no constraint is minimally tighter than $\omega(a)$ with respect to $\Sigma_J$: $\tau(f(a))$ is tighter than $\tau(a)$, $\tau(f(f(a)))$ is tighter than $\tau(f(a))$, etc., and each is tighter than $\omega(a)$. Similarly, for some constraints there exist no minimally weaker constraints even though there exist weaker constraints. For example, again let $\Sigma_H$ be

$$
\Sigma_H = \{\forall z \ (c(a, z) \rightarrow c(f(a), z)),
\forall x \forall y \forall z \ ((c(x, z) \rightarrow c(f(y), z)) \rightarrow (c(x, z) \rightarrow c(f(f(y)), z)))\}.
$$

Recall that no constraint is minimally weaker than $c(a, x)$ with respect to $\Sigma_H$. The following is an infinite chain of constraints, such that each constraint is weaker than $c(a, x)$ and each is tighter than the previous ones in the chain: $\langle c(f(a), x), c(f(a), x) \land c(f(f(a)), x), c(f(a), x) \land c(f(f(a)), x) \land c(f(f(f(a))), x), \ldots \rangle$.

**Theorem 60** One equivalence class of constrained atoms, $[\phi_1/C_1]_{\Sigma}$, covers another, $[\phi_2/C_2]_{\Sigma}$, only if either (1) $[\phi_1/C_1]_{\Sigma}$ and $[\phi_2/C_2]_{\Sigma}$ contain constrained atoms $\phi/C_1'$ and $\phi/C_2'$, respectively, where $C_2'$ is minimally tighter than $C_1'$ with respect to $\Sigma$, or (2) $[\phi_1]_{\geq}$ covers $[\phi_2]_{\geq}$, and a substitution $\theta$ exists such that $\phi_1\theta = \phi_2$, $\text{DOM}(\theta) \subseteq \text{VAR}(\phi_1)$, and $C_1\theta$ is $\Sigma$-equivalent to $C_2$.

**Proof:** Because $[\phi_1/C_1]_{\Sigma}$ covers $[\phi_2/C_2]_{\Sigma}$, there exists a substitution $\theta$ such that $\text{DOM}(\theta) \subseteq \text{VAR}(\phi_1)$ and $\phi_1\theta = \phi_2$. If $\theta$ is a renaming substitution, then $[\phi_1/C_1]_{\geq}$ contains a constrained atom $\phi_2/C_1'$ such that $\phi_2/C_1' \geq_{\Sigma} \phi_2/C_2$. Then since $[\phi_1/C_1]_{\geq}$ covers $[\phi_2/C_2]_{\geq}$, $C_2$ must be minimally tighter than $C_1'$ with respect to $\Sigma$. 78
If $\theta$ is not a renaming substitution, we first show that $[\phi_1]_\Sigma \succ [\phi_2]_\Sigma$. To see this, notice that since $[\phi_1]_\Sigma > [\phi_2]_\Sigma$, if $[\phi_1]_\Sigma$ does not cover $[\phi_2]_\Sigma$ then there exist some non-renaming substitutions $\theta_1$ and $\theta_2$ such that $\phi_1 > \phi_1 \theta_1 > \phi_1 \theta_1 \theta_2 = \phi_2$ and $\text{DOM}(\theta_1 \cdot \theta_2) = \text{DOM}(\theta)$. Then $\theta_1 \cdot \theta_2 = \theta$. Therefore $\phi_1 / C_1 \succ [\phi_1 / C_1]_\Sigma (\phi_1 / C_1) \theta_1$ and $\phi_1 / C_1 \succ (\phi_1 / C_1) \theta_1 \theta_2 \succ [\phi_2 / C_2]_\Sigma \succ [\phi_1 / C_1]_\Sigma$ does not cover $[\phi_2 / C_2]_\Sigma$, which contradicts our assumption. Thus we know that $[\phi_1]_\Sigma$ covers $[\phi_2]_\Sigma$. We now show that $C_1 \theta$ and $C_2$ are $\Sigma$-equivalent. Because $[\phi_1 / C_1]_\Sigma \succ [\phi_1 / C_2]_\Sigma$, and $\phi_1 \theta = \phi_2$, we know that $C_2 \preceq \Sigma C_1 \theta$. If $C_1 \theta$ and $C_2$ are not $\Sigma$-equivalent, then because $C_2 \preceq \Sigma C_1 \theta$, we have more specifically that $C_2 \prec \Sigma C_1 \theta$. Then $\phi_1 / C_1 \succ \phi_2 / C_1 \theta \succ \phi_2 / C_2$, so $[\phi_1 / C_1]_\Sigma$ does not cover $[\phi_2 / C_2]_\Sigma$, which contradicts our assumption. \hfill $\Box$

To see that the If direction of Theorem 60 would not hold, let $\Sigma_K$ be

$$\Sigma_K = \{ \tau(a), \tau'(a), \tau'(b), \forall x (\tau(x) \to \tau'(x)) \}$$

Notice that $[p(x)]_\Sigma$ covers $[p(a)]_\Sigma$, $\theta = \{ x \mapsto a \}$ maps $p(x)$ to $p(a)$, and $\tau'(x) \theta$ is $\Sigma_K$-equivalent to $\tau'(a)$. Therefore, if the If direction held, by condition (2) we would conclude that $[p(x)/\tau'(x)]_\Sigma \succ [p(a)/\tau'(a)]_\Sigma$. But this conclusion is wrong since $p(x)/\tau'(x) \succ [p(x)/\tau(x)]_\Sigma \succ [p(a)/\tau'(a)]_\Sigma$.

The partial characterization in Theorem 60 is useful in the following two sections, when we consider chains of simple constrained atoms and sorted atoms.

### 4.2.2.1 Ascending Chains of Simple Constrained Atoms

With simple constrained atoms, for a given head $\phi$ some constraints are disallowed. Specifically, any constraint that is not a conjunction of atoms whose terms appear in $\phi$ is disallowed. Therefore, in the context of simple constrained atoms, when we say that a constraint $C_2$ is minimally tighter than a constraint $C_1$ for $\phi$ (with respect to a given constraint theory $\Sigma$), we mean that (1) $C_1$ and $C_2$ are conjunctions of atoms built from terms that appear in $\phi$, (2) $C_2 \prec \Sigma C_1$, and (3) for no other constraint $C$ that is a conjunction of atoms built from terms in $\phi$ do we have $C_1 \prec \Sigma C \prec \Sigma C_2$. We say that a constraint $C'$ for $\phi$ is in expanded form if $C$ contains every atom $\alpha$, built from a constraint
predicate and terms in \( \phi \), such that \( C \Sigma \)-entails \( \alpha \). If \( C \) is a constraint for \( \phi \), and \( C' \) is the result of conjoining with \( C \) every such atom \( \alpha \), then we say that \( C' \) is the expanded form of \( C \). Obviously, any constraint is \( \Sigma \)-equivalent to its expanded form. Notice that for any simple constrained atom \( \phi/C' \) and any constraint theory \( \Sigma \) that uses only a finite set of constraint predicates, \( C \) has a finite expanded form. Specifically, \( C \) can \( \Sigma \)-entail an atom \( \alpha \) only if the predicate from which \( \alpha \) is built appears in \( \Sigma \). If \( K \) is the number of constraint predicates in \( \Sigma \), \( r \) is the maximum of the arities of these predicates, and \( n \) is the size of \( \phi \), then at most \( Kn^r \) atoms can be built from predicates that appear in \( \Sigma \) and terms that appear in \( \phi \). If \( L \) is the number of constraint predicates in \( C \) but not in \( \Sigma \), and \( q \) is the maximum of the arities of these predicates, then at most \( Ln^q \) atoms can be built from predicates that appear in \( C \), but not \( \Sigma \), and terms that appear in \( \phi \).

In contrast with arbitrary constraints, for the constraint of any simple constrained atom there exist minimally tighter constraints, provided \( \Sigma \) uses only a finite set of constraint predicates. This is most easily seen if constraints are assumed to be in expanded form. To obtain a constraint \( C' \) that is minimally tighter than some constraint \( C \) on a simple constrained atom \( \phi \), conjoin to \( C \) either (1) an atom built from a constraint predicate that does not appear in \( \Sigma \) and from terms in \( \phi \), or (2) a set \( S \) of atoms built from a constraint predicates in \( \Sigma \) and terms in \( \phi \) such that: the atoms of \( S \) are not already in \( C \), \( C' \) is in expanded form, and no subset \( S' \) of \( S \) yields a constraint in expanded form when conjoined to \( C \). Notice, therefore, that \( C' \) contains more atoms than does \( C \). Under the same provision, that \( \Sigma \) uses only a finite set of constraint predicates, if weaker constraints exist for any given simple constrained atom then minimally weaker constraints exist. For any simple constrained atom \( \phi/C \), simply remove from \( C \) any subset \( S \) of atoms such that: the resulting constraint is in expanded form, and there exists no subset \( S' \) of \( S \) such that removing \( S' \) from \( C \) yields a constraint in expanded form. Notice from these arguments that for any simple constrained atom, \( \phi/C \), the number of constraints that are minimally tighter than \( C \) may be infinite, but the number of constraints that are minimally weaker than \( C \) is finite.

The preceding discussion and Theorem 54 together yield the following simple lemma.
Lemma 61 Let $\Sigma$ be a constraint theory that contains no equality and uses a finite set of constraint predicates. Then the dominating set of any equivalence class of simple constrained atoms $[\phi/C]_{\Sigma}$ is finite.

Proof: Let $K$ be the number of constraint predicates in $\Sigma$, and let $r$ be the maximum of the arities of these predicates. Let $L$ be the number of constraint predicates that appear in $C$ but not in $\Sigma$, and let $q$ be the maximum of the arities of these predicates. Let $n$ be the size of $\phi$. By Theorem 29, if some equivalence class of simple constrained atoms $[\phi'/C']_{\Sigma}$ is $\Sigma$-more general than $[\phi/C]_{\Sigma}$, then there exists a substitution $\theta$ such that $\phi'\theta = \phi$ and $C$ is tighter than $C'\theta$ with respect to $\Sigma$. Notice first that, from the discussion in the previous section, there are only finitely many choices of $\phi'$, up to the renaming of variables, such that $\phi' \geq \phi$, and each such $\phi'$ has size at most $n$. Furthermore, for any choice of $\phi'$, if $C$ is tighter than $C'\theta$ then $C'\theta$, and so $C'$, must be built from only constraint predicates in $\Sigma$ or $C$. But there are at most $Kn^r + Ln^3$ atoms that can be built from these constraint predicates and terms in $\phi'$, so there are only finitely many choices for $C'$, given the choice of $\phi'$. Therefore, there are only finitely many possible simple constrained atoms $\phi'/C'$, up to the renaming of variables, that are $\Sigma$-more general than $\phi/C$; hence, the set of equivalence classes $[\phi'/C']_{\Sigma}$ that dominate $[\phi/C]_{\Sigma}$ is finite.

The following corollaries to this lemma are immediate.

Corollary 62 Let $\Sigma$ be a constraint theory without equality that uses a finite set of constraint predicates. Then there exists no infinite chain of equivalence classes of simple constrained atoms, ordered by $\geq_{\Sigma}$, between any pair of such equivalence classes, and there exists no infinite ascending chain of equivalence classes of simple constrained atoms.

Corollary 63 Let $\Sigma$ be a constraint theory without equality, that uses only a finite set of constraint predicates. Let $\phi_1/C_1$ be a simple constrained atom that is strictly $\Sigma$-more general than $\phi_2/C_2$. Then any maximal chain between $\phi_1/C_1$ and $\phi_2/C_2$ includes exactly one simple constrained atom covered by $\phi_1/C_1$ and exactly one simple constrained atom that covers $\phi_2/C_2$. 

81
Notice that in general this statement is not true for equivalence classes of arbitrary constrained atoms. Take the example given in the previous section involving the constraint theory $\Sigma_H$. We saw an infinite maximal chain between $[p(x)]_{\geq \Sigma}$ and $[p(x)/c(a, x)]_{\geq \Sigma}$, but the cover set of $[p(x)/c(a, x)]_{\geq \Sigma}$ is empty. Hence that chain includes no class that covers $[p(x)/c(a, x)]_{\geq \Sigma}$. We have seen similar examples where a maximal chain between two equivalence classes does not include a class covered by the maximal class in the chain.

**Corollary 64** Let $\Sigma$ be a constraint theory, without equality, that uses a finite set of constraint predicates. Then there exists no infinite descending chain of equivalence classes of simple constrained atoms whose members are all $\Sigma$-more general than some class $[\phi/C]_{\geq \Sigma}$.

**Corollary 65** Let $\Sigma$ be a constraint theory without equality that uses a finite set of constraint predicates. Then the cover set of any equivalence class of simple constrained atoms is finite.

**Corollary 66** Let $\Sigma$ be a constraint theory without equality, that uses a finite set of constraint predicates. Any set (finite or infinite) of equivalence classes, under $\geq \Sigma$, of simple constrained atoms has only finitely many mubs.

The remainder of this section develops a more precise bound on the sizes of chains. In order to precisely count the sizes of chains between two constrained atoms, which will provide a bound on the sizes of ascending chains as well, it is helpful to understand the cover sets of simple constrained atoms. Observe from Theorem 60 that for simple constrained atoms $\phi_1/C_1$ and $\phi_2/C_2$, $\phi_1/C_1$ covers $\phi_2/C_2$ only if exactly one of the following is true (where $C_1$ and $C_2$ are assumed to be in expanded form):\(^4\)

1. $\phi_1$ and $\phi_2$ are variants, where $\phi_1\theta = \phi_2$, and $C_2$ is minimally tighter than $C_1\theta$

2. $\phi_2/C_2 = (\phi_1/C_1)\theta$, where $\theta$ maps one variable that appears in $\phi_1$ to another variable in $\phi_1$ and maps every other variable to itself or a distinct variable that does not appear in $\phi_1$

\(^4\)In items 2 and 3, applying the substitution $\theta$ to $C_1$ may lead to duplicate atoms. (In item 3, this can occur only if $f$ is a constant.) Duplicate atoms are removed, as is standard with conjunction.
3. \( \phi_2/C_2 = (\phi_1/C_1)\theta \), where \( \theta \) maps one variable in \( \phi_1 \) to a term \( f(x_1, \ldots, x_i) \), where \( f \) is an \( i \)-ary function and \( x_1, \ldots, x_i \) are distinct variables that do not appear in \( \phi_1 \), and maps every other variable to itself or a distinct variable that does not appear in \( \phi_1 \).

**Theorem 67** Let \( \Sigma \) be a constraint theory, without equality, that uses only a finite set of constraint predicates. Let \( K \) be the number of constraint predicates in \( \Sigma \), and let \( r \) be the maximum of the arities of these constraint predicates. Let \( \phi'/C' \) be any simple constrained atom, let \( n \) be the size of \( \phi' \), let \( L \) be the number of constraint predicates that appear in \( C' \) but not \( \Sigma \), and let \( q \) be the maximum of the arities of these constraint predicates. Then the size of the largest ascending chain of simple constrained atoms with minimal element \( \phi'/C' \) is at most \( Kn^r + Ln^q + n + 1 \).

**Proof:** Notice that for any simple constrained atom \( \phi/C \) that is \( \Sigma \)-more general than \( \phi'/C' \), \( \phi \) has size at most \( n \), and \( C \) contains only constraint predicates in \( \Sigma \) or \( C' \). We define the \((n, L, q)\)-height-bound of any simple constrained atom \( \phi/C \) of head size at most \( n \). Let \( m \) be the number of variables (not variable occurrences) in \( \phi \), let \( k \) be the size of \( \phi \), and let \( T \) be the number of distinct terms (not term occurrences) in \( \phi \). Let \( A \) be the number of atoms in the expanded form of constraint \( C \). Note that \( A \) is at most \( KT^r + LT^q \leq Kn^r + Ln^q \). Then the \((n, L, q)\)-height-bound of \( \phi/C \) is \( K(T + n - k)^r + L(T + n - k)^q - A + (n - k) + m + 1 \). The maximum value for the \((n, L, q)\)-height-bound, over all constrained atoms \( \phi/C \) with head size at most \( n \), is \( Kn^r + Ln^q + n + 1 \); this bound is realized when \( \phi \) has size \( n \), the arguments to \( \phi \) are distinct variables (so \( m = k = T \)), and \( C \) is empty (so \( A = 0 \)). Therefore, we complete the proof of the theorem by showing that the size of a maximum chain of simple constrained atoms with \( \phi'/C' \) as the minimal element and \( \phi/C \) as the maximal element is at most the \((n, L, q)\)-height-bound of \( \phi/C \). The proof is by induction on the \((n, L, q)\)-height-bound of \( \phi/C \).

**Basis:** The \((n, L, q)\)-height-bound of \( \phi/C \) is 1. Because \( A \) is at most \( KT^r + LT^q \), it must be the case that: \( \phi \) has size \( n \) (so \( n = k \)) and contains no variables (so \( m = 0 \),
and $A$ in fact is exactly $KT^r + LT^q$. But in this case $\phi/C$ has no $\Sigma$-instances other than itself, so the largest chain of which $\phi/C$ is the maximal element is a chain of size 1.

**Inductive:** We show that transforming $\phi/C$ to a simple constrained atom $\phi_2/C_2$ covered by $\phi/C$ reduces the $(n, L, q)$-height-bound. This suffices since, by Corollary 63, any maximal chain between $\phi/C$ and $\phi'/C'$ must include some such $\phi_2/C_2$. Consider three cases (assume $C$ and $C_2$ are in expanded form). **Case 1:** $\phi$ and $\phi_2$ are variants, where $\phi_\theta = \phi_2$, and $C_2$ is minimally tighter than $C_\theta$. Then going from $\phi/C$ to $\phi_2/C_2$ increases $A$ and changes the values of no other parameters in the definition of $(n, L, q)$-height-bound. The result is a decrease in $(n, L, q)$-height-bound. **Case 2:** $\phi_2/C_2 = (\phi/C)_\theta$, where $\theta$ maps one variable $x$ that appears in $\phi$ to another variable $y$ in $\phi$ and maps every other variable to itself or a distinct variable that does not appear in $\phi_1$. Going from $\phi/C$ to $\phi_2/C_2$ leaves $k$ unchanged but reduces $m$, so it reduces $(n - k) + m + 1$. It remains to show that $K(T + n - k)^r + L(T + n - k)^q - A$ does not increase. More specifically, we show that $A$ decreases no more than $K(T + n - k)^r + L(T + n - k)^q$ decreases. $A$ (the number of atoms in the constraint) may decrease only if applying $\theta$ to some atoms containing $x$ in the constraint yields atoms that are already in the constraint. But at most $(KT^r + LT^q) - (K(T - 1)^r + L(T - 1)^q)$ atoms in the constraint $C$ contain $x$, so $A$ is reduced by at most this much. $(KT^r + LT^q$ atoms can be built from terms in $\phi$, and $K(T - 1)^r + L(T - 1)^q$ such atoms can be built without using $x$.) On the other hand, since $\phi_2$ has one distinct term fewer than $\phi$, we know that $K(T + n - k)^r + L(T + n - k)^q$ is reduced to $K((T - 1) + n - k)^r + L((T - 1) + n - k)^r$. Because $k$ is no greater than $n$, this reduction is at least as great as that of $A$. **Case 3:** $\phi_2/C_2 = (\phi/C)_\theta$, where $\theta$ maps one variable in $\phi$ to a term $f(x_1, ..., x_i)$, where $f$ is an $i$-ary function and $x_1, ..., x_i$ are distinct variables that do not appear in $\phi$, and maps every other variable to itself or a distinct variable that does not appear in $\phi$. First consider the effect of such a substitution on $(n - k) + m + 1$. Substituting $f(x_1, ..., x_i)$ for $x$ removes $x$ from $\phi$ and adds to $\phi$ exactly $i$ variables. Therefore, applying such a substitution increases $m$ by $i - 1$ and increases $k$ by $i$ (decreases $(n - k)$ by $i$). Hence applying such a substitution reduces the $(n - k) + m + 1$ by 1. It remains to show that $K(T + n - k)^r + L(T + n - k)^q - A$ does not

84
increase. Again, we show that $A$ decreases no more than $K(T + n - k)^r + L(T + n - k)^q$
decreases. As noted in Case 2, $A$ (the number of atoms in the constraint) may decrease
only if applying $\theta$ to some atoms containing $x$ in the constraint yields atoms that are
already in the constraint. But this can occur only if $f(x_1, ..., x_i)$ appears in $\phi/C$, which
can be the case only if $f$ is a constant (since $x_1, ..., x_i$ do not appear in $\phi/C$) and appears
in $\phi/C$. Then the argument from Case 2, where $y$ is substituted for $x$, applies here as
well, with $f$ substituted for $x$ instead. □

By the way we have defined equivalence classes of atoms, a largest chain of equivalence
classes of atoms corresponds to a largest chain of atoms. (Take one atom from each
equivalence class; conversely, take the equivalence class of each atom.) Therefore, the
following corollary is an immediate result of Theorem 67.

**Corollary 68** Let $\Sigma$ be a constraint theory, without equality, that uses $K$ constraint
predicates, and let $r$ be the maximum of the arities of these constraint predicates. Let
$\phi/C$ be a $\Sigma$-admissible simple constrained atom, where $\phi$ has size $n$, $C$ contains at most
$L$ constraint predicates that do not appear in $\Sigma$, and the maximum of the arities of these
constraint predicates is $q$. Then the depth of $[\phi/C]_{\geq \Sigma}$ in the set of equivalence classes of
simple constrained atoms, ordered by $\geq_{\Sigma}$, is at most $Kn^r + Ln^q + n + 1$. In other words,
the size of the largest (ascending) chain of equivalence classes of simple constrained atoms
with $[\phi/C]_{\geq \Sigma}$ as the minimal element is at most $Kn^r + Ln^q + n + 1$.

We now note that for at least some nonempty constraint theories, this upper bound
is achieved, that is, chains of length exactly $Kn^r + Ln^q + n + 1$ exist. Let $\Sigma_L$ be the
following constraint theory.

$$
\Sigma_L = \{ \forall x_1 ... \forall x_r \ (\tau_K(x_1, ..., x_r) \rightarrow \tau_{K-1}(x_1, ..., x_r)), \\
\forall x_1 ... \forall x_r \ (\tau_{K-1}(x_1, ..., x_r) \rightarrow \tau_{K-2}(x_1, ..., x_r)), ..., \\
\forall x_1 ... \forall x_r \ (\tau_1(x_1, ..., x_r) \rightarrow \tau_1(x_1, ..., x_r)) \}
$$

Let the minimal element in the chain have the head $p(c_1, ..., c_n)$, where $p$ is an $n$-ary
predicate symbol, and $c_1, ..., c_n$ are distinct constants. Let the constraint of this element,
in expanded form, contain all the atoms built from terms in the head, constraint predicates in \( \Sigma_L \), and the constraint predicates \( \omega_1, ..., \omega_L \) (of arity \( q \)) that do not appear in \( \Sigma_L \). Notice that the constraint contains \( K n^r + L n^q \) atoms. To obtain the next \( K n^r + L n^q \) atoms in the chain, remove atoms from the constraint, one at a time, in an order that observes the following restriction. For all \( 1 \leq i < j \leq K \) and for all \( 1 \leq i_1, ..., i_r \leq n \), remove \( \tau_i(c_{i_1}, ..., c_{i_r}) \) before removing \( \tau_j(c_{i_1}, ..., c_{i_r}) \). To get the final \( n \) members of the chain, replace the constants in the head with distinct variables, in any order.

It is worth noting that a very different chain of the same size may be obtained, by using only one constant rather than \( n \) constants, and by using variable co-references, provided \( n = q = r \). Again let the ordering be with respect to \( \Sigma_L \). Let the minimal simple constrained atom in the chain have the head \( p(c, ..., c) \), where \( p \) again is a \( n \)-ary predicate symbol. Let the constraint of this atom, in expanded form, consist of one atom built from each constraint predicate \( \tau_1, ..., \tau_K, \omega_1, ..., \omega_L \), where all the arguments in each atom are \( c \). To get the next atom in the chain, rewrite every occurrence of \( c \) (in the head and in the constraint) to a variable \( x_1 \). To get the next atom, replace the second occurrence of \( x_1 \) in the head by some new variable \( x_2 \), and let the constraint consist of every atom that can be built from the constraint predicates \( \tau_1, ..., \tau_K, \omega_1, ..., \omega_L \) and the variables \( x_1 \) and \( x_2 \). Repeat \( n - 2 \) times to get the next \( n - 2 \) atoms, with the last one having the head \( p(x_1, ..., x_n) \) and a constraint consisting of every atom that can be built from \( \tau_1, ..., \tau_K, \omega_1, ..., \omega_L \) and the variables \( x_1, ..., x_n \). Notice that the constraint contains \( K n^r + L n^q \) atoms. To get the remaining \( K n^r + L n^q \) atoms, remove from the constraint one atom at a time in any order, subject to the following restriction. For all \( 1 \leq i < j \leq K \) and for all \( 1 \leq i_1, ..., i_r \leq n \), remove \( \tau_i(x_{i_1}, ..., x_{i_r}) \) before removing \( \tau_j(x_{i_1}, ..., x_{i_r}) \).

Of course, there are a variety of other ways to get chains of this size as well. We close this subsection with one more way, that involves a unary function symbol, again using \( \Sigma_L \). Let the minimal element in the chain have the head \( p(f(f(...(c)...))) \), where \( p \) is an \( n \)-ary predicate symbol, and \( f \) is a unary function symbol, and \( c \) is a constant. Let the constraint of this element contain all the atoms built from constraint predicates \( \tau_1, ..., \tau_K, \omega_1, ..., \omega_L \) and terms in the head, that is, \( c, f(c), f(f(c)), \) etc. Then proceed to
build this chain exactly as we built the first one, but with the terms $c, f(c)$, etc., taking
the place of the constants $c_1, ..., c_n$.

4.2.3 Ascending Chains of Sorted Atoms

Recall that every sorted atom is a simple constrained atom whose constraint predicates
are unary and are applied to variables only, and every sort theory is a constraint theory
whose constraint predicates are unary. As a result, the following corollaries of Lemma 61
are easy to verify.

Lemma 69 Let $\Sigma$ be a sort theory that uses a finite set of sorts. Then the dominating
set of any equivalence class of sorted atoms $[\alpha]_{\geq \Sigma}$ is finite.

Corollary 70 Let $\Sigma$ be a sort theory that uses a finite set of sorts. Then there exists
no infinite chain of equivalence classes of sorted atoms, ordered by $\geq \Sigma$, between any pair
of such equivalence classes, and there exists no infinite ascending chain of equivalence
classes of sorted atoms.

Corollary 71 Let $\Sigma$ be a sort theory that uses only a finite set of sorts. Let $\alpha_1$ be a
sorted atom that is strictly $\Sigma$-more general than $\alpha_2$. Then any maximal chain between $\alpha_1$
and $\alpha_2$ includes exactly one sorted atom covered by $\alpha_1$ and exactly one sorted atom that
covers $\alpha_2$.

Corollary 72 Let $\Sigma$ be a sort theory that uses a finite set of sorts. Then there exists no
infinite descending chain of equivalence classes of sorted atoms whose members are all
$\Sigma$-more general than some class $[\alpha]_{\geq \Sigma}$.

Corollary 73 Let $\Sigma$ be a sort theory that uses a finite set of sorts. Then the cover set
of any equivalence class of sorted atoms is finite.

Corollary 74 Let $\Sigma$ be a sort theory that uses a finite set of sorts. Then any set (finite
or infinite) of sorted atoms has only finitely many mubs.
In addition, Theorem 67 implies that where $K$ is the number of sorts in $\Sigma$, the largest ascending chain of equivalence classes of sorted atoms, ordered by $\geq_{\Sigma}$, with a minimal element of size $n$ is $Kn + Ln + n + 1$. Recall that for sorted atoms, the constraint is a conjunction of atoms of the form $\tau(x)$, where $\tau$ is a sort symbol and $x$ is a variable in the head, such that at most one conjunct involves any given variable. Therefore, $L$ is at most the size of the head ($n$). Hence we may rewrite the bound as $Kn + n^2 + n + 1$. In fact, no chain of sorted atoms (size at most $n$) is even this large; a better bound of $Kn + n + 1$ can be obtained. Nevertheless, we will have no need for a precise bound on the sizes of chains of sorted atoms, but only to know that they are finite, so we will not present a proof here.

### 4.3 Sufficient Conditions for GLBs and LUBs

As stated in Chapter 1, unification is used to compute greatest or maximal lower bounds in an instantiation ordering. The existence of greatest lower bounds (glbs) simplifies unification-based reasoning procedures. Similarly, generalization or anti-unification computes least or minimal upper bounds, and the existence of least upper bounds (lubs) often simplifies non-deductive reasoning procedures. In particular, we will see in Chapter 8 that the question of whether lubs exist for a restricted class of constrained atoms, ordered by $\geq_{\Sigma}$, sometimes determines whether that class can be learned from examples according to a widely-recognized formalization of inductive learning (pac-learnability).

We now examine sufficient conditions for the existence of glbs and lubs. Recall that if both glbs and lubs exist, then the set of equivalence classes of constrained formulas, ordered by $\geq_{\Sigma}$, is a lattice.

Following Reynolds’ [60] work with ordinary atoms and their algebraic properties, we provide the following simple extension to the set of constrained formulas.

**Definition 75** A generalized constrained formula (GCF) is either a constrained formula or one of the special symbols $\top$ and $\bot$. A generalized constrained atom (GCA) is a constrained atom or $\top$ or $\bot$.  

88
We extend the $\geq_\Sigma$ ordering to apply to GCFs as follows. For any constraint theory $\Sigma$ and any GCF $A$:

- $\top \geq_\Sigma A$
- $A \geq_\Sigma \bot$
- $A \geq_\Sigma \top$ only if $A = \top$
- $\bot \geq_\Sigma A$ only if $A = \bot$ or $A$ is a $\Sigma$-inadmissible constrained formula

We have defined the meanings of $\forall A$ and $\exists A$ where $A$ is a constrained formula, but we have not done so where $A$ is $\top$ or $\bot$. We take the semantic value of $\forall \top$ to be $False$, of $\exists \top$ to be $True$, of $\forall \bot$ to be $True$, and of $\exists \bot$ to be $False$. Doing so allows the semantic results of the $\geq_\Sigma$ ordering, as applied to constrained formulas, to extend to the ordering’s application to GCFs.

Recall that the $\geq_\Sigma$ ordering is a quasi-ordering, or preorder, on constrained formulas, for any choice of $\Sigma$. Clearly it is, more generally, a quasi-ordering on GCFs and on equivalence classes of GCFs. This partially-ordered set has a unique greatest element, $[\top]_{\geq_\Sigma}$, which contains only the GCF $\top$, and a unique least element, $[\bot]_{\geq_\Sigma}$, which contains $\bot$ and all the $\Sigma$-inadmissible constrained formulas.\(^5\) Later we will see conditions under which this partially-ordered set is a lattice, though in general it may not be, and we will note why this property is particularly important for applications.

Before discussing sufficient conditions for glbs and lubs, we observe the following. Any restriction on constrained formulas may be defined in terms of constrained formulas in variable-abstract form. For example, the class of constrained formulas that have no equality in the constraint may also be defined as the class of constrained formulas in variable-abstract form with at most one equation $x = t$ in the constraint for each variable $x$ in the head, where $t$ contains no variables that appear in the head. Throughout the rest of this section, we assume that constrained formulas are in variable-abstract form.

\(^5\)Notice that the addition of $\bot$, unlike the addition of $\top$, does not add another equivalence class. Rather, $\bot$ serves only a mnemonic purpose, so that $[\bot]_{\Sigma}$ can be used to denote the set of $\Sigma$-inadmissible constrained formulas.
The following definition helps us identify sufficient conditions for the existence of glbs and lubs.

**Definition 76 (Closure Under Conjunction or Disjunction)** We say that a class \( \mathcal{R} \) of constrained formulas is closed under conjunction (closed under disjunction) with respect to a constraint theory \( \Sigma \) if for any constrained formulas \( \phi/C_1 \) and \( \phi/C_2 \), \( \mathcal{R} \) contains a constrained formula \( \phi/C \), where \( C \) is \( \Sigma \)-equivalent to \( C_1 \land C_2 \) (\( C_1 \lor C_2 \)). We say that \( \mathcal{R} \) is closed under infinite conjunction (closed under infinite disjunction) with respect to \( \Sigma \) if for any (possibly infinite) set of constrained formulas \( \{ \phi/C_1, \phi/C_2, \ldots \} \), \( \mathcal{R} \) contains a constrained formula \( \phi/C \), where \( C \) is \( \Sigma \)-equivalent to \( C_1 \land C_2 \land \ldots \) (\( C_1 \lor C_2 \lor \ldots \)).

We now consider cases under which sets of equivalence classes have glbs or lubs. Any set consisting of just one equivalence class has a glb and lub, which is the equivalence class itself. Any set of equivalence classes \( \{[e_1]_{\geq \Sigma}, \ldots, [e_n]_{\geq \Sigma} \} \), \( n \geq 2 \), where some \( e_i \) and \( e_j \) have heads with different structure or predicates, has a glb, \( [\top]_{\geq \Sigma} \), and an lub, \( [\bot]_{\geq \Sigma} \).

Any set of equivalence classes that includes \( [\top]_{\geq \Sigma} \) has a glb, \( [\top]_{\geq \Sigma} \); the mlbs of such a set (of size at least two) are the same as when \( [\top]_{\geq \Sigma} \) is removed. Similarly, any set of equivalence classes that includes \( [\bot]_{\geq \Sigma} \) has an lub, \( [\bot]_{\geq \Sigma} \); the mubs of such a set (of size at least two) are the same as when \( [\bot]_{\geq \Sigma} \) is removed.

What, then, about glbs and lubs of sets of the form \( \{[e_1]_{\geq \Sigma}, \ldots, [e_n]_{\geq \Sigma} \} \), \( n \geq 2 \), where \( e_1, \ldots, e_n \) have heads with the same structure and predicates, and so can be written as constrained formulas with the same head? If \( \mathcal{R} \) is closed under conjunction then any two equivalence classes \( [\phi/C_1]_{\geq \Sigma} \) and \( [\phi/C_2]_{\geq \Sigma} \) have a glb, \( [\phi/C_3]_{\geq \Sigma} \), where \( C_3 \) is \( \Sigma \)-equivalent to \( C_1 \land C_2 \). Similarly, if \( \mathcal{R} \) is closed under disjunction then any two equivalence classes \( [\phi/C_1]_{\geq \Sigma} \) and \( [\phi/C_2]_{\geq \Sigma} \) have an lub, \( [\phi/C_3]_{\geq \Sigma} \), where \( C_3 \) is \( \Sigma \)-equivalent to \( C_1 \lor C_2 \). If every such set of size two has a glb and an lub, then any finite set has a glb and an lub. The following two lemmas give additional sufficient conditions for the existence of glbs and lubs.
Lemma 77 If \( \mathcal{R} \) is closed under conjunction, and every pair \( \{[e_1]_{\geq 2}, [e_2]_{\geq 2}\} \) has only finitely many mlbs, then in fact every pair has a glb. If \( \mathcal{R} \) is closed under infinite conjunction, then every pair has a glb.

Proof: We assume that \( e_1 \) and \( e_2 \) do not belong to \([T]_{\geq 2}\) or \([\bot]_{\geq 2}\), and that \( e_1 \) and \( e_2 \) can be written with the same head, as \( \phi/C_1 \) and \( \phi/C_2 \) (otherwise, the result holds trivially). If \( [\phi/C_1]_{\geq 2} \) and \( [\phi/C_2]_{\geq 2} \) have two or more mlbs, these cannot include \([T]_{\geq 2}\) or \([\bot]_{\geq 2}\), since either must be the only mlb (a glb). Furthermore, the mlbs can all be written with the head \( \phi \). Consider two cases. If \( [\phi/C_1]_{\geq 2} \) and \( [\phi/C_2]_{\geq 2} \) have a finite set of mlbs, \( \{[\phi/C_1']_{\geq 2}, ..., [\phi/C_n']_{\geq 2}\} \), then in fact they have a glb: \( [\phi/C]_{\geq 2} \), where \( C \) is \( \Sigma \)-equivalent to \( C_1' \land ... \land C_n' \). If \( [\phi/C_1]_{\geq 2} \) and \( [\phi/C_2]_{\geq 2} \) have an infinite set of mlbs, \( \{[\phi/C_1']_{\geq 2}, [\phi/C_2']_{\geq 2}, ..., \} \), and \( \mathcal{R} \) is closed under infinite conjunction, then they have a glb: \( [\phi/C]_{\geq 2} \), where \( C \) is \( \Sigma \)-equivalent to \( C_1' \land C_2' \land ... \). \( \square \)

Lemma 78 If \( \mathcal{R} \) is closed under disjunction, and every pair \( \{[e_1]_{\geq 2}, [e_2]_{\geq 2}\} \) has only finitely many mubs, then in fact every pair has a lub. If \( \mathcal{R} \) is closed under infinite disjunction, then every pair has an lub.

Proof: Again we may assume without loss of generality that \( e_1 \) and \( e_2 \) do not belong to \([T]_{\geq 2}\) or \([\bot]_{\geq 2}\), and that \( e_1 \) and \( e_2 \) can be written as constrained formulas with the same head, as \( \phi/C_1 \) and \( \phi/C_2 \). If \( [\phi/C_1]_{\geq 2} \) and \( [\phi/C_2]_{\geq 2} \) have two or more mubs, these cannot include \([T]_{\geq 2}\) or \([\bot]_{\geq 2}\), since either must be the only mub (an lub). Furthermore, the mubs can all be written with the head \( \phi \). Consider two cases. If \( [\phi/C_1]_{\geq 2} \) and \( [\phi/C_2]_{\geq 2} \) have a finite set of mubs, \( \{[\phi/C_1']_{\geq 2}, ..., [\phi/C_n']_{\geq 2}\} \), then in fact they have an lub: \( [\phi/C]_{\geq 2} \), where \( C \) is \( \Sigma \)-equivalent to \( C_1' \lor ... \lor C_n' \). If \( [\phi/C_1]_{\geq 2} \) and \( [\phi/C_2]_{\geq 2} \) have an infinite set of mubs, \( \{[\phi/C_1']_{\geq 2}, [\phi/C_2']_{\geq 2}, ..., \} \), and \( \mathcal{R} \) is closed under infinite disjunction, then they have an lub: \( [\phi/C]_{\geq 2} \), where \( C \) is \( \Sigma \)-equivalent to \( C_1' \lor C_2' \lor ... \). \( \square \)

It follows from these results, for example, that the equivalence classes of ordinary formulas and of the full set of constrained formulas have glbs and lubs. Equivalence classes of the full class of constrained formulas have lubs because the set of constrained
formulas is closed under conjunction and \textit{glbs} because this set is closed under disjunction. Equivalence classes of ordinary formulas have \textit{lubs} because the set of ordinary formulas, in variable-abstract form, is closed under conjunction: given two constraints, \( C_1 \) and \( C_2 \), conforming to the restrictions for constraints of variable-abstract ordinary formulas, unification yields a constraint \( C_3 \) that is equivalent to \( C_1 \land C_2 \) and that conforms to the restrictions for constraints of variable-abstract ordinary formulas. (If \( C_1 \) and \( C_2 \) are not unifiable, any unsatisfiable constraint is equivalent to \( C_1 \land C_2 \).) It has \textit{glbs} because (1) the set of ordinary formulas is closed under conjunction, and (2) the set of \textit{mubs} is finite. In Chapter 6 we will see that equivalence classes of simple constrained formulas have \textit{lubs} and \textit{glbs}, provided \( \Sigma \) does not contain the equality predicate and uses only finitely-many constraint predicates.

If \( \Sigma \) may contain equality, on the other hand, then it is possible that even some sets of ordinary atoms have no \textit{glb} or \textit{lub} in the \( \geq_{\Sigma} \) ordering. For example, let \( \Sigma_M \) be 
\[
\{g(a,b,b) = g(a,a,b), g(c,d,d) = g(c,c,d)\}.
\]
Then \( g(a,b,b) \) and \( g(c,d,d) \) have two \textit{mubs}, \( g(x,y,y) \) and \( g(x,x,y) \), but no \textit{lub}. It is well-known [58] that \( E \)-unification may yield multiple maximal common instances of two atoms, so \textit{glbs} are not guaranteed either.

Even if \( \Sigma \) contains no equality, sorted atoms do not meet any of the sufficient conditions given here for their equivalence classes to have \textit{lubs} and \textit{glbs}. And in fact it is quite easy to show that sort theories exist such that some sets of sorted atoms do not have \textit{lubs} or \textit{glbs}. For a simple example, let \( \tau \) and \( \omega \) denote sorts, and let \( \Sigma_M \) be 
\[
\{\tau(a), \tau(b), \tau(c), \omega(b), \omega(c), \omega(d)\}.
\]
Then the (sorted) atoms \( p(b) \) and \( p(c) \) have two \textit{mubs}, \( p(x:\tau) \) and \( p(x:\omega) \), neither of which is \( \Sigma_M \)-more general than the other. Therefore, \( p(b) \) and \( p(c) \) have no \textit{lub}. (We use this simple theory, and a similar one built with a unary function, in Chapter 8 to show that sorted atoms cannot be pac-learned.) Frisch [24] has shown that sorted atoms can have infinitely many \textit{milbs}, modulo the renaming of variables, even if the sort theory is finite (and so contains only finitely many constraint predicates, or sort symbols). We have seen that any set of sorted atoms has a finite set of \textit{mubs}, provided \( \Sigma \) uses a finite set of sorts. We show in Chapter 5 that two sorted atoms may have a number of \textit{mubs} that is exponential in the sizes of the sorted atoms.
4.4 VC-Dimension of Ordinary Atoms

In this brief section we address the question of the VC-Dimension of the poset of equivalence classes of atoms. In Chapter 6, once we have shown that the poset of equivalence classes of simple constrained atoms is an upper semi-lattice, we will determine the VC-Dimension of simple constrained atoms. (We will have no use for the VC-Dimension of equivalence classes of sorted atoms.) Recall that the poset of equivalence classes of atoms is a lattice, and recall our earlier observation that an upper bound on the VC-Dimension of a lattice (more generally, an upper semi-lattice) is the size of the largest embedded Boolean Algebra (Section 4.1). Recall also that an upper bound on the size of the largest embedded Boolean Algebra is the size of the largest chain. From these observations, we have immediately that the VC-dimension of equivalence classes of ordinary atoms, where the atoms have size at most $n$, is at most $n + 1$. Note that, therefore, the VC-dimension for these classes grows polynomially in the size of the atoms. We next note that in fact the bound of $n + 1$ is reached for ordinary atoms.

We present an example for ordinary atoms of size 4, from which it is easy also to identify such sets for all other values of $n$. Consider the following set of 5 atoms of size 4: \{ $p(a,a,a,a)$, $p(b,a,a,a)$, $p(b,b,a,a)$, $p(b,b,b,a)$, $p(b,b,b,b)$ \}. For each atom in the set, there is a way to exclude, or fail to cover, only that atom, and these ways of exclusion are orthogonal. For example, any atom that has a $b$ for the first argument fails to cover the first atom. Any atom whose first two arguments are the same fails to cover the second atom. Any atom whose second and third arguments are the same fails to cover the third atom. Because these are orthogonal, they can be combined to exclude any subset of the atoms while including all others. Here are two examples of such combinations. The atom $p(x,y,y,y)$ excludes the third and fourth atoms because the second, third, and fourth arguments are the same. The atom $p(b,b,x,x)$ excludes the first atom because the first argument is a $b$, the second atom because the first and second arguments are the same, and the fourth atom because the third and fourth arguments are the same. For any
subset we can construct an atom that covers the atoms in the subset and excludes the others.

4.5 Directions for Further Work

The applications in this dissertation require the use of only a few, central algebraic and combinatorial properties of posets, which this chapter has examined. A variety of other properties of posets exist that we have not considered. It is likely that some of these properties also are applicable to the study of equivalence classes of ordinary or constrained atoms, and the search for such applications be a fascinating area for further research.

To see another area for further research, recall that the $\geq_\Sigma$ ordering on constrained atoms is equivalent to the $\Sigma$-entailment ordering on the universal or existential closures of these atoms. One could imagine a variety of forms of logical sentences, other than the universal or existential closures of constrained atoms, ordered by entailment with respect to a background theory. These would also yield posets of equivalence classes of logical sentences. A study of the algebraic and combinatorial properties of these posets could reveal properties of the entailment ordering that would be useful in automated reasoning.
Chapter 5

Computing Sorted Anti-Unification

5.1 Preliminaries

Having defined sorted atoms, as well as an instantiation ordering on sorted atoms that takes into account a sort theory, we are ready to define sorted anti-unification and examine its computational properties. Sorted anti-unification is of particular interest because it uses taxonomic background information, which has a rich tradition in artificial intelligence research. For example, deduction in sorted logic, in which a taxonomic background theory is built into the deduction mechanism, led to the first theorem prover to successfully solve Schubert’s Steamroller [68; 17; 61]. Much research has been invested in deductive techniques for sorted logics [15; 14; 25; 24; 66; 67; 65]. The arguments and methods for knowledge-base vivification, another approach to efficient deduction, are based on the use of taxonomic background information [42; 8; 20]. The use of taxonomic background information is prevalent in the history of machine learning as well. Early machine learning systems such as AQ and INDUCE performed inductive inference with respect to taxonomic background information [18; 45]. In addition, the classic example of a speed-up learning system based on induction, LEX, operated with respect to taxonomic background information [46; 62]. The work on this thesis began as an investigation of anti-unification with respect to taxonomic background information, motivated by the research just mentioned. The expectation was that if useful anti-unification operations
exist, sorted anti-unification certainly should be a useful operation. This chapter casts
some doubts on the utility of sorted anti-unification. Through a detailed analysis of sorted
anti-unification, this chapter leads to new insights into some of the research described
above and the use of taxonomic information in general; these insights are developed more
fully in Chapter 8.

A definition of sorted anti-unification analogous to that of ordinary anti-unification
would be the following: find a least sorted atom, in the $\geq_\Sigma$ ordering, that is $\Sigma$-more
general than every sorted atom in a given set. Example 79, which involves sorted atoms
and the sort theory $\Sigma_1$ (Chapter 2), illustrates the error in this definition. (Throughout
this chapter we represent sorted atoms and sorted terms in the in-line syntax.)

**Example 79**

Let $E = \{\text{eats(}clyde,\text{nuts)\}}, \text{eats(jumbo, nuts)}\}$. 

$E$ has two minimal $\Sigma_1$-generalizations:

\[\text{eats} (x: \text{ELEPHANT, nuts})\]
\[\text{eats} (y: \text{IN-CIRCUS, nuts})\]

$E$ has a least ordinary generalization ($LGG_2$):

\[\text{eats} (x, \text{nuts})\]

We would like to use a $\Sigma$-generalization to characterize the set of all sorted atoms
that are $\Sigma$-more general than any given set of sorted atoms, in the same way that we use
the $LGG$ of a set of ordinary atoms. But Example 79 shows that there may be no least
$\Sigma$-generalization: neither $\text{eats} (x: \text{ELEPHANT, nuts})$ nor $\text{eats} (y: \text{IN-CIRCUS, nuts})$ is $\Sigma_1$-more
general than the other. (In fact, we saw this in Chapter 4 already—that equivalence
classes of sorted atoms may have no $mubs$. ) An analogous characterization therefore
requires a set of minimal $\Sigma$-generalizations. Specifically, the goal of sorted anti-unification
is to find a complete set of incomparable \( \Sigma \)-generalizations (or \( CIG_{\geq \Sigma} \)) for a given set of sorted atoms. A set \( G \) is a \( CIG_{\geq \Sigma} \) of a set \( E \) of sorted atoms if and only if

- each member of \( G \) is \( \Sigma \)-more general than every member of \( E \) [Correctness],
- any sorted atom that is \( \Sigma \)-more general than every member of \( E \) is \( \Sigma \)-more general than some member of \( G \) [Completeness], and
- no member of \( G \) is \( \Sigma \)-more general than any other [Incomparability].

This definition of \( CIG_{\geq \Sigma} \) extends to any quasi-ordered set. Let \( S \) be any set with an associated quasi-ordering. Then a \( CIG_{\geq} \) of any subset \( E \) of \( S \) is a set \( G \) such that

- each member of \( G \) is greater than every member of \( E \) [Correctness],
- any member of \( S \) that is greater than every member of \( E \) is greater than some member of \( G \) [Completeness], and
- no member of \( G \) is greater than any other [Incomparability].

The following observation is useful in understanding \( CIG_{\geq \Sigma} \)s of sorted atoms. It is easily verified that two sorted atoms or sorted terms are variants (each is \( \Sigma \)-more general than the other) if and only if one can be obtained from the other by renaming the variables and sorts. By “renaming sorts”, we mean replacing sorts with logically equivalent ones according to \( \Sigma \).\(^1\) Furthermore, we say that one set of sorted atoms or sorted terms is a variant of a second set if and only if the members of the sets can be put in one-to-one correspondence such that corresponding members are variants of one another.

Some sets of sorted atoms have no \( \Sigma \)-generalization; a \( CIG_{\geq \Sigma} \) of any such set is the empty set.

**Lemma 80** Let \( \Sigma \) be a sort theory that uses a finite set of sorts. A set of sorted atoms has an empty \( CIG_{\geq \Sigma} \) if and only if it has members built from different ordinary predicates.

\(^1\)A sort \( \tau \) is logically equivalent to \( \tau' \) according to a sort theory \( \Sigma \) if and only if \( \tau \preceq_{\Sigma} \tau' \) and \( \tau' \preceq_{\Sigma} \tau \).
Proof: If: A set of sorted atoms that has members built from different ordinary predicates has no \( \Sigma \)-generalizations, for any \( \Sigma \). Only If: Any set of sorted atoms built from the same ordinary predicate has at least one \( \Sigma \)-generalization. From Lemma 69, the set of \( \Sigma \)-generalizations is finite, up to the renaming of variables and sorts. It follows that the set has a nonempty \( CIG_{\geq \Sigma} \).

Might other sets of sorted atoms have more than one \( CIG_{\geq \Sigma} \)? Can a \( CIG_{\geq \Sigma} \) be infinite? The following observation is useful in answering these questions. Recall that, by definition, for any set \( S \) quasi-ordered by some relation \( \geq \), and for any two elements \( e_1 \) and \( e_2 \) of \( S \), we have \( e_1 \geq e_2 \) if and only if \( [e_1]_{\geq} \geq [e_2]_{\geq} \).

Observation 3 Let \( S \) be a set quasi-ordered by \( \geq \), where \( \geq \) induces no infinite chains between elements of \( S \). \( G = \{g_1, g_2, \ldots\} \) (finite or infinite) is a \( CIG_\geq \) of a set \( E = \{e_1, e_2, \ldots\} \) (finite or infinite) in \( S \) if and only if \( [g_1]_{\geq}, [g_2]_{\geq}, \ldots \) are exactly the minimal upper bounds of the set \( \{[e_1]_{\geq}, [e_2]_{\geq}, \ldots\} \) in the corresponding poset.

Recall that for any sort theory \( \Sigma \), if \( \Sigma \) uses only finitely many sorts then there exists no infinite chain between two sorted atoms. Therefore, Observation 3 yields the following lemmas.

Lemma 81 Any \( CIG_{\geq \Sigma} \) of a set of sorted atoms can be obtained from another by uniformly renaming variables and their sorts.

Because of this lemma, we sometimes speak of the \( CIG_{\geq \Sigma} \) of a set of sorted atoms.

Lemma 82 Let \( \Sigma \) be a sort theory that uses only finitely many sorts. Every \( CIG_{\geq \Sigma} \) of a set of sorted atoms is finite.

Proof: This corollary follows from Observation 3, and Corollary 74, which says that any set of equivalence classes of sorted atoms has only finitely many mubs. \( \square \)

We have noted that sorted anti-unification may return a non-singleton \( CIG_{\geq \Sigma} \). The following example illustrates an additional distinction between sorted and ordinary anti-unification, which a sorted anti-unification algorithm must accommodate.
Example 83

Let \( E = \{ p(\text{mom}(clyde)), p(\text{mom}(jumbo)) \} \).

Then a \( CIG_{\Sigma_1} \) of \( E \) is:

\[
\{ p(y:\text{GRAY}), \\
p(\text{mom}(x:\text{ELEPHANT})), \\
p(\text{mom}(z:\text{IN-CIRCUS})) \}
\]

while a least ordinary generalization (\( LGG_{\geq} \)) of \( E \) is:

\[ p(\text{mom}(x)) \]

In Example 83, the sorted variable \( y:\text{GRAY} \) \( \Sigma_1 \)-subsumes neither of the other sorted terms \( \text{mom}(x:\text{ELEPHANT}) \) and \( \text{mom}(z:\text{IN-CIRCUS}) \). This contrasts with the unsorted case, where a variable always subsumes any other term. We refer to a variable that \( \Sigma \)-subsumes all members of a set of (possibly sorted) terms as a variable generalization of that set. We refer to a non-variable sorted atom or term that \( \Sigma \)-subsumes all members of a set as a structured generalization. A sorted anti-unification algorithm cannot assume that a variable generalization of a set of sorted terms \( \Sigma \)-subsumes every structured generalization of that set. It therefore must compare the variable generalizations with the structured generalizations.

Because variable generalizations need not \( \Sigma \)-subsume structured generalizations, members of a \( CIG_{\geq} \) may have different structures. In the \( CIG_{\geq_{\Sigma_1}} \) of Example 83, the argument to \( p \) in one generalization is a variable while the arguments to \( p \) in the other two are built from the function symbol “\( \text{mom} \)”.

Such structural differences can become more remarkable as the expressions being generalized grow more complex. That variable generalizations need not \( \Sigma \)-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. The function \( \text{mom} \) is polymorphic because whether a term built from \( \text{mom} \) denotes a member of the
sort GRAY depends on the argument to \textit{mom}. If the argument is \textit{clyde} or \textit{jumbo}, the term
denotes something gray; otherwise it may not. If \textit{mom} were instead a monomorphic function,
that is, if all moms were known to be gray, the variable \( y \) \text{GRAY} would \( \Sigma_1 \)-subsume
any structured generalization of \( \text{mom}(\text{clyde}) \) and \( \text{mom}(\text{jumbo}) \).

The following example provides additional insight into the computation of sorted
anti-unification.

\textbf{Example 84}

Let \( E = \{ \text{loves}(\text{clyde}, \text{mom}(\text{clyde})), \text{loves}(\text{jumbo}, \text{mom}(\text{jumbo})) \} \).

Then a \( CIG_{\Sigma_1} \) of \( E \) is

\[
\{ \text{loves}(x: \text{ELEPHANT}, \text{mom}(x: \text{ELEPHANT})), \text{loves}(z: \text{IN}-\text{CIRCUS}, \text{mom}(x: \text{ELEPHANT})), \\
\text{loves}(x: \text{ELEPHANT}, \text{mom}(z: \text{IN}-\text{CIRCUS})), \text{loves}(z: \text{IN}-\text{CIRCUS}, \text{mom}(z: \text{IN}-\text{CIRCUS})), \\
\text{loves}(x: \text{ELEPHANT}, y: \text{GRAY}), \text{loves}(z: \text{IN}-\text{CIRCUS}, y: \text{GRAY}) \}
\]

while \( \text{loves}(x, \text{mom}(x)) \) is a least ordinary generalization of \( E \).

The interesting point of Example 84 is that the \( CIG_{\geq \Sigma_1} \) of \( E \) is built from the cross-
product of \( CIG_{\geq \Sigma_1} \)'s of the parts—the arguments—of the formulas in \( E \). Care is required
in building the \( CIG_{\geq \Sigma_1} \) 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 sorted atoms rather than sorted
terms.

Of course, in some cases a sorted anti-unification procedure may not halt if \( \Sigma \) uses
an infinite set of sort symbols. Therefore, the remainder of this chapter assumes that
the set of sort symbols in \( \Sigma \) is finite. It also assumes that the algorithm has a list of the
sorts used in \( \Sigma \) (for example, perhaps \( \Sigma \) is finite and the algorithm has a copy of \( \Sigma \)).
Furthermore, computation of sorted anti-unification obviously requires access to certain
other kinds of information from the sort theory \( \Sigma \). Specifically, the ability to determine
the sorted atomic consequences of \( \Sigma \) (determine whether \( \Sigma \models \forall \tau(t) \) for any given sort \( \tau \)
and sorted term \( t \) ) is sufficient for the computation of sorted anti-unification, as we will
see in the following section. This ability is “almost” necessary as well. Specifically, it is straightforward to verify that if no two sorts are $\Sigma$-equivalent (there are no redundant sort symbols), then a sorted anti-unification algorithm, if it exists, can be used to determine the sorted atomic consequences of $\Sigma$. In general, the sorted atomic consequences of a sort theory can be undecidable, though many useful theories exist for which these consequences can be computed. For example, these consequences are decidable (and efficiently computable) for the example sort theory $\Sigma_1$. We call queries about the sorted atomic consequences of $\Sigma$ *taxonomic queries*. In the remainder of this section we assume the existence of an oracle to answer these queries.

### 5.2 A Sorted Anti-Unification Algorithm

The Sorted Anti-Unification (SA) Algorithm, shown in Figure 5.2, computes the $CIG_{\geq \Sigma}$ of any finite set of sorted atoms. The algorithm uses a bijection $\phi$ from any pair of the form $(\langle t_1, \ldots, t_m \rangle, \tau)$, where $\langle t_1, \ldots, t_m \rangle$ is a tuple of sorted terms such that $\Sigma \models \forall \tau(t_i)$ for all $1 \leq i \leq m$, to a variable of sort $\tau$.\footnote{This function is similar to the bijection between variable names and tuples of terms that the Ordinary Anti-Unification Algorithm uses [39].} To avoid accidental variable collisions, we further stipulate that the range of $\phi$ is disjoint from the alphabet from which the sorted atoms or terms given as input to the algorithm are built.

**Theorem 85 (Correctness of the Sorted Anti-Unification Algorithm)** *Given an oracle for taxonomic queries, the Sorted Anti-Unification Algorithm halts and returns a $CIG_{\geq \Sigma}$ of $E$.*

**Proof:** To prove the theorem, we show that for any input as specified, the algorithm terminates, and the set it returns is a $CIG_{\geq \Sigma}$ of the input, that is, it is correct, complete, and without comparable members. Throughout the proof of the Sorted Anti-Unification (SA) Algorithm’s correctness, we call the sort theory “$\Sigma_i$”; therefore, we at times speak of the SA (result of applying the Sorted Anti-Unification Algorithm), Structured-Set, and
Sorted Anti-Unification (SA) Algorithm

Input: A tuple of sorted terms/atoms $E = \langle \alpha_1, ..., \alpha_m \rangle$ and a sort theory $\Sigma$.

Output: A $CIG_{\geq} \Sigma$ of $\langle \alpha_1, ..., \alpha_m \rangle$.

1. If $\alpha_1 = \alpha_2 = ... = \alpha_m$ then return $\{ \alpha_1 \}$.

2. If $E$ contains sorted atoms with different predicate symbols or contains both sorted terms and sorted atoms, then return $\{ \}$. 

3. If every member of $E$ is built from the same predicate or primary function, that is, if for all $1 \leq i \leq m$, $\alpha_i$ is $p(t_{i,1}, t_{i,2}, ..., t_{i,n})$ where $p$ is an $n$-ary predicate/function symbol and $t_{i,1}, t_{i,2}, ..., t_{i,n}$ are sorted terms, then
   Let Structured-Set = $\{ p(r_1, r_2, ..., r_n) \mid r_j \in SA(\langle t_{1,j}, ..., t_{m,j} \rangle, \Sigma), \text{for all } 1 \leq j \leq n \}$
   Otherwise,
   Let Structured-Set = $\{ \}$. 

4. If $\alpha_1, ..., \alpha_m$ are sorted terms, then
   a. Let Variable-Set be:
      \[ \{ \phi(\langle \alpha_1, ..., \alpha_m \rangle, \tau) \mid \Sigma \models \varphi(\alpha_i) \text{ for all } 1 \leq i \leq m, \text{ and} \]
      \[ \text{for every } e \in \text{Structured-Set: } \Sigma \not\models \varphi(e) \} \]
   b. For each variable $x: \tau \in \text{Variable-Set}$ (in pre-specified order by sorts):
      If another variable $y: \omega$ remains in Variable-Set, and $\omega \leq \Sigma \tau$, then
      Remove $x: \tau$ from Variable-Set
      Otherwise,
      Variable-Set = $\{ \}$. 

5. Return Structured-Set $\cup$ Variable-Set.

Figure 5.1: The Sorted Anti-Unification (SA) Algorithm
Variable-Set of a tuple of sorted atoms or sorted terms with the sort theory \( \Sigma \) assumed. We use the \( S \)-more general formulation of the \( \Sigma \)-more general ordering. Recall that, according to this formulation, one sorted atom, \( \alpha_1 \), is \( \Sigma \)-more general than a second, \( \alpha_2 \), if and only if there exists a substitution, \( \theta \), that is well-sorted with respect to \( \Sigma \), such that \( \alpha_1 \theta = \alpha_2 \). Recall also that a substitution \( \theta \) is well-sorted with respect to a sort theory \( \Sigma \) if and only if for any variable \( x: \tau \), if \( \langle x: \tau \rangle \theta = t \) then \( \Sigma \models \neg \tau(t) \).

**Correctness:** Let \( \beta \) be an arbitrary member of \( \text{SA}(\alpha_1, \ldots, \alpha_m) \) where \( \alpha_1, \ldots, \alpha_m \) are sorted expressions built from some alphabet \( A \). Correctness requires that \( \beta \geq \Sigma \alpha_i \), for each \( 1 \leq i \leq m \). We prove the result for an arbitrary \( \alpha_i \). Based on the bijection \( \phi \), we build a substitution \( \theta \), well-sorted with respect to \( \Sigma \), for which \( \beta \theta = \alpha_i \). Let \( \theta \) map each variable not in the range of \( \phi \) to itself. Let \( x: \tau \) be an arbitrary variable in the range of \( \phi \). Because \( \phi \) is a bijection, \( \phi \) maps between \( x: \tau \) and exactly one pair, \( \langle \langle t_1, \ldots, t_m \rangle, \tau \rangle \).

By the definition of \( \phi \), \( \Sigma \models \neg \tau(t_i) \), for all \( 1 \leq i \leq m \). Let \( \theta \) map \( x: \tau \) to \( t_i \). It follows that \( \theta \) is well-sorted with respect to \( \Sigma \). It remains to show that \( \beta \theta = \alpha_i \). We do so by induction on the complexity of \( \beta \). Recall that a variable is in the range of \( \phi \) only if it is not in the alphabet \( A \) from which \( \alpha_i \) is built. Hence if \( \beta = \alpha_i \), then no variable in \( \beta \) is in the range of \( \phi \), so \( \beta \theta = \beta = \alpha_i \). Thus we may assume that \( \beta \neq \alpha_i \). The inductive hypothesis is that if \( \alpha_1, \ldots, \alpha_m \) are sorted atoms or sorted terms built from the alphabet \( A \), and \( \beta \in \text{SA}(\alpha_1, \ldots, \alpha_m) \), then \( \beta \theta = \alpha_i \).

**Basis:** \( \beta \) is a constant or a variable. \( \beta \) cannot be a constant, for if it were then \( \beta \) and \( \alpha_1, \ldots, \alpha_m \) would all be the same, contrary to the assumption that \( \beta \neq \alpha_i \). Therefore, \( \beta \) is a variable, \( x: \tau \), distinct from \( \alpha_i \), so it must be in \( \text{Variable-Set}(\alpha_1, \ldots, \alpha_m) \) at the end of Step 4. Therefore \( x: \tau = \phi(\langle \alpha_1, \ldots, \alpha_m \rangle, \tau) \). Then \( \langle x: \tau \rangle \theta = \alpha_i \) by the definition of \( \theta \).

**Inductive:** \( \beta = p(r_1, \ldots, r_n) \). Then \( \alpha_i = p(t_{i,1}, \ldots, t_{i,n}) \). Because we have assumed \( \beta \neq \alpha_i \), \( \beta \) is in \( \text{Structured-Set}(\alpha_1, \ldots, \alpha_m) \), so for all \( 1 \leq j \leq n \) we know that \( r_j \) is in \( \text{SA}(t_{1,j}, \ldots, t_{m,j}) \). For all \( 1 \leq j \leq n, t_{1,j}, \ldots, t_{m,j} \) are sorted terms built from alphabet \( A \), so we may apply the inductive hypothesis, which tells us \( r_j \theta = t_{i,j} \). Therefore \( p(r_1, \ldots, r_n) \theta = p(t_{i,1}, \ldots, t_{i,n}) \). \( \square \)
Completeness: Let $\beta'$ be any sorted atom or sorted term such that $\beta' \geq_{\Sigma} \alpha_i$, for all $1 \leq i \leq m$. We show that $\beta' \geq_{\Sigma} \beta$ for some $\beta \in SA(\alpha_1, ..., \alpha_m)$. If $\alpha_1 = ... = \alpha_m$, then the result is obvious, so assume this is not the case. Because $\beta' \geq_{\Sigma} \alpha_i$, for all $1 \leq i \leq m$, substitutions $\theta_1, ..., \theta_m$, well-sorted with respect to $\Sigma$, exist such that $\beta'\theta_i = \alpha_i$, for all $1 \leq i \leq m$. Based on $\theta_1, ..., \theta_m$, we construct a substitution $\theta'$, well-sorted with respect to $\Sigma$, such that $\beta'\theta' = \beta$ for some $\beta \in SA(\alpha_1, ..., \alpha_m)$. Let $\theta'$ map each variable not occurring in $\beta'$ to any $\Sigma$-instance of that variable. Let $x: \tau$ be an arbitrary variable occurring in $\beta'$. $\Sigma \models \tau(x: \tau)\theta_i$, for all $1 \leq i \leq m$, because each $\theta_i$ is well-sorted with respect to $\Sigma$. Therefore $x: \tau$ is in $\text{Variable-Set}((x: \tau)\theta_1, ..., (x: \tau)\theta_m)$, modulo renaming, as computed at step 4a. Hence some $\Sigma$-instance of $x\tau$ is in $SA((x: \tau)\theta_1, ..., (x: \tau)\theta_m)$. Let $(x: \tau)\theta'$ be that $\Sigma$-instance. $\theta'$ so defined is clearly well-sorted with respect to $\Sigma$. We show by induction on the structure of $\beta'$ that $\beta'\theta' = \beta$, for some $\beta \in SA(\alpha_1, ..., \alpha_m)$.

Basis: $\beta'$ is a constant or a variable. $\beta'$ is not a constant, for if it were, $\beta', \alpha_1, ..., \alpha_m$ would all be the same, contrary to our initial assumption. Therefore, $\beta'$ is a variable, so $\beta'\theta' \in SA(\alpha_1, ..., \alpha_m)$ by the definition of $\theta'$.

Inductive: $\beta' = p(t_1, ..., t_n)$. Thus $\alpha_i = p(t_1\theta_i, ..., t_n\theta_i)$, for all $1 \leq i \leq m$. By the inductive hypothesis, for all $1 \leq j \leq n$ we have $t_j\theta' = r_j$ for some $r_j \in SA(t_j\theta_1, ..., t_j\theta_m)$. Then $p(t_1, ..., t_n)\theta' = p(r_1, ..., r_n)$. Because $r_j \in SA(t_j\theta_1, ..., t_j\theta_m)$, for all $1 \leq j \leq n$, we know $p(r_1, ..., r_n)$ is in the $\text{Structured-Set}(\alpha_1, ..., \alpha_m)$. Notice that the SA Algorithm never removes a member of $\text{Structured-Set}$. Therefore, $p(r_1, ..., r_n)$ is in $SA(\alpha_1, ..., \alpha_m)$.

Incomparability of members: For any $\beta_1$ and $\beta_2 \in SA(\alpha_1, ..., \alpha_m)$, if $\beta_1 \geq_{\Sigma} \beta_2$ then $\beta_1 = \beta_2$. No member of the final Variable-Set is $\Sigma$-more general than another, because of step 4b. Step 4a ensures that no member of the Variable-Set is $\Sigma$-more general than a member of $\text{Structured-Set}$. Because no non-variable term can be $\Sigma$-more general than a variable, no member of the $\text{Structured-Set}$ can be $\Sigma$-more general than a member of the Variable-Set. Induction on the complexity of the least complex sorted atom in the input verifies that no member of the $\text{Structured-Set}$ is $\Sigma$-more general than any other.
Termination of the 2-SG Algorithm: Algorithm steps 1 and 2 obviously halt. We show that each of the subsequent steps halts if it is reached. We show termination for step 3 last, as it depends on termination of the other steps.

Step 4: If all \( \alpha_i \) are sorted terms, the “Otherwise” branch of the conditional is taken, and step 4 halts. If not all \( \alpha_i \) are sorted terms, the “If” branch is taken. The oracle for taxonomic queries can be used to compute the set of sorts whose variables \( \Sigma \)-subsume all \( \alpha_i \) but do not \( \Sigma \)-subsume any member of Structured-Set. Because \( \phi \) is a computable function, step 4a halts. The Variable-Set is therefore finite at the end of step 4a. Hence, given the oracle for taxonomic queries, step 4b terminates with a finite Variable-Set.

Step 5: If step 5 is reached, the Structured-Set computed in step 3 must be finite. We have seen that the Variable-Set computed in step 4 is finite. Therefore, the union of the two sets is finite and can be obtained in finite time.

Step 3: We show by induction on the complexity of the least complex \( \alpha_i \) that step 3 terminates if it is reached.

Basis: \( \alpha_i \) is a constant or a variable. If the algorithm reaches step 3, it is not the case that \( \alpha_1 = \ldots = \alpha_m \). The “Otherwise” branch of the conditional is taken, so step 3 halts after setting the Structured-Set to \{ \}.

Inductive: If any two sorted terms in the input are built from different primary function symbols, then the “Otherwise” branch of the conditional is taken. Hence step 3 halts after setting the Structured-Set to \{ \}. Otherwise, \( \alpha_i \) is of the form \( p(t_{i,1}, \ldots, t_{i,n}) \), for all \( 1 \leq i \leq m \). The “If” branch of the conditional is taken. By the inductive hypothesis, step 3 halts for each of the recursive calls to the algorithm. We have already shown that all other steps halt if reached; hence, the recursive instantiations of the algorithm halt and return the terms \( r_j \in SA(t_{1,j}, \ldots, t_{m,j}) \) for all \( 1 \leq j \leq n \). Therefore the set of choices of \( r_j \), each \( 1 \leq j \leq n \), is finite. Step 3 builds a term for each combination of choices of \( r_1, \ldots, r_n \). Because the number of such combinations is finite, Step 3 halts. \( \Box \)
The reader is encouraged to walk through the SA 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 79 shows that a set of sorted atoms may have more than one minimal sorted generalization. The algorithm therefore returns a set rather than a single generalization.

Example 83 shows that variable generalizations are not guaranteed to $\Sigma$-subsume structured generalizations. Therefore, step 4 of the SA 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 $\Sigma$-subsume variable generalizations, and, because of the way they are built, no structured generalization built by the SA Algorithm $\Sigma$-subsumes another.

Example 84 illustrates the cross-product operation inherent in generalizing sorted atoms 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 SA Algorithm builds structured generalizations of sorted atoms using the cross-product of the sorted generalizations of their components. Correct variable co-references are ensured by the bijection $\phi$ used for variable naming in step 4, the only step at which new variables are introduced. For example, if $\phi$ maps $\langle clyde, jumbo \rangle_{\text{ELEPHANT}}$ to $x: \text{ELEPHANT}$ during anti-unification of the first arguments in the sorted atoms $\text{loves}(clyde, \text{mother}(clyde))$ and $\text{loves}(jumbo, \text{mother}(jumbo))$, then during anti-unification of the second arguments, it maps an input to $x: \text{ELEPHANT}$ if and only if that input is also $\langle clyde, jumbo \rangle_{\text{ELEPHANT}}$. Thus a variable repeats in a $\Sigma$-generalization only if a tuple of sorted terms, one from each input sorted atom, repeats in the same way in the input sorted atoms. Note that repetition of any particular tuple of sorted terms in the input sorted atoms does not guarantee a corresponding variable repetition in every $\Sigma$-generalization, because the tuple of sorted terms may be generalized in more than one way.

106
5.3 Problem Complexity

We have seen that sorted anti-unification is computable given an oracle for taxonomic queries. Unfortunately, the size of a $CIG_{\geq 2}$ can be exponential in the sizes of the sorted atoms generalized, so sorted anti-unification can require exponential time and space. Figure 5.2 shows the two possible causes of such large $CIG_{\geq 2}$s.

The sketch on the left shows two terms, $a$ and $b$, that denote individuals belonging to both of the sorts $\tau$ and $\omega$. $\tau$ and $\omega$ are the smallest (by set inclusion) sorts to which these individuals belong, and $\tau$ is neither a subset nor a superset of $\omega$. Thus $a$ and $b$ have two minimal generalizations: a variable of sort $\tau$ and a variable of sort $\omega$. The situation depicted here can, of course, occur with more than two sorts.

The sketch on the right of Figure 3 shows two terms, $f(t_1)$ and $f(t_2)$, whose least structured generalization is $f(t)$. Notice that the function $f$ is polymorphic: the sort theory entails $\forall(\omega(f(t_1)))$ and $\forall(\omega(f(t_2)))$, but it does not entail $\forall(\omega(f(t)))$. Therefore a variable of sort $\omega$ is a variable generalization of $f(t_1)$ and $f(t_2)$ that does not subsume, with respect to the sort theory, any structured generalization. Notice that similar situations can involve more than one sort. They also can involve multiple minimal structured generalizations rather than a least structured generalization. In addition, they can occur with functions of arity greater than one.
Situations of these two forms are the only causes for multiple minimal sorted generalizations. When more complex expressions are generalized, both of these causes can arise repeatedly. If neither is present, sorted anti-unification is tractable. Specifically, a least sorted generalization exists and the SA algorithm computes it. We now examine how the two causes for multiple minimal sorted generalizations can create exponentially large $CIG_{\geq \Sigma}$s.

First, suppose the sorts $\tau$ and $\omega$ are the two most specific (with respect to $\Sigma$) sorts that contain individuals $a$ and $b$, as in Figure 5.2, and we wish to compute the $CIG_{\geq \Sigma}$ of $p(a, a, a, a, a)$ and $p(b, b, b, b, b)$. The $CIG_{\geq \Sigma}$ contains $2^5 = 32$ sorted atoms. It contains every sorted atom, up to renaming, of the form $p(t_1, t_2, t_3, t_4, t_5)$, where each of $t_1, t_2, t_3, t_4, t_5$ is a variable $x: \tau$ or $y: \omega$. From this example, it is easy to see how the size of a $CIG_{\geq \Sigma}$ can be exponential in the sizes of the sorted atoms being generalized. Second, suppose $\Sigma$ contains some polymorphic function $f$ such that the variable generalization $x: \omega$ of terms $f(t_1)$ and $f(t_2)$ is not $\Sigma$-more general than any structured generalization, also as in Figure 5.2. For simplicity, assume that $t_1$ and $t_2$ have a least structured generalization. The $CIG_{\geq \Sigma}$ of $p(f(t_1), f(t_1), f(t_1), f(t_1))$ and $p(f(t_2), f(t_2), f(t_2), f(t_2))$ contains $2^5 = 32$ sorted atoms: it contains every sorted atom of the form $p(t_1', t_2', t_3', t_4', t_5')$ where each of $t_1', ..., t_5'$ is either $f(t)$ or is a variable $x: \omega$.

A $CIG_{\geq \Sigma}$ can always be represented in size polynomial in the size of the sorted atoms generalized and the number of sort symbols in the sorted atoms or $\Sigma$. We can modify the SA algorithm to compute such a compact representation. Unfortunately, Theorem 86, which follows, shows that the problem has no polynomial-time algorithm (assuming $P \neq NP$) even when we allow a compact representation of the $CIG_{\geq \Sigma}$s and are guaranteed polynomial-time response to taxonomic queries. By compact representation in Theorem 86, we mean any representation whose size is polynomial in the sum of the sizes of the sorted atoms generalized, and from which we can, in time polynomial in the size of the representation, determine whether a variant of any given sorted atom is in the $CIG_{\geq \Sigma}$. We call the problem of computing such a compact representation, given efficient response to taxonomic queries, simplified sorted anti-unification (SSA).
Theorem 86  *Simplified Sorted Anti-Unification (SSA) is NP-hard.*

*Proof:* The proof shows that 3-Satisfiability (3SAT) is polynomially reducible to SSA. Let \( \beta = b_1 \land \ldots \land b_k \) be an arbitrary 3-CNF formula (propositional formula in conjunctive normal form with conjuncts of length at most 3) over the propositional variables \( p_1, \ldots, p_l \), and let \(|\beta|\) be the length of the encoding of \( \beta \). We wish to know if \( \beta \) is satisfiable. Consider an instance of SSA constructed as follows.

We first construct the sorted atoms to be generalized. Let \( f \) be an \( l \)-ary function symbol, let \( \alpha_1 \) be \( f(p_1, \ldots, p_l) \), where \( p_1, \ldots, p_l \) are interpreted as constant symbols, and let \( \alpha_2 \) be \( f(c_1, \ldots, c_l) \), where \( c_1, \ldots, c_l \) are constant symbols, none of which occurs in \( \{p_1, \ldots, p_l\} \).

We next construct the sort theory, \( \Sigma \). Let \( \Sigma \) use exactly the sort symbols ‘\( T \)’ and ‘\( T' \)’ (intuitively, true), ‘\( F \)’ (for false), ‘\( \text{UNIV} \)’, and ‘\( B_1 \), ..., ‘\( B_k \)’ (one for each of \( b_1 \), ..., \( b_k \)). Let \( \Sigma \) contain the clauses described below plus the assumption that every individual is a member of the sort \( \text{UNIV} \). Note that \( \Sigma \) does not entail that either \( T \) or \( F \) has a subset; for certain choices of \( \beta \), \( \Sigma \) may entail that \( B_i \) is a subset of \( B_j \), for some \( 1 \leq i, j \leq k \).

- For all \( 1 \leq i \leq l \), \( T(p_i) \).
- For all \( 1 \leq i \leq l \), \( F(p_i) \).
- For all \( 1 \leq i \leq l \), \( T(c_i) \).
- For all \( 1 \leq i \leq l \), \( F(c_i) \).

\( \forall x_1 \ldots \forall x_l \) \( (T(x_i) \rightarrow B_j(f(x_1, \ldots, x_i))) \), for any \( 1 \leq i \leq l \) and \( 1 \leq j \leq k \) such that \( p_i \) is a literal in the propositional clause \( b_j \).

\( \forall x_1 \ldots \forall x_l \) \( (F(x_i) \rightarrow B_j(f(x_1, \ldots, x_i))) \), for any \( 1 \leq i \leq l \) and \( 1 \leq j \leq k \) such that \( \overline{p_i} \) is a literal in the propositional clause \( b_j \).

\( \forall x_1 \ldots \forall x_l \) \( (B_1(f(x_1, \ldots, x_l)) \land \ldots \land B_k(f(x_1, \ldots, x_l)) \rightarrow T'(f(x_1, \ldots, x_l))) \).

Note that the theory entails, among other sentences, the following: \( T'(f(p_1, \ldots, p_l)) \) and \( T'(f(c_1, \ldots, c_l)) \). Therefore, any variable of sort \( T' \) is a \( \Sigma \)-generalization of \( \alpha_1 \) and \( \alpha_2 \).
As an example, let $\beta$ be

$$(p_1 \lor p_2 \lor p_4) \land (p_2 \lor p_3) \land (p_3 \lor p_4)$$

Then $\Sigma$ contains the atoms $T(p_1), \ldots, T(p_4), T(c_1), \ldots, T(c_4), F(p_1), \ldots, F(p_4), F(c_1), \ldots, F(c_4)$, as well as the clauses:

\[
\begin{align*}
\forall x_1 \forall x_2 \forall x_3 \forall x_4 & \ (T(x_1) \rightarrow B_1(f(x_1, x_2, x_3, x_4))) \\
\forall x_1 \forall x_2 \forall x_3 \forall x_4 & \ (T(x_2) \rightarrow B_1(f(x_1, x_2, x_3, x_4))) \\
\forall x_1 \forall x_2 \forall x_3 \forall x_4 & \ (T(x_4) \rightarrow B_1(f(x_1, x_2, x_3, x_4))) \\
\forall x_1 \forall x_2 \forall x_3 \forall x_4 & \ (T(x_2) \rightarrow B_2(f(x_1, x_2, x_3, x_4))) \\
\forall x_1 \forall x_2 \forall x_3 \forall x_4 & \ (T(x_3) \rightarrow B_2(f(x_1, x_2, x_3, x_4))) \\
\forall x_1 \forall x_2 \forall x_3 \forall x_4 & \ (F(x_4) \rightarrow B_3(f(x_1, x_2, x_3, x_4))) \\
\forall x_1 \forall x_2 \forall x_3 \forall x_4 & \ (B_1(f(x_1, x_2, x_3, x_3)) \land B_2(f(x_1, x_2, x_3, x_3)) \land B_3(f(x_1, x_2, x_3, x_3)) \rightarrow T'(f(x_1, x_2, x_3, x_4)))
\end{align*}
\]

Clearly $\Sigma$ may be built in time polynomial in $|\beta|$. It is also easily verified that any taxonomic query can be answered in time polynomial in the size of the query (the theory is finite and contains no recursion). We now show that $\beta$ is satisfiable if and only if no variable of type $T'$ is in the $CIG_{\Sigma}^\alpha$ of $\alpha_1$ and $\alpha_2$. Showing this proves that a polynomial-time algorithm for SSA also provides a polynomial-time algorithm for 3SAT. (Recall that we require the ability to check our representation of a $CIG_{\Sigma}^\alpha$ in polynomial time for the presence of a particular expression, modulo variable renaming, so we can check in polynomial time for a variable of type $T'$ in the $CIG_{\Sigma}^\alpha$.)

The proof rests on the following 2 claims, which are proven afterward.

1. For any term $t \in CIG_{\Sigma}^\alpha(\alpha_1, \alpha_2)$, $t$ is a variable of sort $T'$ or is of the form $f(y_1:t_1, \ldots, y_i:t_i)$, where $y_1:t_1, \ldots, y_i:t_i$ are distinct variables of sort $T$ or $F$.

2. $\beta$ is satisfiable if and only if for some term $f(y_1:t_1, \ldots, y_i:t_i)$, where $y_1:t_1, \ldots, y_i:t_i$ are distinct variables of sort $T$ or $F$, $\Sigma \models \forall T'(f(y_1:t_1, \ldots, y_i:t_i))$ (in which case any variable $x:T'$ is $\Sigma$-more general than $f(y_1:t_1, \ldots, y_i:t_i)$).
If: If there is no variable of sort $T'$ in $CIG_{\geq \Sigma} (\alpha_1, \alpha_2)$ then $\beta$ is satisfiable. Let $x:T'$ be any variable of sort $T'$. We know that $x:T' \geq_{\Sigma} \alpha_1$ and $x:T' \geq_{\Sigma} \alpha_2$. Then there must be a $t \in CIG_{\geq \Sigma}(\alpha_1, \alpha_2)$ such that $x:T' \geq_{\Sigma} t$, by the definition of $CIG_{\geq \Sigma}$ (Completeness). From Claim 1 it follows that $t$ is either some variable of sort $T'$ or is a term $f(y_1; \tau_1, \ldots, y_i; \tau_i)$, where $y_1; \tau_1, \ldots, y_i; \tau_i$ are distinct variables of sort $T$ or $F$. If there is no variable of sort $T'$ in $CIG_{\geq \Sigma}(\alpha_1, \alpha_2)$ then $t$ is $f(y_1; \tau_1, \ldots, y_i; \tau_i)$, where $y_1; \tau_1, \ldots, y_i; \tau_i$ are distinct variables of sort $T$ or $F$. By $x:T' \geq_{\Sigma} t$, it must be that $\Sigma \models \forall t'(f(y_1; \tau_1, \ldots, y_i; \tau_i))$. Then by Claim 2 $\beta$ is satisfiable.

Only If: If $\beta$ is satisfiable then there is no variable of sort $T'$ in $CIG_{\geq \Sigma}(\alpha_1, \alpha_2)$. Assume $\beta$ is satisfiable yet some variable $x:T'$ is in $CIG_{\geq \Sigma}(\alpha_1, \alpha_2)$. By Claim 2 and $\beta$ satisfiable, there is a term $f(y_1; \tau_1, \ldots, y_i; \tau_i)$, where $y_1; \tau_1, \ldots, y_i; \tau_i$ are distinct variables of sort $T$ or $F$, for which $\Sigma \models \forall t'(f(y_1; \tau_1, \ldots, y_i; \tau_i))$. Then $x:T' \geq_{\Sigma} f(y_1; \tau_1, \ldots, y_i; \tau_i)$. By $x:T'$ in $CIG_{\geq \Sigma}(\alpha_1, \alpha_2)$, there is no term $t \in CIG_{\geq \Sigma}(\alpha_1, \alpha_2)$ for which $x:T' \geq_{\Sigma} t$. Then by $x:T' \geq_{\Sigma} f(y_1; \tau_1, \ldots, y_i; \tau_i)$ and the transitivity of $\geq_{\Sigma}$, there is no term $t \in CIG_{\geq \Sigma}(\alpha_1, \alpha_2)$ for which $f(y_1; \tau_1, \ldots, y_i; \tau_i) \geq_{\Sigma} t$. But because $y_1; \tau_1, \ldots, y_i; \tau_i$ are distinct variables of sort $T$ or $F$, $f(y_1; \tau_1, \ldots, y_i; \tau_i)$ is $\Sigma$-more general than $\alpha_1$ and $\alpha_2$. Thus by the definition of $CIG_{\geq \Sigma}$ (Completeness), there must be a $t \in CIG_{\geq \Sigma}(\alpha_1, \alpha_2)$ for which $f(y_1; \tau_1, \ldots, y_i; \tau_i) \geq_{\Sigma} t$. This gives a contradiction.

Claim 1 For any term $t \in CIG_{\geq \Sigma}(\alpha_1, \alpha_2)$, $t$ is a variable of sort $T'$ or is of the form $f(y_1; \tau_1, \ldots, y_i; \tau_i)$, where $y_1; \tau_1, \ldots, y_i; \tau_i$ are distinct variables of sort $T$ or $F$.

We first show that any term $\Sigma$-more general than $\alpha_1$ and $\alpha_2$ is of the form $x:\text{UNIV}$, $x:T'$, $x:B_1$, ..., $x:B_k$, or $f(y_1; \tau_1, \ldots, y_i; \tau_i)$, where $y_1; \tau_1, \ldots, y_i; \tau_i$ are distinct variables of sort $T$, $F$, or UNIV. We then show that each term $f(y_1; \tau_1, \ldots, y_i; \tau_i)$, where $y_1; \tau_1, \ldots, y_i; \tau_i$ are distinct variables of sort $T$ or $F$, is in $CIG_{\geq \Sigma}(\alpha_1, \alpha_2)$, modulo renaming. We finally show that if a term is of the form $x:\text{UNIV}$, $x:B_1$, ..., $x:B_k$, or $f(y_1; \tau_1, \ldots, y_i; \tau_i)$, where $y_1; \tau_1, \ldots, y_i; \tau_i$ are distinct variables of sort $T$, $F$, or UNIV and at least one is UNIV, then
the term is strictly $\Sigma$-more general than some term $f(y_1: \tau_1, ..., y_l: \tau_l)$, where $y_1: \tau_1, ..., y_l: \tau_l$ are distinct variables of sort $T$ or $F$. By the definition of $CIG_{\Sigma}$, the result follows.

(1) Any term that is $\Sigma$-more general than $\alpha_1$ and $\alpha_2$ is of the form $x: \text{UNIV}$, $x: T'$, $x: B_1$, ..., $x: B_k$, or $f(y_1: \tau_1, ..., y_l: \tau_l)$, where $y_1: \tau_1, ..., y_l: \tau_l$ are distinct variables of sort $T$, $F$, or \text{UNIV}. Assume some other term $t$ is $\Sigma$-more general than $\alpha_1$ and $\alpha_2$. If $t$ is of the form $f(y_1: \tau_1, ..., y_l: \tau_l)$, where $y_i: \tau_i = y_j: \tau_j$, for some $1 \leq i \neq j \leq l$, then $t \not\backsim \Sigma \alpha_1$. Then $t$ must be, modulo renaming, either (1) $x: \tau$, where $\tau$ is in $\Sigma$ but is not one of \text{UNIV}, $T'$, $B_1$, ..., $B_k$, or (2) $f(y_1: \tau_1, ..., y_l: \tau_l)$, where $y_1: \tau_1, ..., y_l: \tau_l$ are distinct variables and some $\tau_i$, $1 \leq i \leq l$, is neither $T$, $F$, nor \text{UNIV}.

\textit{Case 1:} Then $t$ is of the form $x: F$ or $x: T$. If $t$ is $x: F$ then $\Sigma \models F(\alpha_1)$ and $\Sigma \models F(\alpha_2)$. Because $\Sigma$ is a set of definite clauses, if $\Sigma \models F(\alpha_1)$ then there must be a definite clause in $\Sigma$ whose consequent is of the form $F(s)$, where $s \geq \alpha_1$. There is no such clause. The same argument applies for $t$ of the form $x: T$.

\textit{Case 2:} If $t = f(y_1: \tau_1, ..., y_l: \tau_l)$, where some $\tau_i$, $1 \leq i \leq l$, is not $T$, $F$, or \text{UNIV}, then $\tau_i$ must be one of $B_1$, ..., $B_k$, or $T'$. By $t \backsim \Sigma \alpha_1$ and $t \backsim \Sigma \alpha_2$, it follows that $\Sigma \models \tau_i(p_i)$ and $\Sigma \models \tau_i(c_i)$. Then, because $\Sigma$ is a set of definite clauses, $\Sigma$ must contain a clause whose consequent is of the form $\tau_i(s)$, where $\tau_i$ is one of $B_1$, ..., $B_k$, or $T'$ and $s \geq p_i$. There is no such clause.

(2) Every term $f(y_1: \tau_1, ..., y_l: \tau_l)$, where $y_1: \tau_1, ..., y_l: \tau_l$ are distinct variables of sort $T$ or $F$, is in $CIG_{\Sigma}(\alpha_1, \alpha_2)$, modulo renaming. Assume some such term $t = f(y_1: \tau_1, ..., y_l: \tau_l)$ is not in $CIG_{\Sigma}(\alpha_1, \alpha_2)$, modulo renaming. Clearly $t$ is $\Sigma$-more general than $\alpha_1$ and $\alpha_2$, so by definition of $CIG_{\Sigma}$ (Completeness), $t$ must be $\Sigma$-more general than some other term $t' \in CIG_{\Sigma}(\alpha_1, \alpha_2)$. By $t \backsim \Sigma t'$, $t'$ is of the form $f(s_1, ..., s_l)$ where $\Sigma \models \tau_i(s_i)$ for all $1 \leq i \leq l$. By $t' \backsim \Sigma \alpha_1$ and $t' \backsim \Sigma \alpha_2$, $s_1, ..., s_l$ must be distinct variables, since $p_i \neq c_i$ for all $1 \leq i \leq l$ and $p_i \neq p_j$ for all $1 \leq i \neq j \leq l$. But because each $\tau_i$, $1 \leq i \leq l$, is a variable of sort $T$ or $F$, and neither $T$ nor $F$ has a subsort, it cannot be the case that $s_1, ..., s_l$ are distinct variables yet $\Sigma \models \tau_i(s_i)$ for $1 \leq i \leq l$. 

112
(3) If a term $t$ is of the form $x: \text{UNIV}, x:B_1, \ldots, x:B_k$, or $f(y_1: \tau_1, \ldots, y_i: \tau_i)$, where $y_1: \tau_1, \ldots, y_i: \tau_i$ are distinct variables of sort $T$, $F$, or UNIV and at least one is UNIV, then $t$ is strictly $\Sigma$-more general than some term $f(y_1: \tau_1, \ldots, y_i: \tau_i)$, where $y_1: \tau_1, \ldots, y_i: \tau_i$ are distinct variables of sort $T$ or $F$. The result is clear if $t$ is a variant of $x: \text{UNIV}$. The result is also clear if $t = f(y_1: \tau_1, \ldots, y_i: \tau_i)$, where $y_1: \tau_1, \ldots, y_i: \tau_i$ are distinct variables of sort $T$, $F$, or UNIV and at least one is UNIV. We therefore need only prove the case where $t$ is a variant of $x:B_i$, for arbitrary $1 \leq i \leq l$.

For arbitrary $i, 1 \leq i \leq l$, clause $b_i$ in $\beta$ contains at least one propositional literal $p_j$ or $\overline{p_j}$, for some $1 \leq j \leq l$. Consider 2 cases.

Case 1: $b_i$ contains a propositional literal $p_j$, for some $1 \leq j \leq l$. Therefore, $\Sigma$ contains a clause $\forall x_1 \ldots \forall x_l (T(x_j) \rightarrow B_i(f(x_1, \ldots, x_l)))$. Then $\Sigma \models \forall B_i(f(y_1:T, \ldots, y_i:T))$. Thus $x:B_i \geq_{\Sigma} f(y_1:T, \ldots, y_i:T)$.

Case 2: $b_i$ contains a propositional literal $\overline{p_j}$, for some $1 \leq j \leq l$. Therefore, $\Sigma$ contains a clause $\forall x_1 \ldots \forall x_l (F(x_j) \rightarrow B_i(f(x_1, \ldots, x_l)))$. Then $\Sigma \models \forall B_i(f(y_1:F, \ldots, y_i:F))$. Thus $x:B_i \geq_{\Sigma} f(y_1:F, \ldots, y_i:F)$.

Claim 2 $\beta$ is satisfiable if and only if for some term $f(y_1: \tau_1, \ldots, y_i: \tau_i)$, where $y_1: \tau_1, \ldots, y_i: \tau_i$ are distinct variables of sort $T$ or $F$, $\Sigma \models \forall f(y_1: \tau_1, \ldots, y_i: \tau_i)$.

Clearly $\Sigma \models T'(x)$ if and only if $\Sigma \models \forall B_1(f(y_1: \tau_1, \ldots, y_i: \tau_i)) \land \ldots \land B_k(f(y_1: \tau_1, \ldots, y_i: \tau_i))$. Therefore, it suffices to show that $\beta$ is satisfiable if and only if the following hold for some term $f(y_1: \tau_1, \ldots, y_i: \tau_i)$, where $y_1: \tau_1, \ldots, y_i: \tau_i$ are distinct variables of sort $T$ or $F$: $\Sigma \models \forall B_1(f(y_1: \tau_1, \ldots, y_i: \tau_i)), \ldots, \Sigma \models \forall (B_k(f(y_1: \tau_1, \ldots, y_i: \tau_i)))$.

We define the obvious bijection, $V$, between terms of the form $f(y_1: \tau_1, \ldots, y_i: \tau_i)$ (modulo renaming), where $y_1: \tau_1, \ldots, y_i: \tau_i$ are distinct variables of sort $T$ or $F$, and truth assignments over $p_1, \ldots, p_i$. $V$ maps a term $f(y_1: \tau_1, \ldots, y_i: \tau_i)$ to the truth assignment $A$ such that for all $i, 1 \leq i \leq l$, $A(p_i) = \text{True}$ if $\tau_i = T$, and $A(p_i) = \text{False}$ if $\tau_i = F$.

If: If there exists a term $f(y_1: \tau_1, \ldots, y_i: \tau_i)$, where $y_1: \tau_1, \ldots, y_i: \tau_i$ are distinct variables of sort $T$ or $F$, such that $\Sigma \models T'(f(y_1: \tau_1, \ldots, y_i: \tau_i))$, then $\beta$ is satisfiable. Let $f(y_1: \tau_1, \ldots, y_i: \tau_i)$
be any term for which $y_1: \tau_1, ..., y_i: \tau_i$ are distinct variables of sort $T$ or $F$, and $\Sigma \models \forall B_1(f(y_1: \tau_1, ..., y_i: \tau_i))$, ..., $\Sigma \models \forall B_k(f(y_1: \tau_1, ..., y_i: \tau_i))$. Then for each $B_i$, $1 \leq i \leq k$, there is some $j$, $1 \leq j \leq l$, such that either (1) $p_j$ is in $b_i$ and $\tau_j = T$ or (2) $\overline{p_j}$ is in $b_i$ and $\tau_j = F$. Let $V(f(y_1: \tau_1, ..., y_i: \tau_i)) = A$. From the definition of $V$ it follows that for each $b_i$, $1 \leq i \leq k$, there is some $j$, $1 \leq j \leq l$, such that either (1) $p_j$ is in $b_i$ and $A(p_j) = \text{true}$ or (2) $\overline{p_j}$ is in $b_i$ and $A(P_j) = \text{false}$. Then $A$ satisfies each $b_i$, $1 \leq i \leq k$, so $A$ satisfies $\beta$.

**Only If:** If $\beta$ is satisfiable then for some term $f(y_1: \tau_1, ..., y_i: \tau_i)$, where $y_1: \tau_1, ..., y_i: \tau_i$ are distinct variables of sort $T$ or $F$, $\Sigma \models \forall T'(f(y_1: \tau_1, ..., y_i: \tau_i))$. Assume that for every term $f(y_1: \tau_1, ..., y_i: \tau_i)$, where $y_1: \tau_1, ..., y_i: \tau_i$ are distinct variables of sort $T$ or $F$, $\Sigma \not\models \forall B_i(f(y_1: \tau_1, ..., y_i: \tau_i))$ for some $1 \leq i \leq k$. We show that $\beta$ must be unsatisfiable. Consider an arbitrary such term $f(y_1: \tau_1, ..., y_i: \tau_i)$. There is some $B_i$, $1 \leq i \leq k$, such that $\Sigma \not\models \forall B_i(f(y_1: \tau_1, ..., y_i: \tau_i))$. Then for each $j$, $1 \leq j \leq l$, if $p_j$ is in $b_i$ then $\tau_j = F$ and if $\overline{p_j}$ is in $b_i$ then $\tau_j = T$. Let $A$ be $V(f(y_1: \tau_1, ..., y_i: \tau_i))$. Then for each $j$, $1 \leq j \leq l$, if $p_j$ is in $b_i$ then $A(p_j) = \text{false}$ and if $\overline{p_j}$ is in $b_i$ then $A(p_j) = \text{true}$. Hence $A$ does not satisfy $b_i$ so $A$ does not satisfy $\beta$. Because $f(y_1: \tau_1, ..., y_i: \tau_i)$ is an arbitrary term where $y_1: \tau_1, ..., y_i: \tau_i$ are distinct variables of sort $T$ or $F$, and $V$ is a bijection ($V$ maps every truth assignment over $p_1, ..., p_i$ to some such term), there is no truth assignment that satisfies $\beta$.  

To close this chapter, we note the following. If every pair of (possibly sorted) terms has a least variable generalization and if no functions are polymorphic, according to a given sort theory $\Sigma$, then we say that $\Sigma$ meets the *monomorphic tree restriction*. The Sorted Anti-Unification Algorithm always computes a least sorted generalization, or a singleton $CIG_{\geq \Sigma}$, if $\Sigma$ meets this restriction. If queries of the sort theory are answered efficiently, then the algorithm computes the $CIG_{\geq \Sigma}$ efficiently. It might seem that the best course for further study is to seek restrictions such as the monomorphic tree restriction that guarantee efficient sorted anti-unification. The next chapter shows that we have another, better option.
Chapter 6

Computing Simple Constrained Anti-Unification

6.1 Preliminaries

In some cases anti-unification becomes easier if we broaden the language of acceptable generalizations, even if we also broaden the language of expressions to be generalized. This occurs when we extend the languages used in sorted anti-unification to simple constrained atoms and equality-free constraint theories with a finite number of constraint predicates. This extension yields efficient anti-unification, which leads to positive results for knowledge-base vivification, inductive learning, and speed-up learning.

From Observation 3 and the reasoning used in the previous chapter, we have the following results about simple constrained atoms, provided $\Sigma$ is equality-free and uses a finite set of constraint predicates.

**Lemma 87** Any $CIG_{\geq \Sigma}$ of a set of simple constrained atoms can be obtained from another by uniformly renaming variables and constraints.\(^1\)

**Lemma 88** Let $\Sigma$ be an equality-free constraint theory that uses only finitely many constraint predicates. Every $CIG_{\geq \Sigma}$ of a set of simple constrained atoms is finite.

\(^1\)By renaming constraints, we mean replacing a constraint by a $\Sigma$-equivalent constraint.
Proof: This lemma follows from Observation 3 and Corollary 66 (Chapter 4), which states that any set of equivalence classes of simple constrained atoms has only finitely many mubs. \hfill \Box

**Lemma 89** A set of simple constrained atoms has an empty \( CIG_{\Sigma} \) if and only if it has members built from different ordinary predicates.

**Proof:** If: A set of simple constrained atoms that has members built from different ordinary predicates has no \( \Sigma \)-generalizations, for any \( \Sigma \). Only If: Any set of simple constrained atoms built from the same ordinary predicate has at least one \( \Sigma \)-generalization. From Lemma 61, the set of \( \Sigma \)-generalizations is finite, up to the renaming of variables and constraints. It follows that the set has a nonempty \( CIG_{\Sigma} \). \hfill \Box

In fact, this chapter shows that any set of simple constrained atoms built from the same ordinary predicates has a least \( \Sigma \)-generalization, or an \( LGG_{\Sigma} \), where \( \Sigma \) is an equality-free constraint theory that uses a finite set of constraint predicates. In other words, such a set of simple constrained atoms has a singleton \( CIG_{\Sigma} \). Simple constrained anti-unification computes this \( LGG_{\Sigma} \). To see why any set of simple constrained atoms, built from the same ordinary predicate, has an \( LGG_{\Sigma} \), while this is not the case for the more specific class of sorted atoms, let’s rework Example 83 using simple constrained atoms.

**Example 90**

Let \( E \) be \( \{ p(\text{name}(clyde)), p(\text{name}(jumbo)) \} \)

Then an \( LGG_{\Sigma_1} \) of \( E \) is

\[
p(\text{name}(x))/\text{ELEPHANT}(x) \land \text{IN-CIRCUS}(x) \land \text{GRAY}(\text{name}(x))
\]

The \( CIG_{\Sigma_1} \) in Example 83 has multiple generalizations for two reasons. The first reason is that two minimal sorts, rather than one, contain both \textit{clyde} and \textit{jumbo}. Therefore
we obtain two minimal generalizations of *clyde* and *jumbo*, *x*:ELEPHANT and *z*:IN-CIRCUS, which lead to two minimal generalizations of \( p(\text{mom}(\text{clyde})) \) and \( p(\text{mom}(\text{jumbo})) \). Using simple constrained atoms, we can attach multiple sorts to a variable, thereby obtaining one minimal generalization of *clyde* and *jumbo*—a variable, *x*, belonging to both ELEPHANT and IN-CIRCUS sorts. Notice that both sorts, or constraints, are attached to *x* in Example 90. The second reason for multiple minimal generalizations in Example 83 is that both \( \text{mom}(\text{clyde}) \) and \( \text{mom}(\text{jumbo}) \) belong to the sort GRAY, while neither \( \text{mom}(x{:\text{ELEPHANT}}) \) nor \( \text{mom}(z{:\text{IN-CIRCUS}}) \) belongs to that sort. Therefore, the predicate \( p \) applied to a variable of the sort GRAY is a third minimal generalization. Using simple constrained atoms, we can always place stronger constraints on an entire non-variable term to make it more specific, with respect to the sort theory, than any given variable. We do so in Example 90 by constraining \( \text{mom}(x) \) to belong to the sort GRAY, thus making it more specific than a variable of sort GRAY.

### 6.2 A Simple Constrained Anti-Unification Algorithm

The presentation of the algorithm for simple constrained anti-unification requires an additional definition. Given an expression \( e_1 \) and a particular occurrence \( \gamma \) of a term in \( e_1 \), the Simple Constrained Anti-Unification (SCA) Algorithm needs to be able to select from another expression \( e_2 \) the term \( t \) that occurs at the same position in \( e_2 \) as does \( \gamma \) in \( e_1 \). We say that \( t \) is the term in \( e_2 \) corresponding to \( \gamma \) in \( e_1 \). For example, if \( e_1 \) is \( P(f(a),a) \) and \( e_2 \) is \( P(f(x),f(x)) \), then \( x \) is the term in \( e_2 \) corresponding to the first occurrence of \( a \) in \( e_1 \), and \( f(x) \) is the term in \( e_2 \) corresponding to the second occurrence of \( a \) in \( e_1 \). More precisely we define the term in \( e_2 \) corresponding to \( \gamma \) in \( e_1 \) to be \( t \) if and only if:

- \( \gamma \) is \( e_1 \) and \( t \) is \( e_2 \), or
Simple Constrained Anti-Unification (SCA) Algorithm

Input: A constraint theory, $\Sigma$, and a tuple of $m$ simple constrained atoms, $\phi_i/C_i$ for $1 \leq i \leq m$.
Output: An $LGG_{\geq_\Sigma}$ of $\langle \phi_1/C_1, \ldots, \phi_m/C_m \rangle$.

1. Let Head = Ordinary-Anti-Unification($\phi_1$, ..., $\phi_m$).
2. If Head = nil, then return nil.
3. Initialize Constraint-Set to the empty set.
4. Let $r$ be the maximum of the arities of all constraint predicates in $\Sigma$ or any $C_i$.
   For each $1 \leq i \leq r$, where some constraint predicate in $\Sigma$ or any $C_i$ has arity $l$,
   and each tuple $\langle u_1, \ldots, u_l \rangle$ of $l$ terms (not necessarily distinct) in Head:
   a. Let $\langle t_{i,1}, \ldots, t_{i,l} \rangle$ be the tuple of corresponding terms in $\phi_i$, for all $1 \leq i \leq m$.
   b. Let Atomic-Constraints contain exactly the atoms in the set
      $\{\tau(u_1, \ldots, u_l) \mid \Sigma \models \forall C_i (\tau(t_{i,1}, \ldots, t_{i,l}))$ for all $1 \leq i \leq m\}$.
   c. For each atom $\tau(u_1, \ldots, u_l)$ in Atomic-Constraints (in pre-specified order
      by constraint predicates):
      If another atom $\tau'(u_1, \ldots, u_l)$ remains in Atomic-Constraints, and
      $\tau'(u_1, \ldots, u_l) \Sigma \tau(u_1, \ldots, u_l)$, then
      Remove $\tau(u_1, \ldots, u_l)$ from Atomic-Constraints.
   d. Add to Constraint-Set the atoms in Atomic-Constraints.
5. Return the constrained atom whose head is Head and whose constraint is the conjunction
   of the atoms in Constraint-Set.

Figure 6.1: The Simple Constrained Anti-Unification (SCA) Algorithm

- $e_1$ is of the form $f(s_1, \ldots, s_n)$ and $e_2$ is of the form $f(u_1, \ldots, u_n)$, where $f$ is a
  function symbol or predicate, and $\gamma$ is in $s_i$ and $t$ is the term in $u_i$ corresponding
  to $\gamma$ in $s_i$.

If one occurrence of a term $t_1$ in an expression $e_1$ corresponds to a term $t_2$ in an
expression $e_2$, then so does every other occurrence of $t_1$ in $e_1$, provided $e_2$ is an instance
of $e_1$. In this case we sometimes speak of the term in $e_2$ that corresponds to a given
term, rather than term occurrence, in $e_1$. Furthermore, in general a term occurrence $\gamma$
in an expression $e_1$ may have no corresponding term $t$ in an expression $e_2$. But $\gamma$ in $e_1$
necessarily has a corresponding term $t$ in $e_2$ if $e_2$ is an instance of $e_1$. 

118
As already stated, we assume that $\Sigma$ uses a finite set of constraint predicates. We also assume, as we did for the Sorted Anti-Unification Algorithm, that the Simple Constrained Anti-Unification (SCA) Algorithm has a list of the constraint predicates in $\Sigma$ (for example, perhaps $\Sigma$ is finite and it has a copy of $\Sigma$). Furthermore, the SCA Algorithm requires information about certain consequences of $\Sigma$. Specifically, it requires knowledge of the constrained atomic consequences of $\Sigma$ (whether $\Sigma \models \forall(C \rightarrow \tau(t_1, \ldots, t_n))$, where $C$ is any conjunction of atoms built from constraint predicates, $\tau$ is any $n$-ary constraint predicate, and $t_1, \ldots, t_n$ are terms). If $\Sigma$ is a sort theory and $C$ is the constraint of a sorted atom, then the constrained atomic consequences are precisely the sorted atomic consequences of the previous chapter. Thus the queries about constrained atomic consequences are a natural extension of queries regarding sorted atomic consequences. In general these consequences are not decidable, but for many interesting constraint theories they are decidable. For all the example constraint theories we have used, they are efficiently decidable (time polynomial in the size of the query). We call queries about the constrained atomic consequences of a constraint theory constraint queries.

Without loss of generality, we assume that all simple constrained atoms given as input to the Simple Constrained Anti-Unification Algorithm are $\Sigma$-admissible. Testing whether a given simple constrained atom is $\Sigma$-inadmissible can be done efficiently, given an oracle for constraint queries. Therefore, the simple constrained atoms to be given to the algorithm may be tested first, and any $\Sigma$-inadmissible simple constrained atoms may be removed. If all are removed, the algorithm may return any one of them as an $LGG_{\geq \Sigma}$, since $\Sigma$-inadmissible constrained atoms are $\Sigma$-instances of all others.

The following examples show $LGG_{\geq \Sigma}$, the algorithm computes. The reader is encouraged to walk through the algorithm’s computations for these examples. The examples use the constraint theory $\Sigma_5$ (Chapter 3), which we repeat below.

$$
\Sigma_5 = \{ \textbf{bigger}(\text{son(jumbo),son(clyde)}), \textbf{bigger}(\text{son(fred),son(joe)}), \\
\textbf{bigger(jumbo,clyde)}, \textbf{bigger(fred,joe)}, \\
\textbf{elephant(clyde)}, \textbf{elephant(jumbo)}, \textbf{human(fred)}, \textbf{human(joe)}, \\
\forall x \forall y \textbf{elephant}(x) \land \textbf{human}(y) \rightarrow \textbf{bigger}(x, y) \}\n$$

119
Example 91

Let $E$ be $\{\text{intimidates}(\text{son(jumbo)}, \text{son(clyde)}), \text{intimidates}(\text{son(fred)}, \text{son(joe)})\}$

An $LGG_{\Sigma_5}$ of $E$ is:

$$\text{intimidates}(\text{son}(x), \text{son}(y)) / \text{BIGGER}(x, y) \land \text{BIGGER}(\text{son}(x), \text{son}(y))$$

Notice that the head of the $LGG_{\geq \Sigma_5}$ is the least ordinary generalization of the atoms in $E$. Notice also that the constraint of the $LGG_{\geq \Sigma_5}$ is the strongest possible constraint: strengthening the constraint yields a constrained atom that is not $\Sigma$-more general than both atoms in $E$.

Example 92

Let $E$ be $\{\text{intimidates}(\text{clyde}, y_1) / \text{HUMAN}(y_1), \text{intimidates}(x_1, \text{fred}) / \text{ELEPHANT}(x_1)\}$

An $LGG_{\Sigma_5}$ of $E$ is:

$$\text{intimidates}(x_2, y_2) / \text{BIGGER}(x_2, y_2) \land \text{ELEPHANT}(x_2) \land \text{HUMAN}(y_2)$$

Again note that the head of the $LGG_{\geq \Sigma_5}$ is the $LGG$ of the heads of the constrained atoms in $E$, and the constraint of the $LGG_{\geq \Sigma_5}$ is the strongest possible constraint. Notice, however, that in this example the constraint contains some redundancy. $\text{BIGGER}(x_2, y_2)$ could be omitted without weakening the constraint because of the following.

$$\Sigma_5 \models \forall x_2 \forall y_2 (\text{ELEPHANT}(x_2) \land \text{HUMAN}(y_2) \rightarrow \text{BIGGER}(x_2, y_2))$$

This redundancy does not affect the correctness of the computed $LGG_{\geq \Sigma_5}$. Step 4c of the algorithm eliminates redundancy among atoms built from the same tuple of terms in the constraint, but does not attempt to remove other forms of redundancy. To remove all redundancy in the constraint would make the problem intractable, in the same way that ensuring incomparability in the (compactly-represented) $CIG_{\Sigma}$ of a set of sorted atoms leads to intractability.
In order to discuss the time complexity of the algorithm, we need a measure of the size of a constrained atom. We have taken the size of an ordinary or sorted atom to be the number of variable and function symbol occurrences it contains. Since simple constrained atoms can have more complex constraints than sorted atoms, it is useful to distinguish between the head size of a constrained atom, which is the number of variable and function symbol occurrences in the head, and the total size of a constrained atom, which is the head size plus the number of variable, function symbol, and constraint predicate occurrences in the constraint.

**Theorem 93** Let $\Sigma$ be a constraint theory and let $\langle \phi_1/C_1, \ldots, \phi_m/C_m \rangle$ be a tuple of simple constrained atoms, such that the maximum of the arities of all constraint predicates in $\Sigma$ or any $C_i$, $1 \leq i \leq m$, is $r$, and any constraint query of size $s$, regarding $\Sigma$, is made and answered in time $O(s)$. (We assume that $Q$ is at least linear, since linear time would be required to communicate a query.) Let $P$ be the number of constraint predicates appearing in either $\Sigma$ or each $C_i$, $1 \leq i \leq m$. If any $\phi_i$ and $\phi_j$, $1 \leq i \neq j \leq m$, are built from different ordinary predicates, the algorithm returns nil in time $O(H^2)$, where $H$ is the sum of the head sizes of the constrained atoms. Otherwise, let $S$ be the sum of the sizes of the constrained atoms the algorithm receives, and let $k$ be the size of the smallest $\phi_i$, $1 \leq i \leq m$. Then the Simple Constrained Anti-Unification (SCA) Algorithm computes an $LGG_{\geq \Sigma}$ of $\{\phi_1/C_1, \ldots, \phi_m/C_m\}$ in time $O(rk^*((m + P)P(Q(S^2))))$.

**Proof:** We prove correctness and completeness of the computed $LGG_{\geq \Sigma}$, and then verify the time complexity.

**Correctness:** If $SCA(\phi_1/C_1, \ldots, \phi_m/C_m) = \phi/C$ (the result of applying the Simple Constrained Anti-Unification Algorithm to the tuple $\langle \phi_1/C_1, \ldots, \phi_m/C_m \rangle$ is $\phi/C$), correctness requires that $\phi/C \geq \Sigma \phi_i/C_i$ for all $1 \leq i \leq m$. We show this for arbitrary $\phi_i/C_i$. Since

$\phi = OA(\phi_1, \ldots, \phi_m)$ ( $\phi$ is the result of applying the Ordinary Anti-Unification Algorithm to the tuple $\langle \phi_1, \ldots, \phi_m \rangle$), we know there exists a substitution $\theta_i$ such that $\phi \theta_i = \phi_i$. Therefore, for each tuple $\langle u_1, \ldots, u_i \rangle$ of terms in the head, $\langle u_1, \ldots, u_i \rangle \theta_i = \langle t_{i,1}, \ldots, t_{i,i} \rangle$ where $\langle t_{i,1}, \ldots, t_{i,i} \rangle$ is the tuple of corresponding terms in $\phi_i$. It remains to show that
\( \Sigma \models \neg \forall(C_i \rightarrow C \theta_i) \). If this is not the case, then \( C \) contains some atom \( \alpha \) such that \( \Sigma \not\models \neg \forall(C_i \rightarrow \alpha \theta_i) \). Let \( \alpha \) be \( \tau(u_1, \ldots, u_l) \). Let \( \langle t_{i,1}, \ldots, t_{i,l} \rangle \) be the corresponding terms in \( \phi_i \). Then since \( \phi \theta_i = \phi \), we know that \( \langle u_1, \ldots, u_l \rangle \theta_i = \langle t_{i,1}, \ldots, t_{i,l} \rangle \theta_i \). By \( \tau(u_1, \ldots, u_l) \in C \), from step 4 of the CA Algorithm we know \( \Sigma \models \neg \forall(C_i \rightarrow \tau(t_{i,1}, \ldots, t_{i,l})) \), so \( \Sigma \models \neg \forall(C_i \rightarrow \alpha \theta_i) \).

\( \square \)

Completeness: Where \( \phi'/C' \geq \Sigma \phi_i/C_i \) for all \( 1 \leq i \leq m \), and \( SCA(\phi_1/C_1, \ldots, \phi_m/C_m) = \phi/C \), completeness requires that \( \phi'/C' \geq \Sigma \phi/C \). By Theorem 29, we know that \( \phi' \geq \phi \); for all \( 1 \leq i \leq m \). Since \( \phi \) is OA(\( \phi_1, \ldots, \phi_m \)), we know that \( \phi' \geq \phi \), so there is a substitution \( \theta \) such that \( \phi' \theta = \phi \). (In addition, there exist substitutions \( \theta_1, \ldots, \theta_m \) such that \( \phi' \theta_i = \phi_i \) for all \( 1 \leq i \leq m \).) We complete the proof by showing that \( \Sigma \models \neg \forall(C \rightarrow C') \).

Since \( \phi'/C' \geq \Sigma \phi_i/C_i \) for all \( 1 \leq i \leq m \), there exist substitutions \( \theta'_i \) such that \( \phi' \theta'_i = \phi_i \) and \( \Sigma \models \neg \forall(C_i \rightarrow C'_i \theta_i) \) for all \( 1 \leq i \leq m \). Without loss of generality, since \( \text{VARS}(C') \subseteq \text{VARS}(\phi') \), we have \( \Sigma \models \neg \forall(C_i \rightarrow C'_i \theta''_i) \) for all \( 1 \leq i \leq m \) and any substitution \( \theta''_i \) that agrees with \( \theta'_i \) on \( \text{VARS}(\phi') \). But since \( \phi' \theta \theta_i = \phi_i \) and \( \phi' \theta'_i = \phi_i \), we know \( \theta \theta_i \) agrees with \( \theta''_i \) on \( \text{VARS}(\phi') \), for all \( 1 \leq i \leq m \). Hence \( \Sigma \models \neg \forall(C_i \rightarrow C' \theta \theta_i) \) for all \( 1 \leq i \leq m \).

Assume \( \Sigma \not\models \neg \forall(C \rightarrow C' \theta) \). Then for some \( \alpha' \) in \( C' \), we have \( \Sigma \not\models (C \rightarrow \alpha' \theta) \). Let \( \alpha' \) be \( \tau(s_1, \ldots, s_l) \), so \( \alpha' \theta = \tau(s_1 \theta, \ldots, s_l \theta) \). We know that for all \( 1 \leq i \leq m \), \( \Sigma \models \neg \forall(C_i \rightarrow \alpha' \theta_i) \), so it is straightforward to verify that \( \tau \) must appear in \( \Sigma \) or in each \( C_i \). Let \( u_i \) also denote \( s_i \theta \) for all \( 1 \leq i \leq l \). Since \( \phi \theta = \phi \), and \( s_1, \ldots, s_l \) are terms in \( \phi' \), \( \langle u_1, \ldots, u_l \rangle \) is a tuple of terms in \( \phi \). Let \( \langle t_{i,1}, \ldots, t_{i,l} \rangle \) be the tuple of corresponding terms in \( \phi_i \), for all \( 1 \leq i \leq m \). Then since \( \tau(u_1, \ldots, u_l) \theta_i = \tau(t_{i,1}, \ldots, t_{i,l}) \) for all \( 1 \leq i \leq m \), we know that \( \tau(s_1 \theta \theta_i, \ldots, s_l \theta \theta_i) = \tau(t_{i,1}, \ldots, t_{i,l}) \) for all \( 1 \leq i \leq m \). And since \( \Sigma \models \neg \forall(C_i \rightarrow C' \theta \theta_i) \) for all \( 1 \leq i \leq m \), we have \( \Sigma \models \neg \forall(C \rightarrow \tau(s_1 \theta \theta_i, \ldots, s_l \theta \theta_i)) \) for all \( 1 \leq i \leq m \). Therefore, \( \Sigma \models \neg \forall(C \rightarrow \tau(t_{i,1}, \ldots, t_{i,l})) \) for all \( 1 \leq i \leq m \). Then from the statement of step 4 of the algorithm, \( C \) contains an atom \( \tau'(u_1, \ldots, u_l) \) where \( \tau' = \tau \) or \( \tau'(u_1, \ldots, u_l) \not\models \Sigma \tau(u_1, \ldots, u_l) \) \( (\Sigma \models \neg \forall(\tau'(u_1, \ldots, u_l) \rightarrow \tau(u_1, \ldots, u_l))) \). Hence \( \Sigma \models \neg \forall(C \rightarrow \tau(u_1, \ldots, u_l)) \), so \( \Sigma \models \neg \forall(C \rightarrow \alpha' \theta) \). Because \( \alpha \) was an arbitrary atom in \( C' \), we have \( \Sigma \models \neg \forall(C \rightarrow C' \theta) \). \( \square \)
Time Complexity: The time for computing $\phi$ by the ordinary anti-unification step is $O(H^2)$. Hence steps 1 through 3 are completed in time $O(H^2)$. The size of $\phi$ can be no greater than the size of the smallest $\phi_i$. Therefore, $\phi$ contains at most $k$ terms. For each tuple $\langle u_1, \ldots, u_l \rangle$ of $1 \leq l \leq r$ of these $k$ terms, the tuple $\langle t_{i,1}, \ldots, t_{i,m} \rangle$ of corresponding terms in any $\phi_i$ can be computed in time $O(H^2)$ as well.\footnote{In fact, the algorithm could compute all term correspondences once, in time $O(H^2)$, by storing $\phi$ and $\phi_1, \ldots, \phi_m$ in lists and establishing pointers from the first symbol in each term in $\phi$ to the first symbol in the corresponding term in each $\phi_i$. Then computing $\langle t_{i,1}, \ldots, t_{i,m} \rangle$ for each $\langle u_1, \ldots, u_l \rangle$ and $\phi_i$ usually could be done more efficiently.} Hence step 4a can be completed in time $O(mH^2)$. Step 4b may be implemented using $mP$ constraint queries: for each constraint predicate $\tau$ in $\Sigma$ or in each of $C_1, \ldots, C_m$ (there are at most $P$ of these), and for each $\phi_i/C_i$, $1 \leq i \leq m$, ask the oracle whether $\Sigma \models \forall(C_i \rightarrow \tau(t_{i,1}, \ldots, t_{i,m}))$, where $\langle t_{i,1}, \ldots, t_{i,m} \rangle$ is the tuple of terms in $\phi_i$ corresponding to $\langle u_1, \ldots, u_l \rangle$ in $\phi$. Each query has size at most $S + lH$, and so is answered in time $Q(S + lH)$. Each query can be built in time $O(S + lH)$. For simplicity, since $l$ is at most $H$ and $H$ is at most $S$, $S + lH$ is bounded by $O(S^2)$, so step 4b uses time $O(mP(Q(S^2)))$. The set produced by step 4b has size at most $P$. Step 4c may require checking each member of step 4b against every other member. This set has at most $P$ members, of size at most $lH \leq S^2$, so this step requires at most $P^2$ constraint queries of size at most $S^2$. Thus step 4c takes time $O(P^2(Q(S^2)))$. Step 4d can be implemented to take unit time. Thus the total time for the loop of step 4 is $O((m + P)P(Q(S^2)))$. The loop is repeated $rk^r$ times, so the total time for step 4 is $O(rk^r((m + P)P(Q(S^2))))$. Step 5 can be implemented to take time linear in the size of the resulting simple constrained atom, which is clearly bounded above by the time for step 4. Hence the entire algorithm can be implemented to run in time $O(rk^r((m + P)P(Q(S^2))))$.\hfill\□

Thus we see that, with a fixed bound on constraint predicate arity and with efficient response to constraint queries, simple constrained anti-unification can be computed efficiently. This indicates a significant advantage in using simple constrained atoms over using sorted atoms. But simple constrained anti-unification uses a stronger oracle, an oracle for constrained atomic consequences rather than sorted atomic consequences. Could
this be the reason for the improved performance? The following section addresses this
question and the relationship between simple constrained anti-unification and sorted
anti-unification.

6.3 Analysis: Extended Sorted Anti-Unification

In going from sorted atoms to simple constrained atoms, we extended the definitions
of the atoms in three ways. We allowed arbitrary terms to be constrained, rather than
variables only, we allowed each term to participate in multiple constraints, and we allowed
constraint predicates to have arity greater than one. Only the first two extensions are
necessary to ensure the existence of \(LGG_{\geq 2n}\)'s. The third extension was included for
expressiveness rather than efficiency. We refer to the class of atoms obtained by the
making only the first two extensions as extended sorted atoms, and we refer to the anti-
unification operation on these atoms as extended sorted anti-unification.

The Simple Constrained Anti-Unification Algorithm may be used for extended sorted
anti-unification as well. It is used with a constraint theory whose only constraint pred-
icates are unary, i.e., it is used with a sort theory. It follows that in such cases \(r\) (the
maximum of the arities of constraint predicates in \(\Sigma\)) is 1, so the algorithm runs in time
\(O(k(m + P)P(Q(S^2)))\), where \(k\) is the head size of the smallest extended sorted atom
being generalized, \(m\) is the number of extended sorted atoms being generalized, \(S\) is the
sum of the sizes of the atoms, \(P\) is the number of sorts in \(\Sigma\) or every extended sorted atom
given as input to the algorithm, and \(Q(s)\) is the maximum time required to answer
a query of size \(s\) about the sort theory. If the atoms given as input to the algorithm are all
sorted atoms, then it is straightforward to verify that the only constraint queries issued
are about the sorted atomic consequences of \(\Sigma\). Hence it is not the availability of more
powerful queries that allows efficient computation of anti-unification, but it is the more
general language. This more general language—specifically, the ability to attach sorts
to arbitrary terms, and the ability to attach multiple sorts to a term—guarantees the
existence of $LGG_{\geq s}$s, for any set of sorted atoms built from the same ordinary predicate. These $LGG_{\geq s}$s can be computed efficiently.

What we have seen, then, is that in extending the languages that describe the classes of anti-unification problems, we have not always extended the problem classes themselves. The ability to solve a given class of problems implies the ability to solve all its subclasses. But the ability to solve a class of anti-unification problems characterized by a language does not imply the ability to solve all classes of problems characterized by sublanguages. Figure 5 shows how the four classes of anti-unification problems we have studied thus far are related.

This analysis leads us to conclude that extended sorted atoms or simple constrained atoms should be used in place of sorted atoms, because they are more expressive and they admit more efficient reasoning mechanisms. Before reaching this conclusion, though, we should consider whether other typical operations on these atoms are also more efficient.
for extended sorted atoms and simple constrained atoms.\(^3\) The other common operations performed on atoms are (1) checking whether one atom is an instance of another, and (2) unifying a set of atoms, or finding their most general common instance(s). We consider these operations for simple constrained atoms; the results apply directly to extended sorted atoms as well.

We begin with a consideration of (1). Given the ability to determine efficiently the constrained atomic consequences of a constraint theory \(\Sigma\), it is straightforward to check efficiently whether one simple constrained atom, \(\phi_1/C_1\), is \(\Sigma\)-more general than a second, \(\phi_2/C_2\). From Theorem 29, we need only determine whether \(\phi_1 \geq \phi_2\), and, where \(\phi_1 \theta = \phi_2\) and \(\text{DOM}(\theta) \subseteq \text{VARS}(\phi_1)\), whether \(\Sigma \models \overline{\forall}(C_2 \rightarrow \alpha)\) for every atom \(\alpha\) in \(C_1\). The latter determinations are of constrained atomic consequences, and so can be made efficiently. It is straightforward to verify that this algorithm can be implemented to be at least as efficient as the Simple Constrained Anti-Unification Algorithm, given the same pair of simple constrained atoms and the same constraint theory. Notice that for extended sorted atoms, only sorted atomic consequences need to be determined.

We now consider (2). It is straightforward to verify that the most general common \(\Sigma\)-instance of simple constrained atoms \(\phi_1/C_1\) and \(\phi_2/C_2\) is \(\phi_1 \theta/C_1 \theta \land C_2 \theta\), where \(\theta\) is a most general unifier of \(\phi_1\) and \(\phi_2\). Hence, unlike finite sets of sorted atoms, any finite set of simple constrained atoms has a most general common instance, and it can be computed efficiently. In some cases it may be desirable to determine whether such a most general common \(\Sigma\)-instance is \(\Sigma\)-admissible. In general, this cannot be determined, but given the oracle for constraint queries it can be, as follows. Where \(\tau\) is some unary (any arity would work, in fact) constraint predicate that does not appear in \(\Sigma\) or in \(C_1 \theta\) or \(C_2 \theta\) and \(x\) is any variable, issue a constraint query asking whether \(\Sigma \models \overline{\forall}((C_1 \theta \land C_2 \theta) \rightarrow \tau(x))\). Because \(\tau\) does not appear in \(\Sigma\), \(C_1 \theta\), or \(C_2 \theta\), the answer to this query is \text{yes} if and only if \(C_1 \theta \land C_2 \theta\) is not \(\Sigma\)-satisfiable, that is, if and only if \(\phi_1 \theta/C_1 \theta \land C_2 \theta\) is \(\Sigma\)-inadmissible.

Thus it appears that the only reason one might wish to use sorted atoms rather than extended sorted atoms or simple constrained atoms, at least in any situation where anti-

\(^3\)I thank Leonard Pitt for pointing out the need for this argument.
unification will be used, is if sorted atomic consequences can be computed efficiently while extended sorted atomic consequences or constrained atomic consequences cannot. It is difficult to imagine a practical setting in which sorted atomic consequences can be computed efficiently while extended sorted atomic consequences cannot. Chapter 8 further develops the arguments about advantages of simple constrained atoms and extended sorted atoms over sorted atoms.

6.4 VC-Dimension of Simple Constrained Atoms

Having determined that Simple Constrained Anti-Unification always produces an $LGG_{\Sigma}$, we can now observe, as we did in Chapter 4 for ordinary atoms, that the VC-Dimension of any restricted class of simple constrained atoms is equal to the size of the largest chain of such atoms. Therefore, let $\Sigma$ be a constraint theory that contains $K$ constraint predicates, the maximum of whose arities is $r$. The VC-dimension of simple constrained atoms, ordered by $\geq_{\Sigma}$, with head size at most $n$ and with constraints that use at most $L$ constraint predicates that do not appear in $\Sigma$, the maximum of whose arities is $q$, is at most $Kn^r + Ln^q + n + 1$. Notice that for extended sorted atoms, where $\Sigma$ is a sort theory, the VC-dimension is at most $Kn + Ln + n + 1$. 
Chapter 7

E-Anti-Unification and
E-Anti-Unification with Constraints

In this chapter, we allow equality in the constraint theory $\Sigma$. Therefore, the algorithms in this chapter are based primarily on the characterization of the $\Sigma$-more general ordering in Theorem 33. For this reason, we assume throughout this chapter that the constraint theory $\Sigma$ is in Skolem Normal Form and has an initial model.

7.1 $E$-Anti-Unification

We begin with consideration of the anti-unification of ordinary atoms with respect to an equational theory—a constraint theory that uses only the equality predicate. We further specify that any equational theory is in Skolem Normal Form and has an initial model. This operation is $E$-anti-unification, the dual of $E$-unification. We have already given (Chapter 4) the following example, which indicates that $E$-anti-unification will not always return an $LGG_{\Sigma_M}$. Let $\Sigma_M$ be $\{g(a, b, b) = g(a, a, b), g(c, d, d) = g(c, c, d)\}$. Then $g(a, b, b)$ and $g(c, d, d)$ have no $LGG_{\Sigma_M}$, but they have a $CIG_{\Sigma_M}$, whose members are $g(x, y, y)$ and $g(x, x, y)$. More dramatically, a $CIG_{\Sigma}$ may not even exist, in contrast to the other forms of anti-unification we have examined. (Note that this is different from saying that the $CIG_{\Sigma}$ is empty.) For example, consider the following constraint theory.
\[ \Sigma_N = \{a = f(a), b = f(b), \forall x \forall y ((x = f(y)) \to (x = f(f(y)))) \} \]

\(\Sigma_N\) is a Horn clause theory with equality (more specifically, a definite clause theory with equality), so it is a Skolem Normal Form theory that has an initial model. Therefore, it meets our definition of an equational theory. Then the following is an infinite descending chain of ordinary atoms, ordered by \(\Sigma_N\), such that each is \(\Sigma_N\)-more general than both \(p(a)\) and \(p(b)\): \(\langle p(x), p(f(x)), p(f(f(x))), \ldots \rangle\). No atoms other than those in this chain (up to the renaming of variables) are \(\Sigma_N\)-more general than both \(p(a)\) and \(p(b)\). It follows that, while many atoms are \(\Sigma_N\)-more general than both \(p(a)\) and \(p(b)\), the set consisting of \(p(a)\) and \(p(b)\) has no \(CIG_{\geq \Sigma_N}\).

Therefore, in general \(E\)-anti-unification cannot be well-defined in the manner analogous to the other forms of anti-unification that we have studied thus far. We might define \(E\)-anti-unification to compute a \(CIG_{\geq \Sigma}\) when one exists. But such a definition encounters another difficulty: other \(CIG_{\geq \Sigma}\)s may be infinite. For example, consider the following constraint theory.

\[ \Sigma_O = \{ g(a, b) = g(f(a), f(b)), g(a, c) = g(f(a), f(c)), \]  
\[ \forall x \forall y ((g(x, y) = g(f(x), f(y))) \to (g(x, y) = g(f(f(x)), f(f(y)))) \} \]

Notice that this theory is also a definite clause theory (with equality), so it is a Skolem Normal Form theory that has an initial model. The set consisting of the atoms \(g(a, b)\) and \(g(a, c)\) has an infinite \(CIG_{\geq \Sigma_O}\): \(\{g(a, x), g(f(a), f(x)), g(f(a), f(f(x))), \ldots \}\). Therefore, the \(CIG_{\geq \Sigma_O}\) cannot be computed in full.

The next approach we might take is to enumerate the members of a \(CIG_{\geq \Sigma}\), if the \(CIG_{\geq \Sigma}\) exists. But the problem with this approach is in ensuring incomparability. How can we be sure that the next atom we obtain as, supposedly, a member of the \(CIG_{\geq \Sigma}\) is not strictly \(\Sigma\)-more general than some other atom we might compute later?

Because of the difficulties discussed so far, we further restrict our consideration to equational theories in which any term is \(\Sigma\)-equivalent to at most finitely many other
terms. In this case, a $CIG_{\Sigma}$ always exists, is finite, and is nonempty for any set of atoms built from the same predicate. To see this, we present a very simple algorithm for $E$-anti-unification under these circumstances. The algorithm assumes an oracle to provide the set of terms that are $\Sigma$-equivalent to a given term $t_1$. (Recall that terms $t_1$ and $t_2$ are $\Sigma$-equivalent if and only if for any $\Sigma$-model $M$ and value assignment $g$ we have $\llbracket t_1 \rrbracket^M g = \llbracket t_2 \rrbracket^M g$. In other words, $t_1$ and $t_2$ are $\Sigma$-equivalent if and only if $\Sigma \models \bar{v}(t_1 = t_2)$.) This oracle ensures the computability of $E$-transforms (Chapter 3): for any ordinary atom, $\phi$, the $E$-transforms of $\phi$ are exactly the terms that result from replacing any number of term occurrences in $\phi$ by occurrences of $\Sigma$-equivalent terms. For any set $\{\phi_1, \ldots, \phi_m\}$ of atoms, compute a set $S$ consisting of the $LGG$ (found by ordinary anti-unification) of each distinct set of atoms $\{\phi'_1, \ldots, \phi'_m\}$, where for all $1 \leq i \leq m$, $\phi'_i$ is an $E$-transform $\phi_i$. Then, for each atom $\alpha_1$ in $S$ (in any pre-specified order), if $\alpha_1$ is $\Sigma$-more general than some other atom remaining in $S$, remove $\alpha_1$ from $S$. (Recall from Theorem 33 that for any other atom $\alpha_2$, $\alpha_1 \geq_\Sigma \alpha_2$ if and only if $\alpha_2$ has an $E$-transform $\alpha'_2$ such that $\alpha_1 \geq \alpha'_2$. Given the oracle assumed by the algorithm, this can be tested.) Based on Theorem 33, it is straightforward to verify that this algorithm halts, given the specified oracle, and returns a $CIG_{\Sigma}$ (a set of atoms that is correct, complete, and contains no comparable members).

If the size or number of terms that are $\Sigma$-equivalent to a given term may be arbitrarily large, then the algorithm may take arbitrarily long. Any more detailed analysis of the run time of the algorithm requires restrictions on the sizes and number of such terms. At the end of this section, we consider the time complexity of the algorithm for one choice of such restrictions.

Let’s consider two examples of simple equational theories for which the requirements of the algorithm are met. First, let $\Sigma_P$ be $\{\forall x \forall y \ (f(x, y) = f(y, x))\}$. $\Sigma_P$ says that the function $f$ is commutative. Any term $t$ is $\Sigma_P$-equivalent to at most finitely many other terms, and these terms may be obtained by re-ordering the arguments to occurrences of $f$. For example, $p(f(a, f(x, c)))$ is $\Sigma_P$-equivalent to the terms $p(f(a, f(c, x)))$, $p(f(f(x, c), a))$, and $p(f(f(c, x), a))$, but no others.
For the second example, we consider a similar equational theory; in Chapter 8 we will see that this theory is quite useful for applications. In this theory, the symbol set is a binary function symbol that is used to represent sets in the same way that cons is used to represent lists. For example, the set \{1, 2, 3\} may be represented by set(1, set(2, set(3, nil))) or set(3, set(1, set(2, nil))). A set containing these elements and possibly others may be represented by set(3, set(1, set(2, x))). We use the following equational theory about sets.

$$\Sigma_5 = \{\forall x \forall y \forall z \,(\text{set}(x, \text{set}(y, z)) = \text{set}(y, \text{set}(x, z)))\}$$

\(\Sigma_5\) allows re-ordering of the elements in the set, in contrast to lists built from cons. Any term \(t\) is \(\Sigma_5\)-equivalent to at most finitely many other terms, and these terms may be obtained by the obvious re-ordering. For example, the term set(1, set(2, set(3, nil))) is \(\Sigma_5\) equivalent to the following terms but no others:

- set(1, set(3, set(2, nil)))
- set(2, set(1, set(3, nil)))
- set(2, set(3, set(1, nil)))
- set(3, set(1, set(2, nil)))
- set(3, set(2, set(1, nil)))

Now let’s consider three examples of E-anti-unification with respect to \(\Sigma_5\).

**Example 94**

Let \(E = \{p(\text{set}(1, \text{set}(3, \text{set}(2, \text{nil}))))\), \(p(\text{set}(2, \text{set}(3, \text{set}(1, \text{nil}))))\). Then \(E\) has a singleton \(\text{CIG}_{\Sigma_5}^E\):

$$\{p(\text{set}(1, \text{set}(3, \text{set}(2, \text{nil}))))\}$$

The different rewrites of the two atoms in \(E\) yield different \(\Sigma_5\)-generalizations, such as \(p(\text{set}(x, \text{set}(y, \text{set}(z, \text{nil}))))\), but the one in the \(\text{CIG}_{\Sigma_5}^E\) is a \(\Sigma_5\)-instance of all the others.
(There are other \(CIG_{\Sigma_3}\)s as well, which may be obtained by rewriting the atom given here according to \(\Sigma_5\).)

**Example 95**

Let \(E = \{p(set(1, set(2, set(3, nil))), p(set(3, set(4, set(5, nil))))\). Then \(E\) has a singleton \(CIG_{\Sigma_5}\): 

\[
\{p(set(3, set(x, set(y, nil))))\}
\]

Again, the different rewritings of the two atoms in \(E\) yield different \(\Sigma\)-generalizations, such as \(p(set(x, set(y, set(z, nil))))\), and \(p(set(x, set(3, set(y, nil))))\) but the one in the \(CIG_{\Sigma_5}\) is a \(\Sigma_5\)-instance of all the others. Notice that the \(CIG_{\Sigma_5}\) retains the information that both sets contain a 3 and that both sets consist of 3 elements (since the atom in the \(CIG_{\Sigma_5}\) ends in \(nil\), rather than in a variable, which would imply that the sets may have 3 or more elements).

**Example 96**

Let \(E = \{p(set(1, set(2, set(3, nil))), p(set(3, set(4, set(2, set(5, nil))))\). Then \(E\) has a singleton \(CIG_{\Sigma_5}\): 

\[
\{p(set(3, set(2, set(y, z))))\}
\]

Again, the different rewritings of the two atoms in \(E\) yield different \(\Sigma\)-generalizations, such as \(p(set(2, set(x, set(y, z))))\), but the one in the \(CIG_{\Sigma_5}\) is a \(\Sigma\)-instance of all the others. Notice that the \(CIG_{\Sigma_5}\) retains the information that both sets contain a 2 and a 3, and that both sets consist of at least 3 elements. (Since the atom in the \(CIG_{\Sigma_5}\) ends in a variable, it may have \(\Sigma\)-instances that represent sets of 4 or more elements as well.)

Before ending this section on \(E\)-anti-unification, it is worth noting that the two example equality theories given in this section have another interesting property. Not only
does any term have only finitely many \(\Sigma\)-equivalent terms with respect to these theories, but all \(\Sigma\)-equivalent terms have the same size and the same number of variables. It follows that the proofs of Theorem 54, Corollary 55, Corollary 56, and Corollary 57, about ordinary atoms (Chapter 4) ordered by \(\ge\), also apply to ordinary atoms ordered by \(\ge_{\Sigma_p}\) or \(\ge_{\Sigma_e}\). One particularly useful result is that, where \(\phi\) is an ordinary atom of size \(n\), the size of the largest ascending chain of atoms, ordered by \(\ge_{\Sigma_e}\), with \(\phi\) as the minimal element is \(n + 1\). Another result is that we can better analyze the time complexity of the \(E\)-Anti-Unification Algorithm. Given any term \(t\) of size \(n\), there are at most \((\lfloor \frac{n}{2} \rfloor)!\) terms that are \(\Sigma\)-equivalent to \(t\). (For example, if a term \(t\) of size \(n\) represents a set, the set it represents contains \(\lfloor \frac{n}{2} \rfloor\) elements, and the terms that are \(\Sigma_5\)-equivalent to \(t\) correspond to all orderings of the members of the set.) Therefore, any atom of size \(n\) has at most \((\lfloor \frac{n}{2} \rfloor)!\) \(E\)-transforms with respect to \(\Sigma\). Each of these atoms also has size \(n\). Therefore, given \(m\) atoms of size at most \(n\) each, the algorithm generates at most \((\lfloor \frac{n}{2} \rfloor)!^m\) sets of atoms, where each set contains atoms of size at most \(n\). Ordinary anti-unification for each set takes time \(O((nm)^2)\), so the time for all ordinary anti-unification operations is \(O((nm)^2((\lfloor \frac{n}{2} \rfloor)!)^m)\). These anti-unification operations may produce up to \((\lfloor \frac{n}{2} \rfloor)!^m\) \(LGGs\), which must be compared with one another. A comparison to see whether \(\alpha_1\) is \(\Sigma\)-more general than \(\alpha_2\) can be done by (1) generating all \(E\)-transforms \(\alpha_2'\) of \(\alpha_2\), and (2) testing whether any \(\alpha_2'\) is an instance of \(\alpha_1\). Step (1) can be done using the oracle and may produce at most \((\lfloor \frac{n}{2} \rfloor)!\) atoms. Because \(\alpha_1\) and each \(\alpha_2'\) have size at most \(n\), testing whether a particular \(\alpha_2'\) is an instance of \(\alpha_1\) can be completed in time \(O(n^2)\). Thus the total time for a comparison, given the oracle, is \(O(n^2((\lfloor \frac{n}{2} \rfloor)!))\), and the total time for all comparisons is \(O(((\lfloor \frac{n}{2} \rfloor)!)^m)\), which may be rewritten as \(O(n^2(((\lfloor \frac{n}{2} \rfloor)!)^{2m+1}))\). The time for comparisons is the dominant term in the complexity of \(E\)-anti-unification if \(m\) is at least 2 and for large enough \(n\), so the \(E\)-anti-unification algorithm runs in time \(O(n^2(((\lfloor \frac{n}{2} \rfloor)!)^{2m+1}))\).
7.2 Partitioned Constrained E-Anti-Unification

In this section, we define an operation called partitioned constrained E-anti-unification, which is a form of E-anti-unification for simple constrained atoms, and we identify a number of problems associated with its computation. Nevertheless, we show that for a restricted class of constraint theories and simple constrained atoms, it is computable by a simple algorithm.

The goal of partitioned constrained E-anti-unification is to compute the $CIG_{\geq \Sigma}$ of a set of simple constrained atoms with respect to a constraint theory $\Sigma$ that may contain equality. As we did for E-anti-unification, we assume that $\Sigma$ is in Skolem Normal Form and has an initial model. In addition, we require that $\Sigma$ be partitioned into a set of sentences that use only the equality predicate and a set of sentences that do not use the equality predicate. We call the first part the equational part of $\Sigma$ and the second part the equality-free part of $\Sigma$. (In general, when we say that $\Sigma$ is a partitioned constraint theory, we imply that it is also in Skolem Normal Form and has an initial model.) As we did for simple constrained anti-unification, we assume that $\Sigma$ uses only a finite set of constraint predicates, and we assume the existence of an oracle to answer constraint queries.

But every E-anti-unification problem is a partitioned constrained E-anti-unification problem, so some sets of simple constrained atoms have no $CIG_{\geq \Sigma}$, and other sets have an infinite $CIG_{\geq \Sigma}$. Therefore, once again we require that any term is $\Sigma$-equivalent to at most finitely many other terms, up to the renaming of variables. Recall that for simple constrained atoms, according to Theorem 33, E-transforms are taken with respect to a constraint $C$. Therefore, it might seem that we should require, more strongly, that where $C$ is any conjunction of atoms built from constraint predicates in $\Sigma$, for any term $t_1$ there are at most finitely many terms $t_2$ such that $\Sigma \models \forall t(C \rightarrow (t_1 = t_2))$. In fact, it is straightforward to verify that if $\Sigma$ is partitioned then $\Sigma \models \forall(C \rightarrow (t_1 = t_2))$ if and only if $\Sigma \models \forall(t_1 = t_2)$, provided $C$ is $\Sigma$-satisfiable. Any algorithm for partitioned constrained E-anti-unification may initially remove any simple constrained atoms with constraints that
are not $\Sigma$-satisfiable (this can be done efficiently using constraint queries). Therefore, the requirement that any term is $\Sigma$-equivalent to at most finitely many other terms is sufficient.

The obvious algorithm for partitioned constrained $E$-anti-unification would mimic the algorithm for $E$-anti-unification, using the Simple Constrained Anti-Unification Algorithm in place of ordinary anti-unification. It would proceed as follows. (Without loss of generality, we assume that $\Sigma$-inadmissible simple constrained atoms have been removed already.) Let $\Sigma'$ be the equality-free part of $\Sigma$. For any set $\{\phi_1/C_1, \ldots, \phi_m/C_m\}$ of simple constrained atoms, compute a set $S$ consisting of the $LGG_{\Sigma'}$, s (found by simple constrained anti-unification) of the sets $\{\phi'_1/C_1, \ldots, \phi'_m/C_m\}$, where for all $1 \leq i \leq m$, $\phi'_i/C_i$ is an $E$-transform of $\phi_i/C_i$ with respect to $\Sigma$. Then, for each atom $\phi/C$ in $S$ (in any pre-specified order), if $\phi/C$ is $\Sigma$-more general than some other atom remaining in $S$, remove $\phi/C$ from $S$. A comparison to see whether a given simple constrained atom, $\phi_1/C_1$, is $\Sigma$-more general than another, $\phi_2/C_2$, can be done by (1) generating all the $E$-transforms $\phi'_2/C_2$ of $\phi_2/C_2$, with respect to $\Sigma$, and (2) testing whether $\phi_1/C_1$ is $\Sigma'$-more general than any $\phi'_2/C_2$. Therefore, given the oracles assumed by the algorithm, $\Sigma$-instances can be tested. We will continue to refer to this algorithm as the obvious algorithm for partitioned constrained $E$-anti-unification.

But this algorithm contains an error. Specifically, an $E$-transform $\phi'_i/C_i$ of $\phi_i/C_i$ may not be a simple constrained atom, since $C_i$ may contain some term $t$ that appears in $\phi_i$ but not $\phi'_i$. Therefore the Simple Constrained Anti-Unification Algorithm is inappropriate. An obvious solution to this problem is to rewrite each $C_i$ to $C'_i$, where $C'_i$ results from the following: for every term $t$ in $\phi_i$ that does not appear in $\phi'_i$, replace all occurrences of $t$ in $C_i$ with any term $t'$ that is $\Sigma$-equivalent to $t$ and appears in $\phi'_i$. This rewriting would yield sets of simple constrained atoms, to which the Simple Constrained Anti-Unification Algorithm could be applied. But this approach is misguided, because in fact $\phi'_i$ may not contain any term that is $\Sigma$-equivalent to $t$. For example, let $\Sigma_O$ be $\{f(a) = f(b)\}$, and consider the simple constrained atom $p(f(a))/c(a)$. Then $p(f(b))/c(a)$ is an $E$-transform of this atom, with respect to $\Sigma_O$, yet there is no term in $p(f(b))$ that is $\Sigma_O$-equivalent to

135
a. Therefore, more generally, there is no *simple* constrained atom with the head $p(f(b))$ that is $\Sigma_0$-equivalent to $p(f(b))/c(a)$.

One approach we might take is to further restrict $\Sigma$ in some way such that if any constrained atom $\phi'/C$ is $\Sigma$-equivalent to a *simple* constrained atom then it is $\Sigma$-equivalent to a *simple* constrained atom with the head $\phi'$. A specific requirement that gives this result is the following: if two terms $t_1$ and $t_2$ are $\Sigma$-equivalent, then for each subterm of $t_1$ there exists a $\Sigma$-equivalent subterm of $t_2$, and vice-versa. It is straightforward to verify that this requirement gives the desired property. As a positive example, notice that the equality theory $\Sigma_P$, stating that the binary function $f$ is commutative, meets this requirement. Therefore, *any* constraint theory whose equational part is $\Sigma_P$ meets this requirement. As a negative example, notice that the equality theory $\Sigma_5$ fails to meet this requirement. It is straightforward to verify that if $\Sigma$ meets the given requirement, then the obvious algorithm for partitioned constrained $E$-anti-unification halts, given the specified oracles, and returns a $CIG_{\Sigma}$.  

The failure of $\Sigma_5$ to meet the requirement of the preceding paragraph is unfortunate because, as noted in the previous section, Chapter 8 shows that $\Sigma_5$ is particularly useful for some applications. We now examine a combination of restrictions on the constraints and constraint theory that allows us to work with $\Sigma_5$. These restrictions are specifically for $\Sigma_5$ and the set function symbol, although they can be generalized to apply to a broader class of constraint theories. Let $\Sigma$ be any partitioned constraint theory whose equational part is $\Sigma_5$ and whose equality-free part, $\Sigma'$, does not use the set function symbol. Let  

\[ \{ \phi_1/C_1, ..., \phi_m/C_m \} \]

be any set of simple constrained atoms where $C_1, ..., C_m$ do not use the set function symbol. Then, where $\phi_i'$ is any $E$-transform of $\phi_i$, with respect to $\Sigma$, $\phi_i'/C_i$ is a simple constrained atom as well. To see this, notice that any term $t$ in $\phi_i$ is $\Sigma$-equivalent to another, distinct term $t'$ only if $t$ uses the set function symbol. But it is straightforward to verify that any subterm of $t$ that does not contain the set function symbol also appears in $t'$, and vice-versa. It follows that, because $C_i$ uses only terms that appear in $\phi_i$ and do not contain the set function symbol, $C_i$ uses only terms that appear in $\phi_i'$ and do not use the set function symbol. Therefore, $\phi_i'/C_i$ is a simple constrained
atom, as claimed. Furthermore, if \( \phi / C \) is the \( LGG_{\geq \Sigma} \) of any set of simple constrained atoms whose constraints do not use the set function symbol, then \( C \) does not use the set function symbol. Theorem 97, below, follows from these observations.

**Theorem 97** Let \( \Sigma \) be any partitioned constraint theory whose equational part is \( \Sigma_{\Sigma} \) and whose equality-free part, \( \Sigma' \), does not use the set function symbol. Let \( \{ \phi_1 / C_1, ..., \phi_m / C_m \} \) be any set of simple constrained atoms where \( C_1, ..., C_m \) do not use the set function symbol. Let \( n \) be the size of the largest of \( \phi_1, ..., \phi_m \), and let \( k \) be the size of the smallest. Let \( S \) be the sum of the sizes of \( \phi_1 / C_1, ..., \phi_m / C_m \). Let \( r \) be the maximum of the arities of all constraint predicates in \( \Sigma' \) or each \( C_i \), \( 1 \leq i \leq m \), and let \( P \) be the number of constraint predicates appearing in either \( \Sigma' \) or each \( C_i \), \( 1 \leq i \leq m \). Let any constraint query of size \( s \), regarding \( \Sigma' \), be made and answered in time \( Q(s) \). (\( Q \) is at least linear, since linear time would be required to communicate a query.) Then the obvious algorithm for partitioned constrained \( E \)-anti-unification halts in time \( O(\frac{n}{r}!)(\frac{n}{r}!)^{2m+1}(rk^r((m + P) P (Q(S^2)))) \) and returns a \( CIG_{\geq \Sigma} \) of \( \{ \phi_1 / C_1, ..., \phi_m / C_m \} \).

**Proof:**

**Correctness:** Correctness requires that for any \( \phi / C \) returned by the algorithm, \( \phi / C \geq \Sigma \phi_i / C_i \) for each \( \phi_i / C_i \) provided as input to the algorithm. By the correctness of the Simple Constrained Anti-Unification Algorithm, \( \phi / C \geq \Sigma \phi'_i / C_i \), where \( \phi'_i / C_i \) is some \( E \)-transform of \( \phi_i / C_i \) with respect to \( \Sigma \), and therefore \( \phi / C \geq \Sigma \phi_i / C_i \). Then from Theorem 33, \( \phi / C \geq \Sigma \phi_i / C_i \). \( \Box \)

**Completeness:** Completeness requires that for any \( \phi' / C' \) that is \( \Sigma \)-more general than each \( \phi_i / C_i \) provided as input to the algorithm, \( \phi' / C' \) is \( \Sigma \)-more general than some \( \phi / C \) that the algorithm returns as output. If \( \phi' / C' \) is \( \Sigma \)-more general than each \( \phi_i / C_i \), then by Theorem 33 there exists, for all \( 1 \leq i \leq m \), an \( E \)-transform \( \phi'_i / C_i \) (with respect to \( \Sigma \)) of \( \phi_i / C_i \) such that: for some substitution \( \theta_i \), we have \( \phi' \theta_i = \phi'_i \) and \( \Sigma \models \exists x(C_i \rightarrow C' \theta_i) \). Notice that because \( \Sigma \) is partitioned, more specifically there exists, for all \( 1 \leq i \leq m \), an \( E \)-transform \( \phi'_i / C_i \) (with respect to \( \Sigma_{\Sigma} \)) of \( \phi_i / C_i \) such that: for some substitution \( \theta_i \), we have \( \phi' \theta_i = \phi'_i \) and \( \Sigma' \models \exists x(C_i \rightarrow C' \theta_i) \). Notice that \( \{ \phi'_1 / C_1, ..., \phi'_m / C_m \} \) is one of
the sets that the algorithm provides as input to the Simple Constrained Anti-Unification Algorithm. Let \( \phi/C \) be the \( LGG_{\geq \Sigma} \), returned by the Simple Constrained Anti-Unification Algorithm with this input. Then, because \( \Sigma' \) is equality-free, by the definition of \( LGG_{\geq \Sigma} \) for simple constrained atoms and by Theorem 29 we know that: if for all \( 1 \leq i \leq m \) there exists a substitution \( \theta_i \) such that \( \phi'\theta_i = \phi'_i \) and \( \Sigma' \models \overline{\theta}(C_i \rightarrow C'\theta_i) \) (the characterization of \( \geq \Sigma \) in Theorem 29), then there exists a substitution \( \theta \) such that \( \phi'\theta = \phi \) and \( \Sigma' \models \overline{\theta}(C \rightarrow C'\theta) \). Therefore \( \phi'/C' \geq \Sigma' \phi/C \), so \( \phi'/C' \geq \Sigma \phi/C \). Notice that \( \phi/C \) itself may not be returned by the algorithm even though it is an \( LGG_{\geq \Sigma} \), returned by the Simple Constrained Anti-Unification Algorithm; but this is the case only if \( \phi/C \) is \( \Sigma \)-more general than some other simple constrained atom returned by the algorithm, which is necessarily a \( \Sigma \)-instance of \( \phi'/C' \).

Incomparability: Incomparability requires that no simple constrained atom returned by the algorithm is \( \Sigma \)-more general than another. Incomparability is obvious from the last step of the algorithm.

Time Complexity: Given any term \( t \) of size \( n \), there are at most \( ([\frac{n}{2}]!) \) terms that are \( \Sigma \)-equivalent to \( t \). Therefore, any simple constrained atom \( \phi/C \) of head size \( n \) has at most \( ([\frac{n}{2}]!) \) \( E \)-transforms \( \phi'/C \) with respect to \( \Sigma \), and these can be computed in time \( O(S^2(([\frac{n}{2}]!])) \) by re-ordering the arguments to occurrences of the \emph{set} function according to \( \Sigma_5 \). Each of these atoms also has head size \( n \). Therefore, given \( m \) simple constrained atoms of head size at most \( n \) each, the algorithm generates at most \( ([\frac{n}{2}]!)^m \) sets of simple constrained atoms. Note that each set contains atoms of head size at most \( n \). These sets can be computed in time \( O(mS^2(([\frac{n}{2}]!)^m)) \): in time \( O(S^2(([\frac{n}{2}]!))) \) all the \( E \)-transforms of one simple constrained atom can be computed (this is done for \( m \) simple constrained atoms), and in time \( O(mS(([\frac{n}{2}]!))^m) \) these can be written into the sets. Applying the Simple Constrained Anti-Unification Algorithm to each set takes time \( O(rk^m((m + P)P(Q(S^2)))) \), so the time for all simple constrained anti-unification operations is \( O((rk^m((m + P)P(Q(S^2))))(([\frac{n}{2}]!)^m)) \). These anti-unification operations may produce up to \( ([\frac{n}{2}]!)^m \) \( LGG_{\geq \Sigma} \)'s, which must be compared with one another. A comparison to see whether a given simple constrained atom, \( \phi_1/C_1 \), is \( \Sigma \)-more general than
another, $\phi_2/C_2$, can be done by (1) generating all the $E$-transforms $\phi'_2/C_2$, with respect to
$\Sigma$ (or, more precisely, $\Sigma_5$) of $\phi_2/C_2$, and (2) testing whether $\phi_1/C_1$ is $\Sigma'$-more general than
any $\phi'_2/C_2$. Step (1) may produce at most $(|\frac{n}{2}|)!$ simple constrained atoms and takes time
$O(S^2((|\frac{n}{2}|)!))$. Because $\Sigma'$ contains no equality, one comparison in step (2) can be done
as described at the end of Chapter 6, in time no greater than the time required for simple constrained anti-unification. Therefore, an upper bound on the time for all comparisons in
step (2) with a given choice of $\phi_1/C_1$ and $\phi_2/C_2$ is $O((|\frac{n}{2}|)! (r k^r ((m + P) P (Q(S^2))))).$
Because in the worst case all pairs of constrained atoms generated as $LGG_{\Sigma^n}, s$ are compared,
an upper bound on the total time for all comparisons is $O((((|\frac{n}{2}|)!)^m)^2 ((|\frac{n}{2}|)!)(r k^r ((m +
P) P (Q(S^2))))),$ which we may rewrite as $O((((|\frac{n}{2}|)!)^2m+1(r k^r ((m + P) P (Q(S^2))))).$ This term is the dominant term in the complexity of the algorithm, so the algorithm runs in
time $O((((|\frac{n}{2}|)!)^2m+1(r k^r ((m + P) P (Q(S^2))))).$ 

As an example of the algorithm, let $\Sigma_6$ be the partitioned constraint theory whose
equational part is $\Sigma_5$ and whose equality-free part consists of the following sentences
(some of which were in the theory $\Sigma_3$ of Chapter 2).

\texttt{legislature(us-house-of-representatives),}
\texttt{legislature(us-senate)}
\texttt{us-state(north-carolina), us-state(illinois), ...}
\texttt{represents(moseley-braun, illinois, us-senate),}
\texttt{represents(michel, illinois, us-house-of-representatives), ...}
\texttt{\forall x ((\exists y \texttt{represents}(x, y, us-senate)) \rightarrow \texttt{senator}(x)),}
\texttt{republican(dole), republican(gramm), republican(warner), ...}
\texttt{democrat(nunn), democrat(pell), democrat(biden), ...}
\texttt{\forall x (\texttt{represents}(x, virginia, us-senate) \rightarrow \texttt{southerner}(x)),}
\texttt{\forall x (\texttt{represents}(x, texas, us-senate) \rightarrow \texttt{southerner}(x)), ...}
\texttt{\forall x (\texttt{represents}(x, kansas, us-senate) \rightarrow \texttt{midwesterner}(x)),}
\texttt{\forall x (\texttt{represents}(x, illinois, us-senate) \rightarrow \texttt{midwesterner}(x)), ...}
Example 98

Let $E$ be the set consisting of the following pair of simple constrained atoms.

\[
\text{will-\text{oppose}(set(dole, set(gramm, nil)), clinton, z)}/\text{ACTION(z)} \land \text{INCREASES-TAXES(z)}
\]

\[
\text{will-\text{oppose}(set(nunn, set(warner, nil)), clinton, z)}/\text{ACTION(z)} \land \text{CHANGES-MILITARY(z)}
\]

Then the following simple constrained atoms constitute a $CIG_{\Sigma_6}$ of $E$:

\[
\text{will-\text{oppose}(set(x, set(y, nil)), clinton, z)}/\text{ACTION(x) \land SENATOR(x) \land SENATOR(y) \land }
\]

\[
\text{REPUBLICAN(x) \land SOUTHERNER(y)}
\]

\[
\text{will-\text{oppose}(set(x, set(y, nil)), clinton, z)}/\text{ACTION(x) \land SENATOR(x) \land SENATOR(y) \land }
\]

\[
\text{REPUBLICAN(x) \land SOUTHERNER(x)}
\]

The members of the $CIG_{\Sigma_6}$ are obtained from different orderings of the sets in the heads of the simple constrained atoms in $E$, and therefore from different matchings of the members in the sets. Each member of the $CIG_{\Sigma_6}$ says that two senators will oppose President Clinton on action $x$ (e.g., a budget bill or an executive order). The first member says that one senator is a Republican and the other is a Southerner. It is obtained by an ordering that matches nunn (of Georgia) with gramm (of Texas). The second member of the $CIG_{\Sigma_6}$ says that one senator is a Southern Republican but says nothing about the other senator. It is obtained by an ordering that matches warner (of Virginia) with gramm. Notice that neither member of the $CIG_{\Sigma_6}$ is $\Sigma_6$-more general than the other.

This example shows an important property of the use of the $\text{set}$ function symbol and the use of $\Sigma_5$ as the equational part of a partitioned constraint theory $\Sigma$. $\Sigma$-generalizations whose constraints differ in significant ways are obtained by different orderings of the sets. This is a particularly notable property of structural domains, such as the Blocks World, whose members (states) are characterized by a set of objects, or
individuals, and a set of relations among them. The following example is taken from the Blocks World with respect to a partitioned constraint theory, $\Sigma_7$, whose equational part is $\Sigma_5$ and whose equality free part is the following pair of sentences, which we call $\Sigma_7^e$.

$$\forall x \forall y \,(\text{ONTOP}(x, y) \rightarrow \text{ABOVE}(x, y))$$
$$\forall x \forall y \forall z \,(\text{ONTOP}(x, y) \land \text{ABOVE}(y, z) \rightarrow \text{ABOVE}(x, z))$$

**Example 99**

Let $E$ be the set consisting of the following pair of simple constrained atoms.

$$\text{apply}(\text{pickup}(a), \text{domain}(\text{arm}, \text{set}(a, \text{set}(b, \text{set}(c, \text{nil})))), \text{clear}(c)) / \text{ONTOP}(a, b) \land$$
$$\text{ONTOP}(b, c) \land \text{CLEAR}(a) \land \text{EMPTY}(\text{arm})$$

$$\text{apply}(\text{pickup}(e), \text{domain}(\text{arm}, \text{set}(d, \text{set}(e, \text{set}(f, \text{nil})))), \text{clear}(d)) / \text{ONTOP}(e, d) \land$$
$$\text{CLEAR}(e) \land \text{CLEAR}(f) \land \text{EMPTY}(\text{arm})$$

Then the following simple constrained atoms constitute a $CIG_{\geq \Sigma_6}$ of $E$:

$$\text{apply}(\text{pickup}(x), \text{domain}(\text{arm}, \text{set}(x, \text{set}(y, \text{set}(z, \text{nil})))), \text{clear}(z)) / \text{CLEAR}(x) \land$$
$$\text{ABOVE}(x, z) \land \text{EMPTY}(\text{arm})$$

$$\text{apply}(\text{pickup}(x), \text{domain}(\text{arm}, \text{set}(x, \text{set}(y, \text{set}(z, \text{nil})))), \text{clear}(w)) / \text{CLEAR}(x) \land$$
$$\text{ABOVE}(x, w) \land \text{ONTOP}(x, y) \land \text{EMPTY}(\text{arm})$$

$$\text{apply}(\text{pickup}(u), \text{domain}(\text{arm}, \text{set}(x, \text{set}(y, \text{set}(z, \text{nil})))), \text{clear}(w)) / \text{CLEAR}(u) \land$$
$$\text{ABOVE}(u, w) \land \text{ONTOP}(y, w) \land \text{EMPTY}(\text{arm})$$

Intuitively, the first member of $E$ may be viewed as stating that a pickup operation, applied to block $a$ is an optimal action, if the eventual goal is clear($c$), and the state contains blocks $a$, $b$, and $c$ where $a$ is clear (has no block on top of it) and is on top of $b$, $b$ is on top of $c$, and the robot arm is empty. (In a pickup operation, the robot arm picks up a specified block, provided the arm is empty and the block is clear. An action is
the application of an operator, and an optimal action is a first action in a shortest plan (shortest sequence of actions) that achieves the eventual goal.) The second member of $E$
may be viewed as saying that a pickup operation, applied to block $e$ is an optimal action, if the eventual goal is $\text{clear}(e)$, and the state contains blocks $d$, $e$, and $f$ where $e$ and $f$ are clear, $e$ is on top of $d$, and the robot arm is empty. The terms built from the set function in each of these examples may be written in six different ways, and the combinations of writings for the pair of terms yield 36 different pairs of simple constrained atoms; the Simple Constrained Anti-Unification Algorithm is applied to each of these, using $\Sigma'_7$, to yield 36 $LGG_{\Sigma'_7}$’s. When these are compared with one another, the three in the $CIG_{\Sigma'_7}$ above are seen to be the minimal ones—all others are $\Sigma'_7$-more general than at least one of these. The members of the $CIG_{\Sigma'_6}$ may be seen as providing more general descriptions of when an application of the pickup operation is optimal.

Before ending this chapter, we observe the following. Consider any equality-free constraint theory $\Sigma'$, and let $\Sigma$ be the result of adding to $\Sigma'$ the equational theory $\Sigma_5$. What is the effect of this addition on the partially-ordered set of equivalence classes of simple constrained atoms, ordered by $\geq_{\Sigma'}$? The result of changing from $\Sigma'$ to $\Sigma$ is to combine various equivalence classes. Specifically, two simple constrained atoms $\phi_1/C_1$ and $\phi_2/C_2$ are in the same equivalence class under $\geq_{\Sigma'}$ if and only if $\phi_1$ and $\phi_2$ are variants, and, where $\phi_1 \theta = \phi_2$, $C_1 \theta$ and $C_2$ are $\Sigma'$-equivalent. But two simple constrained atoms $\phi_1/C_1$ and $\phi_2/C_2$ are in the same equivalence class under $\geq_{\Sigma}$ if and only if $\phi_1$ and some $E$-transform, relative to $\Sigma$ (and therefore $\Sigma_5$), $\phi'_2$ of $\phi_2$ are variants, and, where $\phi_1 \theta = \phi'_2$, $C_1 \theta$ and $C_2$ are $\Sigma$-equivalent. Therefore, if $\phi_1/C_1$ and $\phi_2/C_2$ are in the same equivalence class under $\geq_{\Sigma}$, then they are in the same equivalence class under $\geq_{\Sigma}$. As a result of this observation, several results from Chapter 4 apply to this case of limited equality. In particular, it is straightforward to verify the following results, where $\Sigma$ is a partitioned constraint theory whose equational part is $\Sigma_5$ and whose equality-free part does not use the set function symbol. We also assume that no constraint may contain the set function symbol.
Lemma 100 The dominating set of any equivalence class of simple constrained atoms \([\phi/C]_{\geq \Sigma}\) is finite.

Corollary 101 There exists no infinite chain of equivalence classes of simple constrained atoms, ordered by \(\geq \Sigma\), between any pair of such equivalence classes, and there exists no infinite ascending chain of equivalence classes of simple constrained atoms.

Corollary 102 Let \(\phi_1/C_1\) be a simple constrained atom that is strictly \(\Sigma\)-more general than \(\phi_2/C_2\). Then any maximal chain between \(\phi_1/C_1\) and \(\phi_2/C_2\) includes exactly one simple constrained atom covered by \(\phi_1/C_1\) and exactly one simple constrained atom that covers \(\phi_2/C_2\).

Corollary 103 There exists no infinite descending chain of equivalence classes of simple constrained atoms whose members are all \(\Sigma\)-more general than some given class \([\phi/C]_{\geq \Sigma}\).

Corollary 104 The cover set of any equivalence class of simple constrained atoms is finite.

Corollary 105 Any set (finite or infinite) of equivalence classes, under \(\geq \Sigma\), of simple constrained atoms has only finitely many muds.

Theorem 106 Let \(K\) be the number of constraint predicates in \(\Sigma\), and let \(r\) be the maximum of the arities of these constraint predicates. Let \(\phi'/C'\) be any simple constrained atom, let \(n\) be the size of \(\phi'\), let \(L\) be the number of constraint predicates that appear in \(C'\) but not \(\Sigma\), and let \(q\) be the maximum of the arities of these constraint predicates. Then the size of the largest ascending chain of simple constrained atoms with minimal element \(\phi'/C'\) is at most \(Kn^r + Ln^q + n + 1\).
Chapter 8

Applications of Anti-Unification

In this chapter we examine applications of anti-unification to inductive logic programming, speed-up learning, and knowledge-base vivification.

8.1 Learnability Results for Inductive Logic Programming

Inductive logic programming (ILP) is a rapidly-growing area of research at the intersection of machine learning and logic programming [47]. It focuses on the design of algorithms that learn first-order Horn clause theories from examples. A natural framework for research in inductive logic programming is the investigation of the learnability/predictability of various classes of Horn clause theories in the widely-known models of pac-learnability [64] and learning by equivalence queries [3]. Until last year, surprisingly little work had been done within this framework. Interest now is rising sharply [13; 12; 11; 19; 21; 43; 47; 53; 6]. The results in the first two parts of this section were first presented two years ago [50; 51; 52] and were the first results for inductive logic programming in this framework. The result in the third part of this section is an extension to these results. The results in the final part of this section are, to my knowledge, the first results for inductive logic programming in this framework that apply to struc-
tural domains. These results are based on the properties of anti-unification that we have studied so far. Before presenting the results, we describe the learning models as they apply to Horn clause theories, or more generally any first-order theories, in the presence of background information.

Much research in recent years has been devoted to studying the learnability of propositional formulas, or concepts expressed in propositional logic. The examples provided to algorithms that learn concepts expressed in propositional logic traditionally have been truth assignments, or models. Such an example is positive if and only if it satisfies the concept. But concepts in first-order logic may (and almost always do) have infinite models. Therefore, algorithms that learn Horn clause theories typically take logical formulas, often ground atomic formulas, as examples instead. Such an example is positive if it is a logical consequence of the concept, and negative otherwise. In addition, such algorithms often take into account a background theory as well. When a background theory is present, an example is positive if it is a logical consequence of the concept together with the background theory and negative otherwise. This typical usage motivates the following definitions. ¹

Let \( \mathcal{C} \) and \( \mathcal{D} \) be classes of logical sentences, and let \( \mathcal{T} \) be a class of logical theories. Intuitively, \( \mathcal{C} \) is a concept class, \( \mathcal{D} \) is an instance class, and \( \mathcal{T} \) is a class of background theories. For any sentence \( \psi_1 \in \mathcal{C} \) and \( \psi_2 \in \mathcal{D} \), and any background theory \( \Sigma \in \mathcal{T} \), \( \psi_1 \) is said to cover \( \psi_2 \) with respect to \( \Sigma \) if and only if \( \psi_1 \Sigma \)-entails \( \psi_2 \). Two sentences of \( \mathcal{C} \) are coverage-equivalent with respect to \( \mathcal{D} \) and \( \Sigma \) if and only if they cover the same sentences of \( \mathcal{D} \) with respect to \( \Sigma \). (Notice that coverage-equivalence is different from logical equivalence or \( \Sigma \)-equivalence. Using logical equivalence or \( \Sigma \)-equivalence would create problems because in some cases it would require a learning algorithm to distinguish between two sentences in \( \mathcal{C} \) that agree on the labels they assign to all possible examples, that is, to all sentences in \( \mathcal{D} \).) A sentence \( \psi_2 \in \mathcal{D} \) is labeled as a positive example by a sentence \( \psi_1 \in \mathcal{C} \) and a background theory \( \Sigma \in \mathcal{T} \) if \( \psi_1 \) covers \( \psi_2 \) with respect to \( \Sigma \). Otherwise, \( \psi_2 \) is labeled as a negative example by \( \psi_1 \) and \( \Sigma \).

¹The definitions are based on standard definitions from computational learning theory [2; 3; 7].
**Definition 107** An equivalence query asks whether a particular sentence (the predicted sentence) of \( \mathcal{C} \) is coverage-equivalent to an unknown target sentence of \( \mathcal{C} \) with respect to a given instance class \( \mathcal{D} \) and background theory \( \Sigma \). An oracle for equivalence queries answers yes to any given query if the predicted sentence is coverage-equivalent to the target with respect to \( \mathcal{D} \) and \( \Sigma \); it answers no otherwise. The oracle gives a counterexample if the predicted sentence is not coverage-equivalent to the target with respect to \( \mathcal{D} \) and \( \Sigma \). A counterexample to an equivalence query is any sentence of \( \mathcal{D} \) that is labeled differently by the predicted and target sentences with respect to \( \Sigma \). (If the predicted and target sentences are not coverage-equivalent with respect to \( \mathcal{D} \) and \( \Sigma \), by the definition of coverage-equivalence a counterexample exists.)

**Definition 108** An algorithm \( A \) is a polynomial-time learning algorithm for a class \( \mathcal{C} \) of logical sentences, relative to a background theory \( \Sigma \), and with examples from a class \( \mathcal{D} \) of logical sentences, if and only if there exists a polynomial \( p(n, m) \) such that: for any target sentence \( \psi \in \mathcal{C} \), when \( A \) is run with an oracle to answer equivalence queries for \( \psi \) with respect to \( \mathcal{D} \) and \( \Sigma \), \( A \) halts and outputs a sentence \( \psi' \in \mathcal{C} \) that is coverage-equivalent to \( \psi \) with respect to \( \mathcal{D} \) and \( \Sigma \). Moreover, at any point during the run, the time used by \( A \) to that point is bounded by \( p(n, m) \), where \( n \) is the size of \( \psi \), and \( m \) is the size of the longest counterexample seen to that point. The class \( \mathcal{C} \) of logical sentences is polynomially learnable by equivalence queries, relative to the class \( \mathcal{T} \) of background theories, and with examples from \( \mathcal{D} \), if and only if there exists such an algorithm \( A \) for any \( \Sigma \in \mathcal{T} \).

**Definition 109** An algorithm \( A \) is a polynomial-time pac-learning algorithm for a class \( \mathcal{C} \) of logical sentences, relative to a background theory \( \Sigma \), and with examples from a class \( \mathcal{D} \) of logical sentences, if and only if \( A \) has the following property. For all \( 0 < \epsilon, \delta < 1 \) and \( n \geq 1 \), there exists a sample size \( m(\epsilon, \delta, n) \), polynomial in \( 1/\epsilon \), \( 1/\delta \), and \( n \), such that for all sentences \( \psi \) in \( \mathcal{C} \) that have size \( n \) or less, and all probability distributions, \( P \), on the sentences of \( \mathcal{D} \), given a random sample of \( m(\epsilon, \delta, n) \) sentences from \( \mathcal{D} \) drawn independently according to \( P \) and labeled according to \( \psi \) and \( \Sigma \), \( A \) produces, with probability at least \( 1 - \delta \), a sentence, \( \psi' \in \mathcal{C} \), such that the probability that a given sentence
drawn according to $P$ is labeled differently, with $\Sigma$, by $\psi$ and $\psi'$ is at most $\epsilon$. Moreover, the algorithm runs in time polynomial in the length of its input (the sum of the sizes of the examples in the sample). The class $C$ is pac-learnable, relative to the class $T$ of background theories, and with examples from $D$, if and only if for any $\Sigma$ in $T$ such an algorithm $A$ exists.

A well-known result of computational learning theory is that polynomial learning by equivalence queries implies pac-learning. More specifically, any algorithm that learns a concept class in polynomial time using equivalence queries can be used to pac-learn the concept class, provided concept membership can be tested efficiently, that is, provided a polynomial-time algorithm exists that determines whether a given concept classifies a given example as positive or negative (relative to a particular background theory) [3]. Concept membership can be tested efficiently for all the concept classes we consider in this chapter (see Section 6.3).

8.1.1 The Learnability of Sorted Atoms

The class of universally-closed sorted atoms is an interesting class of definite clauses: it is the class of non-recursive definite clauses such that the antecedent (body) of any such clause consists of only unary predicates applied to variables that appear in the consequent (head), where each variable is used at most once. This class of definite clauses is simple, and is quite similar to the concept classes used in some early machine learning programs, as noted in Chapter 5. If positive results can be obtained for learnability in inductive logic programming, then surely, it would seem, the class of universally-closed sorted atoms should be learnable. More precisely, this class should be learnable with respect to the class of sort theories for which the time to answer taxonomic queries is bounded by some polynomial in the size of the query. We begin this section with a study of this learning problem. In this problem $C$ and $D$ are both the class of universally-closed sorted atoms, and $T$ is the class of sort theories for which the time to answer taxonomic queries is bounded by some polynomial in the size of the query. In addition, we assume that the
sort theory uses a finite set of sort symbols. For convenience we will at times refer to universally-closed sorted atoms as, simply, *sorted atoms*, with the universal quantification implicit.

Applying one result of computational learning theory [54], we find that learning sorted atoms with respect to a given sort theory, $\Sigma$, in these models is possible only if a (possibly randomized, for pac-learning) polynomial-time algorithm for the following problem exists. Given a set of sorted atoms, each labeled as either a positive or negative example, determine whether there exists a sorted atom, $\alpha$, that covers, with $\Sigma$, all of the positive examples but none of the negative examples. Such a sorted atom is said to be *consistent with the examples*, and the problem of determining whether such a sorted atom exists is the *consistency problem* for sorted atoms.

Recall that for any sorted atoms $\alpha_1$ and $\alpha_2$, and any sort theory $\Sigma$, $\Sigma$ together with the universal closure of $\alpha_1$ entails the universal closure of $\alpha_2$ if and only if $\alpha_1 \supseteq_\Sigma \alpha_2$. Therefore, one sorted atom covers another, relative to $\Sigma$, if and only if the first is $\Sigma$-more general than the second. Thus the consistency problem for sorted atoms may also be viewed as determining whether a sorted atom exists that is $\Sigma$-more general than each of the positive examples but none of the negative examples. We have noted that sorted anti-unification requires exponential time, and that even with a compact representation no polynomial-time algorithm exists, assuming $P \neq NP$. The following result tells us that this complexity carries over to the consistency problem, and so to both pac-learning and learning by equivalence queries if we assume $R \neq NP$. Specifically, this result shows that the consistency problem for sorted atoms is NP-hard (more precisely, $NP$-complete), which implies that sorted atoms are not pac-learnable unless $R = NP$.

**Theorem 110** Sort theories $\Sigma$ exist for which any taxonomic query can be answered efficiently and yet the consistency problem for sorted atoms with respect to $\Sigma$ is NP-complete.

\footnote{That is, if we assume the $NP$-complete problems cannot be solved in random polynomial time.}
Proof: Given a set of sorted atoms, each labeled positive or negative, given a sorted atom \( \alpha \), and given efficient response to taxonomic queries about a given sort theory \( \Sigma \), checking whether \( \alpha \) is \( \Sigma \)-more general than all the positive sorted atoms and none of the negative sorted atoms can be done efficiently. Therefore the problem is in NP. We complete the proof by showing that 3SAT is polynomially reducible to the consistency problem for sorted atoms.

Consider the following sort theory.

\[
\Sigma_Q = \{ \tau(a), \tau(b), \tau(c), \omega(a), \omega(b), \omega(d), \forall x \ \text{UNIV}(x) \}
\]

Obviously any taxonomic query regarding \( \Sigma_Q \) can be answered efficiently, in fact in constant time. We use \( \Sigma_Q \) to transform (in polynomial time) any instance of 3SAT to an instance of the consistency problem for sorted atoms. Let \( \beta = b_1 \land \ldots \land b_k \) be an arbitrary 3-CNF formula over the propositional variables \( p_1, \ldots, p_m \). We build an instance of the consistency problem as follows.

Let the positive examples be the following.

Where \( p \) is an \( n \)-ary predicate, two positive example are \( p(a, \ldots, a) \) and \( p(b, \ldots, b) \).

The rest are:

\[
\{ p(t_{i,1}, \ldots, t_{i,n}) \mid 0 \leq i \leq \lfloor \log_2 n \rfloor - 1 \text{ and for all } 1 \leq j \leq n, t_{i,j} \text{ is } a \text{ if } \left\lfloor \frac{i}{2^j} \right\rfloor \text{ is odd and } b \text{ otherwise} \}
\]

Thus, for example, if \( n = 10 \) the positive examples are

\[
\begin{align*}
p(a, a, a, a, a, a, a, a) \\
p(b, b, b, b, b, b, b, b) \\
p(a, b, a, b, a, b, a, b) \\
p(a, a, b, a, b, a, b, a) \\
p(a, a, a, b, b, b, a, a) \\
p(a, a, a, a, b, b, a, b) \\
\end{align*}
\]
Any sorted atom is $\Sigma_Q$-more general than all of the positive examples if and only if it is of the form $p(t_1, \ldots, t_n)$ where the $t_i$, $1 \leq i \leq n$, are distinct variables of sort $\tau$, $\omega$, or $\text{univ}$.

We build the negative examples from $\beta$ as follows, with one negative example $n_i$ for each clause $b_i$ in $\beta$, $1 \leq i \leq k$. For each $1 \leq i \leq k$, let $p(s_1, \ldots, s_n)$ be a negative example such that for all $1 \leq j \leq n$: if the propositional variable $p_j$ does not appear in $b_j$ then $s_j = a$; if $p_j$ appears negated in $b_j$ then $s_j = c$; and if $p_j$ appears unnegated in $b_j$ then $s_j = d$. (If any variable appears negated and unnegated in $b_j$ then $b_j$ can be removed from $\beta$.) Thus, for example, if $\beta$ is $(p_1 \lor \overline{p_2} \lor p_3) \land (\overline{p_1} \lor p_2 \lor \overline{p_4})$, then $n = 4$, and the two negative examples are

\[
\begin{align*}
p(d, c, d, a) \text{ (from } p_1 \lor \overline{p_2} \lor p_3) \\
p(c, d, a, c) \text{ (from } \overline{p_1} \lor p_2 \lor \overline{p_4})
\end{align*}
\]

Clearly the positive and negative examples can be generated in polynomial time. We now verify that $\beta$ is satisfiable if and only if some sorted atom is consistent with the examples relative to $\Sigma_Q$.

**Consistent Sorted Atom $\Rightarrow$ $\beta$ Satisfiable:**

Suppose some sorted atom $p(t_1, \ldots, t_n)$ is consistent with the examples. Then, as noted earlier, the $t_i$ are distinct variables of sort $\tau$, $\omega$, or $\text{univ}$. Let $V$ be any truth assignment such that for all $1 \leq i \leq n$: $V(p_i) = 1$ (V makes $p_i$ true) if $t_i$ is a variable of sort $\tau$, and $V(p_i) = 0$ (V makes $p_i$ false) if $t_i$ is a variable of sort $\omega$. ($V(p_i)$ may be 0 or 1 if $t_i$ is a variable of sort $\text{univ}$.) For example, if $p(t_1, \ldots, t_n)$ is $p(y_1: \tau, y_2: \text{univ}, y_3: \omega, y_4: \omega)$, then $V(p_1) = 1$, $V(p_2)$ may be 0 or 1, $V(p_3) = 1$, and $V(p_4) = 0$.

We now verify that $V$ satisfies $\beta$. Recall that each negative example $n_i$ is of the form $p(s_1, \ldots, s_n)$ where, for all $1 \leq j \leq n$, $t_j$ is $a$, $c$, or $d$. Consider an arbitrary negative example $n_i = p(s_1, \ldots, s_n)$. Because $p(t_1, \ldots, t_n) \not\geq_{\Sigma Q} p(s_1, \ldots, s_n)$, and $p(t_1, \ldots, t_n)$ has no repeated variables, $t_j \not\geq_{\Sigma Q} s_j$ for some $1 \leq j \leq n$. Then either (1) $t_j$ is a variable of sort $\tau$ and $s_j$ is $d$, or (2) $t_j$ is a variable of sort $\omega$ and $s_j$ is $c$. In case (1), $V(p_j) = 1$ and $p_j$ appears unnegated in $b_i$. In case (2), $V(p_j) = 0$ and $p_j$ appears negated in $b_i$. In either
case, \( V \) satisfies \( b_i \). Since \( n_i \) is an arbitrary negative example, \( V \) satisfies each \( b_i \) and therefore satisfies \( \beta \).

\( \beta \) Satisfiable \( \Rightarrow \) Consistent Sorted Atom:

Suppose some truth assignment \( V \) satisfies \( \beta \). Let \( p(t_1, \ldots, t_n) \) be a sorted atom such that, for all \( 1 \leq j \leq n \), if \( V(p_j) = 0 \) then \( t_j \) is \( y_j;\omega \), and if \( V(p_j) = 1 \) then \( t_j \) is \( y_j;\tau \), where the \( y_j \), \( 1 \leq j \leq n \), are distinct. Then \( p(t_1, \ldots, t_n) \) is \( \Sigma_Q \)-more general than the positive examples. Consider an arbitrary negative example \( n_i = p(s_1, \ldots, s_n) \). Because \( V \) satisfies \( \beta \), \( V \) satisfies \( b_i \). Therefore, for some \( 1 \leq j \leq n \), either (1) \( p_j \) appears negated in \( b_i \) and \( V(p_j) = 0 \), or (2) \( p_j \) appears unnegated in \( b_i \) and \( V(p_j) = 1 \). In case (1), \( s_j \) is \( c \) and \( t_j \) is \( y_j;\omega \); then \( t_j \not\in \Sigma \), so \( p(t_1, \ldots, t_n) \not\in \Sigma \). In case (2), \( s_j \) is \( d \) and \( t_j \) is \( y_j;\tau \); then again, \( t_j \not\in \Sigma \), so \( p(t_1, \ldots, t_n) \not\in \Sigma \). Because \( n_i = p(s_1, \ldots, s_n) \) was an arbitrary negative example, \( p(t_1, \ldots, t_n) \) is not \( \Sigma \)-more general than any negative example, and therefore \( p(t_1, \ldots, t_n) \) is consistent with the examples relative to \( \Sigma \).

Therefore, assuming \( R \neq NP \), no algorithm exists that pac-learns sorted atoms or polynomially learns sorted atoms by equivalence queries.

The prominent feature of the sort theory \( \Sigma_Q \) used in the proof of Theorem 110 is that the constants \( a \) and \( b \) denote individuals belonging to two minimal sorts, \( \tau \) and \( \omega \), each of which contains another named individual (\( c \) in \( \tau \), \( d \) in \( \omega \)) that the other does not. We refer to \( \tau \) and \( \omega \) as multiple minimal sorts. But in fact the result does not require multiple minimal sorts. We can generate the same proof with the following sort theory, which has no multiple minimal sorts.

\[
\Sigma_R = \{ \tau(f(a)), \tau(f(b)), \tau(d), \forall x \text{ UNIV}(x) \}
\]

Notice that any taxonomic query regarding \( \Sigma_R \) can be answered in constant time. In the revised proof, the term \( f(a) \) plays the role of \( a \), \( f(b) \) plays the role of \( b \), \( f(c) \) plays the role of \( c \), and the terms \( f(x_1), f(x_2), \ldots \) play the roles of distinct variables of sort \( \omega \). The constant \( d \) and the distinct variables of sort \( \tau \) are used in the same way as in the original
proof. The key observation is that $f(x)$ and $x: \tau$ are both $\Sigma_R$-more general than $f(a)$ and $f(b)$, while only $f(x)$ is $\Sigma_R$-more general than $f(c)$ and only $x: \tau$ is $\Sigma_R$-more general than $d$. This mirrors the situation in the original proof where $x: \tau$ and $y: \omega$ are $\Sigma_Q$-more general than $a$ and $b$, while only $x: \tau$ is $\Sigma_Q$-more general than $c$, and only $y: \omega$ is $\Sigma_Q$-more general than $d$. Notice that the function $f$ is polymorphic according to $\Sigma_R$, because $f(a)$ and $f(b)$ belong to the sort $\tau$ ($\Sigma_R \models \tau(f(a))$ and $\Sigma_R \models \tau(f(b))$), but $\Sigma_R \nvdash \forall x (\tau(f(x)))$. It is this polymorphism that makes the revised proof possible. Thus function polymorphism can make the problem hard in the same way that multiple minimal sorts can.

It is also worth noting that the hardness result of Theorem 110 is in some sense a stronger kind of result than Theorem 86, because both $\Sigma_Q$ and $\Sigma_R$ are such that taxonomic queries can be answered in constant time, (though making the query still requires linear time), rather than polynomial time, and either theory is sufficient for the encoding of any instance of 3SAT. These theories can be used for the reduction because no part of a particular instance of 3SAT is actually encoded in the theory, but only in the examples. In the proof of Theorem 86, the theory itself changes from one instance of 3SAT to another. Theorem 110 is a stronger kind of result also in the sense that either function polymorphism or multiple minimal sorts in the sort theory can make the problem hard, whereas the reduction for the earlier result used (and seemed to require) both function polymorphism and multiple minimal sorts.

### 8.1.2 The Learnability of Simple Constrained Atoms

The class of universally-closed simple constrained atoms is a larger class of definite clauses than the class of universally-closed sorted atoms. It is the class of non-recursive definite clauses such that every term in the body appears in the head. In this section we study the learnability of universally-closed simple constrained atoms; in addition, we restrict the arity of the constraint predicates used, but whereas the arity bound for sorted atoms is 1, any bound may be chosen for simple constrained atoms. Specifically, in the learning problem we consider, $\mathcal{C}$ and $\mathcal{D}$ are each the class of universally-closed simple constrained atoms with some fixed bound $r$ on the arity of constraint predicates, and $\mathcal{T}$ is the class of
equality-free constraint theories (1) that use a finite set of constraint predicates, each of
arity at most \( r \), and (2) for which constraint queries are answered efficiently. Again, for
convenience we sometimes refer to universally-closed simple constrained atoms as simple
constrained atoms, with the universal quantification implicit. Notice that the class of
simple constrained atoms with \( r = 1 \) is the class of extended sorted atoms.

We show that the Simple Constrained Anti-Unification Algorithm can be used for
polynomial learning by equivalence queries and for pac-learning. We are justified in
using simple constrained anti-unification in learning since one simple constrained atom
covers a second one with respect to a constraint theory \( \Sigma \) if and only if the first is \( \Sigma \)-more
general than the second.

**Theorem 111** Let \( \mathcal{C} \) and \( \mathcal{D} \) each be the class of simple constrained atoms (with \( \bot \) in-
cluded), with constraint predicates of arity at most \( r \). Let \( \mathcal{T} \) be the class of equality-free
constraint theories (1) that use a finite set of constraint predicates, each of arity at most
\( r \), and (2) for which constraint queries are answered efficiently. Then \( \mathcal{C} \) is polynomially
learnable by equivalence queries, relative to \( \mathcal{D} \) and \( \mathcal{T} \).

**Proof:** The following algorithm polynomially learns \( \mathcal{C} \) by equivalence queries with re-
spect to \( \mathcal{D} \) and any constraint theory \( \Sigma \) in \( \mathcal{T} \). The algorithm first conjectures \( \bot \). If it is
correct, the algorithm is finished. If it is incorrect, the only counterexamples are simple
constrained atoms in \( \mathcal{D} \) that are labeled as positive examples. One positive example,
\( \phi/C \), is received in response to the prediction. From this point on, the algorithm con-
jectures the \( LGG_{\geq \Sigma} \) of the positive examples received thus far, as computed by the Simple
Constrained Anti-Unification Algorithm. Based on Theorem 14 the \( LGG_{\geq \Sigma} \) of such a
set of examples is the logically weakest simple constrained atom with respect to \( \Sigma \) that,
together with \( \Sigma \), entails the positive examples seen thus far. Therefore the conjecture
is always logically weaker, relative to \( \Sigma \), than the target. It follows that the algorithm
receives only positive examples as counterexamples.

If the arity of constraint predicates is bounded by some \( r \), then for any given \( \Sigma \) in
\( \mathcal{T} \) the Constrained Anti-Unification Algorithm runs in time polynomial in the sizes of
the simple constrained atoms it receives as input, and it produces a simple constrained atom that uses constraint predicates of arity at most \( r \). Therefore, every conjecture can be computed in time polynomial in the sizes of the counterexamples seen thus far. We complete the proof by showing that the number of counterexamples the algorithm receives is bounded by a polynomial in the size of the first counterexample, \( \phi / C \). Theorem 67 (Chapter 4) enables us to do so.

Because each constrained atom that the algorithm conjectures is strictly \( \Sigma \)-more general than the previous conjectures, the sequence of such conjectures is an ascending chain ordered by \( \geq \Sigma \). Because each conjecture except the first, \( \bot \), is strictly \( \Sigma \)-more general than the first counterexample, \( \phi / C \), we can replace the first conjecture with \( \phi / C \) and still have an ascending chain of the same size. In addition, again because each conjecture except \( \phi / C \) is strictly \( \Sigma \)-more general than \( \phi / C \), \( \phi \) is an instance of the head of each conjecture. Therefore the head size of each conjectured constrained atom in this chain is at most \( n \), where \( n \) is the size of \( \phi \). Thus by Theorem 67 this ascending chain of conjectured simple constrained atoms ordered by \( \geq \Sigma \) has size at most \( Kn^r + Ln^r + n + 1 \), where \( K \) is the number of constraint predicates in \( \Sigma \), \( L \) is the number of constraint predicates in \( C \) but not \( \Sigma \), and \( r \) is the arity-bound on constraint predicates. Hence the algorithm conjectures at most \( Kn^r + Ln^r + n + 1 \) simple constrained atoms, the last of which must be correct. Since \( r \) is fixed, and \( K \) is fixed for a given \( \Sigma \), this number of conjectures is polynomial in the size of the first counterexample, \( \phi / C \). \( \square \)

As noted earlier, polynomial learning by equivalence queries implies polynomial pac-learning. Another way to prove the pac-learnability of a class \( \mathcal{C} \), such as the class of simple constrained atoms with bounded predicate arity, is to (1) show that the VC-dimension of \( \mathcal{C}_n \), where \( \mathcal{C}_n \) is the subclass of \( \mathcal{C} \) consisting of concepts of size at most \( n \), is polynomial in \( n \), and (2) provide a polynomial-time algorithm that solves the consistency problem for \( \mathcal{C} \). We have already seen (Chapter 6) that simple constrained atoms with constraint predicates of arity at most \( r \) meet condition (1), relative to any given constraint theory \( \Sigma \) with a finite set of constraint predicates, each of arity at most \( r \). As for condition (2), the Simple Constrained Anti-Unification Algorithm can be used to solve consistency in
polynomial time relative to any constraint theory \( \Sigma \), where the arities of all constraint predicates is at most \( r \) and constraint queries regarding \( \Sigma \) are answered efficiently. To use the Simple Constrained Anti-Unification Algorithm to solve consistency, use the algorithm to compute the \( LGG_{\geq \Sigma} \) of the positive examples. If any constrained atom is consistent with the set of examples, the \( LGG_{\geq \Sigma} \) of these positive examples is consistent. Then check whether the \( LGG_{\geq \Sigma} \) covers any negative examples. This check can be done in polynomial time using the constraint queries used by the Simple Constrained Anti-Unification Algorithm. If the \( LGG_{\geq \Sigma} \) covers no negative examples, then it is consistent with the set of examples. Otherwise, no simple constrained atom is consistent with the set of examples; that is, the target sentence is not a simple constrained atom.

In general, the learnability of a class of concepts or sentences, such as simple constrained atoms, does not imply the learnability of a subclass, such as ordinary atoms. In this case, however, the result does imply the learnability of ordinary atoms. This is because the learnability problem for ordinary atoms may be viewed as a case of the learnability problem for simple constrained atoms where \( r \) is 0 and \( \Sigma \) is empty.

### 8.1.3 The Learnability of Conjunctions of Simple Constrained Atoms

An obvious question to ask next is whether we can learn conjunctions of universally-closed simple constrained atoms. This class is the class of logic programs whose clauses are non-recursive definite clauses such that every term in the body of a clause appears in the head. For reasons that will become clear later, we restrict constraint theories to be equality-free, to be in Skolem Normal Form, and to have an initial model. This restriction is consistent with inductive logic programming, since background theories typically are logic programs (or more restricted theories), which always are equality-free, are in Skolem Normal Form, and have initial models. Specifically, in this learning problem \( \mathcal{C} \) is the class of conjunctions of universally-closed simple constrained atoms, \( \mathcal{D} \) is the class of universally-closed simple constrained atoms, and \( \mathcal{T} \) is the class of equality-
free constraint theories $\Sigma$ such that $\Sigma$ uses a finite set of constraint predicates, $\Sigma$ is in Skolem Normal Form, $\Sigma$ has an initial model, and constraint queries regarding $\Sigma$ can be answered efficiently. In addition, we again place a bound of $r$ on the arities of all constraint predicates.

The following theorem states that the class of conjunctions of universally closed atoms, or logic programs consisting only of facts (atoms), is not polynomially learnable by equivalence queries. Because the class of conjunctions of universally-closed ordinary atoms is the same as the class of conjunctions of universally-closed simple constrained atoms, where $r$ is 0 and $\Sigma$ is empty, the result implies that the class of conjunctions of universally-closed simple constrained atoms is not polynomially learnable by equivalence queries. The proof shows that the class of conjunctions of universally-closed atoms is polynomially learnable by equivalence queries only if the class of propositional DNF formulas is so learnable. Angluin has shown that the class of propositional DNF formulas is not polynomially learnable by equivalence queries [4].

**Theorem 112** The class of conjunctions of universally-closed atoms is not polynomially learnable by equivalence queries.

*Proof:* The proof show that if the class were polynomially learnable by equivalence queries then so would be the class of propositional DNF formulas. We begin by providing a mapping from truth assignments over $n$ propositional variables $p_1, ..., p_n$ to universally-closed atoms of size $2n + 2$, as well as mappings between DNF formulas over $p_1, ..., p_n$ and conjunctions of universally-closed atoms of size $2n_2$.

Consider an arbitrary truth assignment $V$ over the propositional variables $p_1, ..., p_n$.

Let $\text{Atom}(V)$ map $V$ to the following atom $\alpha$. Initially, let $\alpha$ be $p(x_1, ..., x_{2n+2})$, where $x_1, ..., x_{2n+2}$ are distinct variables. For all $1 \leq j \leq n$, if $V(p_j) = 0$ then change $x_j$ to $x_{2n+1}$, and if $V(p_j) = 1$ then change $x_{n+j}$ to $x_{2n+2}$. Intuitively, repeated variables in an atom encode the corresponding truth assignment. For example, if $V$ is 11001 then $\text{Atom}(V)$

---

3These mappings are modified from mappings that Mike Frazier developed in our work on the learnability of simple recursive definite clause theories [21].
is $p(x_1, x_2, x_{11}, x_{11}, x_5, x_7, x_9, x_{12}, x_{11}, x_{11}, x_{12})$. Occurrences of $x_{11}$ in the first five arguments identify the propositional variables that $V$ maps to 0, and occurrences of $x_{12}$ in the second five arguments identify the propositional variables that $V$ maps to 1. Notice that this mapping is one-to-one, modulo the renaming of variables in an atom. Notice also that the mapping can be computed in polynomial time, and therefore the resulting atom is only polynomially larger than the original truth assignment.

We now define a mapping $\text{Conj-DNF}$ from conjunctions of universally-closed atoms of size $2n + 2$ to DNF formulas over $p_1, \ldots, p_n$. Let $\bigwedge \alpha_1 \land \ldots \land \bigwedge \alpha_k$ be any such conjunction. For each $\alpha_i$, $1 \leq i \leq k$, we define a DNF term (disjunct) $B_i$ as follows. If $\alpha_i$ is built from a predicate other than $p$ (of arity $2n + 2$), then $B_i$ is empty. Otherwise, let $\alpha_i$ be $p(t_1, \ldots, t_{2n+2})$. $B_i$ is empty if any $t_j$, $1 \leq j \leq 2n + 2$, is not a variable, or if $t_j$ and $t_m$ are variables such that $t_j = t_m$ where: (1) $1 \leq j \leq n$ and $n + 1 \leq m \leq 2n$, (2) $1 \leq j \leq n$ and $m = 2n + 2$, (3) $n + 1 \leq j \leq 2n$ and $m = 2n + 1$, or (4) $j = 2n + 1$ and $m = 2n + 2$. (If any of cases (1) through (4) is true, we say that the variables in $\alpha_i$ repeat inappropriately; otherwise we say that the variables repeat appropriately.) Otherwise, if $t_j = t_m$ for $1 \leq j \neq m \leq n$ then let $B_i$ contain the literals $\overline{p_j}$ and $\overline{p_m}$. If $t_{n+j} = t_{n+m}$ for $1 \leq j \neq m \leq n$ then let $B_i$ contain the literals $p_j$ and $p_m$. If $t_j = t_m$ where $1 \leq j \leq n$ and $m = 2n + 1$ then let $B_i$ contain $\overline{p_j}$. If $t_{n+j} = t_m$ where $1 \leq j \leq n$ and $m = 2n + 2$ then let $B_i$ contain $p_j$. Let $B_i$ contain no other literals. Then the resulting DNF formula is $B_1 \lor \ldots \lor B_k$; if every $B_i$, $1 \leq i \leq k$, is empty, then the formula is the empty DNF formula, which classifies every example as negative. As an example,

$$
\text{Conj-DNF}(\overline{\bigwedge}(p(x_1, x_7, x_3, x_8, x_5, x_6, x_7, x_8)) \land \overline{\bigwedge}(p(x_1, x_2, x_7, x_4, x_8, x_6, x_7, x_8)) \\
\land \overline{\bigwedge}(p(x_1, x_2, x_4, x_5, x_6, x_7, x_8)) \land \overline{\bigwedge}(p(x_1, f(x), x_3, x_4, x_5, a, x_7, x_1))
$$

is $(p_1 \land \overline{p_2}) \lor (p_2 \land \overline{p_3}) \lor (\overline{p_1} \land \overline{p_3})$. Notice that this mapping can be computed in polynomial time, and the size of the resulting DNF is actually less than the size of the input. Notice also that it is a many-to-one mapping onto the DNF formulas over $p_1, \ldots, p_n$. Therefore, we can define a mapping DNF-Conj from DNF formulas over $p_1, \ldots, p_n$ to conjunctions of universally-closed atoms of size $2n + 2$, such that if DNF-Conj maps a DNF formula

157
\[ B_1 \lor \ldots \lor B_k \] to a conjunction \( \overline{\wedge} \alpha_1 \land \ldots \land \overline{\wedge} \alpha_k \) then \( \text{Conj-DNF} \) maps \( \overline{\wedge} \alpha_1 \land \ldots \land \overline{\wedge} \alpha_k \) to \( B_1 \lor \ldots \lor B_k \). Let \( B_1 \lor \ldots \lor B_k \) be any DNF formula over \( p_1, \ldots, p_n \). For each DNF term (disjunct) \( B_i \), we define a corresponding atom \( \alpha_i = p(x_1, \ldots, x_{2n+2}) \) as follows. Initially, let \( x_1, \ldots, x_{2n+2} \) be distinct variables. For all \( 1 \leq j \leq n \), if \( B_i \) contains \( p_j \) then change \( x_j \) to \( x_{2n+1} \), and if \( B_i \) contains \( \overline{p_j} \) then change \( x_{n+j} \) to \( x_{2n+2} \). Finally, notice that \( \text{DNF-conj} \) can be computed in polynomial time.

We now show that a conjunction of universally-closed atoms \( C \) entails a universally-closed atom \( \alpha_i \) in the range of the function \( \text{Atom} \), if and only if the truth assignment \( V \) that corresponds to \( \alpha \) (\( \text{Atom}(V) = \alpha \)) satisfies \( \text{Conj-DNF}(C) \).

If: We prove the contrapositive. Let \( \alpha \) be \( p(s_1, \ldots, s_{2n+2}) \). Suppose \( C \) does not entail \( \alpha \). Then, by strong compactness, no atom in \( C \) is more general than \( \alpha \). Consider an arbitrary atom \( \alpha_i \) in \( C \). We show that the corresponding DNF term \( B_i \) in \( \text{Conj-DNF}(C) \) is not satisfied by \( V \). By definition, \( V \) does not satisfy \( B_i \) if \( B_i \) is empty. If \( B_i \) is nonempty, then \( \alpha_i \) has the form \( p(t_1, \ldots, t_{2n+2}) \), where \( t_1, \ldots, t_{2n+2} \) are variables that repeat appropriately. Because \( \alpha_i \not\subset \alpha \), there must exist \( 1 \leq j < m \leq 2n+2 \) such that \( t_j = t_m \) yet \( s_j \neq s_m \). To have \( t_j = t_m \), one of the following must hold: (1) both \( j \) and \( m \) are in \( 1, \ldots, n \), (2) \( j \) is in \( 1, \ldots, n \) and \( m = 2n+1 \), (3) both \( j \) and \( m \) are in \( n+1, \ldots, 2n \), or (4) \( j \) is in \( n+1, \ldots, 2n \), and \( m = 2n+2 \). In case (1), \( B_i \) contains both \( \overline{p_j} \) and \( \overline{p_m} \); in case (2), \( B_i \) contains \( \overline{p_j} \); in case (3), \( B_i \) contains both \( p_{j-n} \) and \( p_{m-n} \); in case (4), \( B_i \) contains \( p_{j-n} \). But from the definition of the function \( \text{Atom} \), because \( s_j \neq s_m \): in case (1), \( V \) maps \( p_j \) and \( p_m \) to different truth values; in case (2), \( V \) maps \( p_j \) to 1; in case (3), \( V \) maps \( p_{j-n} \) and \( p_{m-n} \) to different truth values; in case (4), \( V \) maps \( p_j \) to 0. In every case, \( V \) fails to satisfy \( B_i \). Because \( \alpha_i \) was an arbitrary atom in \( C \), \( B_i \) is an arbitrary disjunct in \( \text{Conj-DNF}(C) \). Therefore, \( V \) does not satisfy any disjunct of the DNF formula, so it does not satisfy the formula.

Only-If: Again we prove the contrapositive. Assume that \( V \) does not satisfy \( \text{Conj-DNF}(C) \). We show that \( C \) does not entail \( \alpha \), where \( \text{Atom}(V) = \alpha \). By strong compactness, it suffices to show that no atom \( \alpha_i \) in \( C \) is more general than \( \alpha \). Let \( \alpha_i \) be an arbitrary atom in \( C \), and let \( B_i \) be the corresponding disjunct in \( \text{Conj-DNF}(C) \). Let \( \alpha \)
be \( p(s_1, ..., s_{2n+2}) \). We show that \( \alpha_i \) is not more general than \( \alpha \). If \( B_i \) is empty, then one of the following must be true: \( \alpha_i \) is built from some predicate other than \( p \), \( \alpha_i \) contains a non-variable term, or the variables of \( \alpha_i \) repeat inappropriately. In any of these cases, it follows from the definition of the function \( \text{Atom} \) that \( \alpha_i \) is not more general than \( \alpha \). Therefore, we may assume that \( B_i \) is nonempty and \( \alpha_i \) is of the form \( p(t_1, ..., t_{2n+2}) \), where \( t_1, ..., t_{2n+2} \) are variables that repeat appropriately. Because \( V \) does not satisfy \( \text{Conj-DNF}(C) \), \( V \) does not satisfy \( B_i \). Therefore, either (1) \( B_i \) contains a literal \( p_j \), and \( V \) maps \( p_j \) to 0, or (2) \( B_i \) contains a literal \( \overline{p}_j \), and \( V \) maps \( p_j \) to 1. In case (1), \( t_{j+n} = t_m \), where: \( m \neq j + n \), and either \( n + 1 \leq m \leq 2n \) or \( m = 2n + 2 \). But because \( V \) maps \( p_j \) to 0, rather than to 1, \( s_{j+n} \) is a variable that appears nowhere else in \( \alpha \). Therefore \( \alpha_i \not\subset \alpha \). In case (2), \( t_j = t_m \), where: \( j \neq m \), and either \( 1 \leq m \leq n \) or \( m = 2n + 1 \). But because \( V \) maps \( p_j \) to 1, rather than to 0, \( s_j \) is a variable that appears nowhere else in \( \alpha \). Therefore \( \alpha_i \not\subset \alpha \).

We now show that a learning algorithm \( A \) for conjunctions of universally-closed atoms can be used to build a learning algorithm \( A' \) for DNF. \( A' \) will act as the oracle for equivalence queries posed by \( A \). \( A' \) will tell \( A \) that it is correct only when \( A \) conjectures any conjunction \( C \) such that \( \text{Conj-DNF}(C) \) is equivalent to the target DNF formula; at least one such conjunction exists—the result of applying \( \text{DNF-Conj} \) to the target. (For DNF, as for other classes of propositional formulas, coverage-equivalence and logical equivalence are the same, so we need not specify which is meant here.) When \( A \) conjectures a conjunction of universally-closed atoms \( C \), \( A' \) poses an equivalence query \( D = \text{Conj-DNF}(C) \) to the oracle for equivalence queries for DNF. If the conjecture is correct, then \( A' \) is finished. Otherwise, \( A' \) receives a truth assignment \( V \) as a counterexample. \( A' \) then returns \( \text{Atom}(V) \) to \( A \) as a counterexample; from our preceding arguments, \( D \) classifies \( \text{Atom}(V) \) differently from every conjunction \( C \) for which \( \text{Conj-DNF}(C) \) is equivalent to the target DNF. Thus \( \text{Atom}(V) \) is appropriate as a counterexample to \( D \). \( A' \) continues to interact with \( A \) in this manner. Because the output of \( \text{Conj-DNF} \) or \( \text{Atom} \) is only polynomially-larger than the input, and these functions can be computed in polynomial time, if at any point in the run \( A \) has used only time polynomial in the size of its target
and the size of the largest counterexample seen thus far, then the same is true of $A'$. The result follows. \hfill \Box

This theorem says nothing about pac-learnability. But the mappings used in the proof of the theorem can be used in a straightforward manner to show that the class of conjunctions of (universally-closed) ordinary atoms is pac-learnable only if the class of propositional DNF formulas is pac-learnable. The proof of the following theorem does so.

**Theorem 113** If the class of conjunctions of universally-closed ordinary atoms is pac-learnable then the class of propositional DNF formulas is pac-learnable.

*Proof:* From the notion of prediction-preserving reductions, developed by Pitt and Warmuth [55], the theorem follows if we produce (1) a function $f_i$ (the *instance mapping*) from truth assignments over the propositional variables $p_1, \ldots, p_n$ to universally-closed atoms, and (2) an invertible function $f_c$ (the *concept mapping*) from DNF formulas over $p_1, \ldots, p_n$ to conjunctions of universally-closed atoms, such that the following are true.

- A truth assignment $V$ satisfies a DNF formula $\phi$ if and only if the universally-closed atom $f_i(V)$ is entailed by the conjunction $f_c(\phi)$.
- The size of $f_c(\phi)$ is bounded by a polynomial in the size of $\phi$.
- The time to compute $f_i(V)$ is bounded by a polynomial in the size of $V$.
- The time to compute the inverse $f_c^{-1}(C)$ of $f_c(C)$, where $C$ is a conjunction of universally-closed atoms, is bounded by a polynomial in the size of $C$.

If these properties hold, then any pac-learning algorithm for conjunctions of universally-closed atoms can be used to pac-learn DNF. It is straightforward to verify that choosing the function $\text{Atom}$ for $f_i$, $\text{DNF-conj}$ for $f_c$, and $\text{conj-DNF}$ (restricted to the range of $\text{DNF-conj}$) for $f_c^{-1}$ fulfills these requirements. \hfill \Box

Because of the difficulty in finding a pac-learning algorithm for DNF, this result leads us to conclude that it will be difficult to obtain a pac-learning algorithm for conjunctions
of universally-closed atoms. Furthermore, because a pac-learning algorithm for conjunctions of universally-closed simple constrained atoms can be used to learn conjunctions of universally-closed atoms, we conclude that it will be difficult to obtain a pac-learning algorithm for conjunctions of universally-closed simple constrained atoms.

One technique we might consider to allow us to learn this class is to use subset queries, which we now define in the special context of simple constrained atoms. A subset query asks whether the target, a conjunction of universally-closed simple constrained atoms, $\Sigma$-entails a given universally-closed simple constrained atom $\nabla(\phi/C)$ (with constraint predicates of arity at most $r$). It is called a subset query because it is equivalent to asking whether the set of universally-closed simple constrained atoms that are $\Sigma$-entailed by $\nabla(\phi/C)$ (that $\nabla(\phi/C)$ would classify as positive) is a subset of the universally-closed simple constrained atoms that are $\Sigma$-entailed by the target. Because $\Sigma$ is in Skolem Normal Form and has an initial model, it follows from the strong compactness property of simple constrained atoms (Theorem 25, Chapter 2) that $\nabla(\phi/C)$ is $\Sigma$-entailed by the target if and only if for some conjunct $\nabla(\phi'/C')$ in the target, we have $\phi'/C' \geq_\Sigma \phi/C$. We will see that this result leads to a positive learnability result for conjunctions of universally-closed simple constrained atoms, with the additional help of subset queries. This is surprising, because it is not known whether DNF is pac-learnable with the addition of subset queries, and it is known that the “obvious” algorithm for DNF using subset queries does not succeed in pac-learning DNF [1]. We will see that the positive result does not imply the pac-learnability of DNF with subset queries; rather, due to a subtle distinction between first-order and propositional logic, while the strong compactness property we have described applies to universally-closed conjunctions of simple constrained atoms, DNF has no analogous property.

---

4 Technically, this also could be called a membership query, since any universally-closed simple constrained atom, with constraint predicates of arity at most $r$, is in $\mathcal{D}$. Because of the power of the query it seems more appropriate to call it a subset query. Later, we will note that all of the positive learnability results of this chapter also hold if only ground atoms are used as examples. When only ground atoms may serve as examples, our subset queries cannot be considered to be membership queries since they are not necessarily ground.
Theorem 114 Let $C$ be the class of conjunctions of universally-closed simple constrained atoms whose constraints use predicates of arity at most $r$. Let $D$ be the class of universally-closed simple constrained atoms whose constraints use predicates of arity at most $r$. Let $T$ be the class of constraint theories $\Sigma$ such that $\Sigma$ uses a finite set of constraint predicates, each of arity at most $r$, $\Sigma$ is in Skolem Normal Form, $\Sigma$ has an initial model, and constraint queries regarding $\Sigma$ can be answered efficiently. Then $C$ is polynomially learnable by equivalence queries and subset queries, relative to $D$ and $T$.

Proof: The following single algorithm polynomially learns $C$ by equivalence queries and subset queries with respect to $D$ and any constraint theory $\Sigma$ in $T$. The algorithm first predicts $\bot$. If it is correct, the algorithm is finished. If it is incorrect, the only counterexamples are simple constrained atoms in $D$ that are labeled as positive examples. One positive example, $\overline{\forall}(\phi/C)$, is received in response to the prediction. From this point on, the algorithm maintains a set of simple constrained atoms $S = \{\phi_1/C_1, ..., \phi_m/C_m\}$; initially, $m = 1$ and $\phi_1/C_1$ is $\phi/C$. At any point in the run, the algorithm's next conjecture is the concept $\overline{\forall}(\phi_1/C_1) \land ... \land \overline{\forall}(\phi_m/C_m)$. If this conjecture is correct, the algorithm is finished. If it is incorrect, it receives a new counterexample $\overline{\forall}(\phi'/C')$; later, we will show that this must be a positive example. Then for all $1 \leq i \leq m$ the algorithm uses the Simple Constrained Anti-Unification Algorithm to compute the $LGG_{\geq n}$ of $\phi_i/C_i$ and $\phi'/C'$. It issues a subset query about each $\phi_i/C_i$ (more precisely, $\overline{\forall}(\phi_i/C_i)$); if the answer to the subset query regarding $\phi_i/C_i$ is yes, the algorithm replaces $\phi_i/C_i$ in $S$ by $\phi_i/C_i$. If the answer to every subset query is no, the algorithm adds $\phi'/C'$ to $S$. The algorithm is then prepared to make its next conjecture.

We now show by induction that any conjecture by the algorithm is logically weaker than the target, with respect to $\Sigma$ (the target $\Sigma$-entails the conjecture); it follows that each counterexample is a positive example. Clearly the first conjecture, $\bot$, is weaker than the target. Also for the basis, because the second conjecture, $\overline{\forall}(\phi/C)$, is also a positive example, it is logically weaker than the target with respect to $\Sigma$. Now assume, by the inductive hypothesis, that the most recent conjecture, $\overline{\forall}(\phi_1/C_1) \land ... \land \overline{\forall}(\phi_m/C_m)$, is logically weaker than the target with respect to $\Sigma$. It follows from strong compactness
that, for all $1 \leq i \leq m$, the target contains a (universally-closed) simple constrained atom that is $\Sigma$-more general than $\phi_i/C_i$. In addition, because the conjecture is weaker than the target with respect to $\Sigma$, the counterexample $\overline{\vartheta}(\phi'/C')$ to the conjecture is a positive example. Again, from strong compactness, the target contains at least one simple constrained atom $\phi''/C''$ that is $\Sigma$-more general than $\phi'/C'$. If any such $\phi''/C''$ is also $\Sigma$-more general than some atom(s) $\phi_i/C_i$ in the most recent conjecture, then the $LGG_{\Sigma} \phi_i'/C_i'$ of $\phi_i/C_i$ and $\phi'/C'$ is, by definition of $LGG_{\Sigma}$, a $\Sigma$-instance of $\phi''/C''$. Therefore, replacing $\phi_i/C_i$ by $\phi_i'/C_i'$ will yield a new conjecture that is weaker than the target with respect to $\Sigma$. If no such $\phi''/C''$ is $\Sigma$-more general than any atom in the most recent conjecture, then $\phi'/C'$ is added to $S$, so $\overline{\vartheta}(\phi'/C')$ is added to the conjecture to obtain the new conjecture. Because $\phi'/C'$ is a $\Sigma$-instance of a simple constrained atom in the target, the new conjecture is weaker than the target with respect to $\Sigma$.

Next, observe that if a new simple constrained atom $\phi'/C'$ is added to $S$, then $\phi'/C'$ is a $\Sigma$-instance of some simple constrained atom in the target of which no other member of $S$ is a $\Sigma$-instance. It follows that $S$ contains at most the number of simple constrained atoms in the target, so any conjecture contains at most the number of simple constrained atoms in the target.

Now consider the time the algorithm requires between conjectures. Because the arity of constraint predicates is at most $r$, the Simple Constrained Anti-Unification Algorithm runs in time polynomial in the sizes of the simple constrained atoms it receives as input, and it produces a simple constrained atom that uses constraint predicates of arity at most $r$. To update $S$ requires at most $m$ calls to the Simple Constrained Anti-Unification Algorithm and $m$ subset queries, where $m$ is the number of conjuncts in the target. Therefore, the time required to update a conjecture is polynomial in the sizes of the counterexamples seen thus far and the size of the target.

Finally, we show that where $m$ is the number of simple constrained atoms in the target, $K$ is the number of constraint predicates in $\Sigma$, $n$ is the maximum of the head sizes of the counterexamples, and $L$ is the maximum number of constraint predicates in some counterexample but not in $\Sigma$, the number of conjectures that the algorithm makes
is $m(Kn^r + Ln^r + n + 1)$. This will complete the proof, since for any particular $\Sigma$ this value is polynomial in the size of the target and the size of the first counterexample. The maximum number of times that a new simple constrained atom can be added to $S$ is $m$. The maximum number of times that any of the (at most $m$) simple constrained atoms in $S$ can be replaced by a $\Sigma$-more general simple constrained atom is $m(Kn^r + Ln^r + n)$, since from Theorem 67 the maximum size of an ascending chain of simple constrained atoms as described is $Kn^r + Ln^r + n + 1$. 

It follows that conjunctions of universally-closed simple constrained atoms, where the arities of constraint predicates are bounded, are pac-learnable with the additional help of subset queries, with respect to constraint theories $\Sigma$ that are restricted as in the theorem.

A reasonable question is whether the same algorithm (modulo the $\text{Conj-DNF}$ and $\text{Atom}$ functions we described earlier) might work for DNF.\(^5\) This is an obvious choice for DNF, but it is known to fail. (It is based on the computation of minterms, and such approaches to learning DNF fail [1].) Consider the following example. Suppose 1100 $(V(p_1) = 1, V(p_2) = 1, V(p_3) = 0, V(p_4) = 0)$ and 0101 are the first two positive examples, and we wish to know whether our next conjecture should be $(p_1 \land p_2 \land \overline{p_3} \land \overline{p_4}) \lor (\overline{p_1} \land p_2 \land \overline{p_3} \land p_4)$ or $(p_2 \land \overline{p_3})$. (We wish to know whether the two positive examples should be “combined”.) The analogous algorithm for DNF would choose the latter conjecture if and only if the answer to the following subset query is yes: are all truth assignments that satisfy $(p_2 \land \overline{p_3})$ positive examples? But in fact all may be positive examples, yet the first conjecture should be chosen (the two positive examples should not be “combined”). For example, the target may be $(p_1) \lor (p_4)$; then the single term in the second conjecture cannot be “generalized” to either term in the target. The problem is that DNF formulas do not have a property analogous to the strong compactness property of conjunctions of (universally-closed) simple constrained atoms.

It should be noted that the above distinction is cause for care in using the learnability result for conjunctions of simple constrained atoms. The result holds if examples are

\(^5\)I thank Larry Watanabe for helpful discussions on this point.
classified according to a target conjunction, relative to \( \Sigma \). But if examples are classified in some different way, for example by a human teacher with additional knowledge, the algorithm could fail. For example, if a human teacher knows the actual model (that satisfies \( \Sigma \) and some supposed target conjunction), and the domain of that model is finite (for example, it contains only 0 and 1), then the teacher may answer subset queries in a way that is consistent with the model but not necessarily with the target and \( \Sigma \). In that case the answers to subset queries may fail to direct the choices that the learning algorithm needs to make in the same way that they fail for DNF.

Finally, it should be noted that while the obvious algorithm for learning DNF with subset queries does not work, the class of monotone DNF formulas (those in which all literals are unnegated propositional variables) is learnable with subset queries, or even more simply, with membership queries. Given the relationship between conjunctions of (universally-closed) atoms and DNF, the reader might question whether the learnability of conjunctions of atoms, or simple constrained atoms, is closely related to the learnability of monotone DNF. In fact, it seems that conjunctions of atoms and monotone DNF are different in a fundamental way, even though both are learnable by equivalence queries and subset queries. The learning algorithm for monotone DNF can be described at a high level as follows. The algorithm first conjectures the empty formula (which classifies all examples as negative). It receives a positive example and uses membership queries to determine a term in the DNF formula that is satisfied by the example. It then conjectures that term and receives a new positive example, which satisfies a different term of the formula. Again, the algorithm uses membership queries to determine a (new) term of the DNF formula that is satisfied by the new example, and it next conjectures the disjunction of the terms it has so far. The algorithm continues in this manner until it has every term of the monotone DNF formula. Suppose we try to develop an analogous learning algorithm for conjunctions of atoms. Specifically, when the algorithm obtains a positive example (an atom), it uses membership or subset queries to find an atom in the target that entails the example (that is, of which the example is an instance). The following adversary forces exponentially-many membership or subset queries (exponential
in the size of the positive example and "target"). For simplicity, let the atoms be built 
from a 4-ary predicate; the argument generalizes to atoms built from predicates of any 
arity.

Let the first positive example be \( p(a, a, a, a) \). This atom has an exponentially-large 
cover set (recall the discussion in Section 4.2.1) consisting of exactly the atoms that result 
from replacing some nonempty subset of the constants in the atom with a variable \( x \): 
\( p(x, a, a, a) \), \( p(a, x, a, a) \), \( p(a, a, x, a) \), \( p(a, a, a, x) \), \( p(x, x, a, a) \), \( p(x, a, x, a) \), etc. (Note, for 
example, that \( p(x, x, a, a) \) is not more general than, or does not entail, \( p(x, a, a, a) \), since 
the second has instances such as \( p(b, a, a, a) \) that are not instances of the first.) The 
only common instance of any two or more of the members of this cover set is \( p(a, a, a, a) \). 
Furthermore, each member of the cover set has in infinite number of instances that 
are not instances of any other member. For example, \( p(x, a, a, a) \) has instances such 
as \( p(b, a, a, a) \), \( p(c, a, a, a) \), \ldots, \( p(f(a), a, a, a) \), \( p(f(b), a, a, a) \), etc. On any membership 
or subset query (except \( p(a, a, a, a) \), which provides no new information) the adversary 
answers \textit{no}, until it has seen an instance of every member of the cover set (possibly 
that member itself) except one. When it sees an instance of the remaining member, 
the adversary answers \textit{yes} and considers that member to be the target. Notice that any 
atom that is not an instance of some member of the cover set (e.g., \( p(x, y, f(z), w) \)) is 
more general than, or entails, some atom not in this target, and so is not entailed by the 
target; therefore, \textit{no} answers to queries regarding these atoms are appropriate.

8.1.4 The Learnability of Conjunctions of Simple Constrained 
Atoms Relative to Partitioned Constraint Theories

In this section we further expand the positive learnability results we have obtained so 
far. One type of inductive learning problem that has been of great interest to machine 
learning researchers is the problem of learning in structural domains. In such a problem 
the examples are defined by a set of objects and a set of relations that hold among those 
objects. As noted in Chapter 7, one example of a structural domain is the Blocks World,
where the objects are blocks (and possibly a robot arm), and the relations include ontop and clear. In this section we consider the applicability of anti-unification to the problem of learning in structural domains. This section owes much to the work of Haussler [32]. It uses anti-unification to extend his results for structural domains in some simple but useful ways.

In first-order logic, terms denote objects, or individuals, when taken with a model and value assignment. In inductive logic programming, the terms in different examples—and therefore the objects those terms represent—are in effect matched by their locations in the heads of their respective definite clauses. This is the case with the learning algorithms we have examined in this chapter, based on anti-unification, as well. For example, suppose we have the following two positive examples of Blocks World states that contain a stack of blocks.

\[
\text{has-stack}(a, b, c) / \text{ontop}(a, b) \land \text{ontop}(b, c) \land \text{clear}(a)
\]

which may be written as

\[
\text{ontop}(a, b) \land \text{ontop}(b, c) \land \text{clear}(a) \rightarrow \text{has-stack}(a, b, c)
\]

This example says that the state consisting of blocks named \(a\), \(b\), and \(c\), where \(a\) is on top of \(b\), \(b\) is on top of \(c\), and \(a\) is clear, contains a stack.

\[
\text{has-stack}(d, e, f) / \text{ontop}(f, e) \land \text{clear}(d) \land \text{clear}(f)
\]

which may be written as

\[
\text{ontop}(f, e) \land \text{clear}(d) \land \text{clear}(f) \rightarrow \text{has-stack}(d, e, f)
\]

This example says that the state containing blocks named \(d\), \(e\), and \(f\), where \(f\) is on top of \(d\), and \(e\) and \(f\) are clear, contains a stack. The learning algorithm in Section 8.2 would compute the \(LGG_{2}\) of the examples, which is based on a matching of \(a\) with \(d\), \(b\) with
$e,$ and $c$ with $f$. This operation yields the following $\Sigma$-generalization (for now, take $\Sigma$ to be empty), where the variable $x$ corresponds to $a$ and $d$, $y$ corresponds to $b$ and $e$, and $z$ corresponds to $c$ and $f$.

$$\neg (\text{has-stack}(x, y, z)/\text{clear}(x))$$

which may be written as

$$\neg (\text{clear}(x) \rightarrow \text{has-stack}(x, y, z))$$

Obviously, such a $\Sigma$-generalization is not very useful in learning the concept $\text{has-stack}$. The problem is in the representation. If we re-order the terms in the head of one of the clauses, the result is more appropriate. For example, if the head of the second clause were instead $\text{has-stack}(f, e, d)$, then the resulting $\Sigma$-generalization would be the following.

$$\neg (\text{has-stack}(x, y, z)/\text{clear}(x) \land \text{ontop}(x, y))$$

which may be written as

$$\neg (\text{clear}(x) \land \text{ontop}(x, y) \rightarrow \text{has-stack}(x, y, z))$$

But while this particular order may be the best for this pair of examples, it may not be the best for another. In general, the problem is that the obvious representation for Blocks World states, or states in other structural domains, as universally-closed simple constrained atoms, or more generally as definite clauses, is flawed because it imposes an artificial order on the objects in the state. Does this mean that anti-unification and inductive logic programming have nothing to say about structural domains? This section shows otherwise, by using the $\text{set}$ function symbol and partitioned constraint theories with the equational part $\Sigma_5$, as introduced in Chapter 7. We repeat $\Sigma_5$ below.

$$\Sigma_5 = \{ \forall x \forall y \forall z \ (\text{set}(x, \text{set}(y, z)) = \text{set}(y, \text{set}(x, z))) \}$$
Suppose that for the preceding example we choose $\Sigma_5$ as the background theory. In addition, suppose we represent the domains of states using the set function, so that the heads of the two examples are the following.

$$\text{has-stack}(\text{set}(a, \text{set}(b, \text{set}(c, \text{nil}))))$$

$$\text{has-stack}(\text{set}(d, \text{set}(e, \text{set}(f, \text{nil}))))$$

If we do so, then the partitioned constrained $E$-anti-unification algorithm produces a $CIG_{\Sigma_5}$ consisting of several $\Sigma_5$-generalizations, including

$$\overline{V}(\text{has-stack} (\text{has-stack} (\text{set}(x, \text{set}(y, \text{set}(z, \text{nil})))))) / \text{clear}(x) \land \text{ontop}(x, y)$$

which also may be written as

$$\overline{V}(\text{clear}(x) \land \text{ontop}(x, y) \rightarrow \text{has-stack} (\text{has-stack} (\text{set}(x, \text{set}(y, \text{set}(z, \text{nil}))))))$$

This $\Sigma_5$-generalization may be read as saying that any state with a set of three objects, \{x, y, z\}, such that x is clear and is on top of y, contains a stack. This $\Sigma_5$-generalization is the least $\Sigma_5$-generalization, or $LGG_{\Sigma_5}$, for one particular matching of the members of the sets in the two examples. The $CIG_{\Sigma_5}$ consists of at most one $\Sigma_5$-generalization for each distinct matching of the members of the sets in the two examples. Some matchings may contribute no $\Sigma_5$-generalization to the $CIG_{\Sigma_5}$ because they may produce $LGG_{\Sigma_5}$s that are $\Sigma_5$-more general than the $LGG_{\Sigma_5}$s produced by other matchings.

The preceding discussion motivates consideration of the learning problem we consider next. The problem formulation uses the following definition: the constant nil is a set-term, any variable is a set-term, and any term of the form set(t₁, t₂) is a set-term if and only if t₂ is a set-term; no other terms are set-terms. In the learning problem we consider, $C$ is the class of conjunctions of universally-closed simple constrained atoms in which: at most one argument in the head contains the set function symbol, this argument is a set-term, and the set-function symbol does not appear in the constraint. We also call
such simple constrained atoms simple constrained set-atoms. \( \mathcal{D} \) is the class of universally-closed simple constrained set-atoms. Furthermore, as in the previous sections we require a bound \( r \) on the arity of all constraint predicates used. Finally, \( \mathcal{T} \) is the class of Skolem Normal Form partitioned constraint theories, \( \Sigma \), such that (1) \( \Sigma \) has an initial model, (2) the equational part of \( \Sigma \) is \( \Sigma_5 \), (3) the set function does not appear in the equality-free part of \( \Sigma \), (4) constraint queries regarding the equality-free part of \( \Sigma \) can be answered efficiently, (5) \( \Sigma \) uses only a finite set of constraint predicates, and (6) the arity of any constraint predicate in \( \Sigma \) is at most \( r \).

It is worth noting that where \( \Sigma \) is any theory in the class \( \mathcal{T} \) as defined, any simple constrained atom that is \( \Sigma \)-more than a simple constrained set-atom is itself a simple constrained set-atom. Furthermore, the size of the set-term in a simple constrained set-atom is no greater than the size of the largest set-term in any of its \( \Sigma \)-instances. It is also worth noting that, because \( \Sigma_5 \) is a definite clause theory (with equality), any partitioned constraint theory is in Skolem Normal Form and has an initial model if its equational part is \( \Sigma_5 \) and its equality-free part is a definite clause theory without equality.

Notice how this learning problem is applicable to structural domains. We may use a set-term to denote the set of individuals in the domain, and we may use the constraint to specify the relations among those individuals. The equality-free part of the constraint theory \( \Sigma \) provides background information about the relations. For examples, the reader may refer to Example 98 and Example 99 in Chapter 7.

In our intended model, a set-term such as \( \text{set}(a, \text{set}(b, \text{set}(c, \text{nil}))) \) represents the set whose elements are the individuals denoted by \( a \), \( b \), and \( c \). Therefore, for shorthand we also write this term as \( \{a, b, c\} \). Similarly, a set-term ending in a variable, such as \( \text{set}(a, \text{set}(b, x)) \), represents a set containing the elements denoted by \( a \) and \( b \), and possibly others. For shorthand, we also represent such a term as \( \{a, b, X\} \), where the capital \( X \) denotes some number of additional items.

We now consider one additional example, also from the Blocks World. The first simple constrained atom in this example (below) says that, in the following state of the Blocks World, the operation \( \text{unstack}(a, b) \), in which a robot arm lifts block \( a \) from block \( b \), is an
optimal action toward achieving the goal \(\text{ontop}(b, a) \land \text{clear}(b)\). (The goal asks for block \(a\) to be on top of \(b\), and for \(a\) to be free from having any block on top.)

\[
\text{ontop}(a, b) \land \text{ontop}(b, c) \land \text{ontable}(c) \land \text{clear}(a) \land \text{empty}(\text{arm})
\]

This may be represented as follows.

\[
\text{apply}(\text{unstack}(a, b), [\text{ontop}(b, a), \text{clear}(b)], \text{domain}(\text{arm}, \{a, b, c\}))/\text{ONTOP}(a, b) \land \text{ONTOP}(b, c) \land \\
\text{ONTABLE}(c) \land \text{CLEAR}(a) \land \text{EMPTY}(\text{arm})
\]

The first argument in the head specifies the optimal operator application, the second specifies the goal as a list of conjuncts, and the third specifies that the state contains a robot arm and the set of blocks \(\{a, b, c\}\). The constraint specifies the relations (ground atoms) that hold among the objects in the state.

The second simple constrained atom (below) says that, in the following state of the Blocks World, the operation \(\text{unstack}(e, d)\), in which a robot arm lifts block \(e\) from block \(d\), is an optimal action toward achieving the goal \(\text{ontop}(d, e) \land \text{clear}(d)\). (The goal asks for block \(d\) to be on top of \(e\), and for \(d\) to be free from having any block on top.)

\[
\text{ontop}(e, d) \land \text{clear}(e) \land \text{ontable}(d) \land \text{clear}(f) \land \text{ontable}(f) \land \text{clear}(g) \land \\
\text{ontable}(g) \land \text{empty}(\text{arm})
\]

This may be represented as follows.

\[
\text{apply}(\text{unstack}(e, d), [\text{ontop}(d, e), \text{clear}(d)], \text{domain}(\text{arm}, \{d, e, f, g\}))/\text{ONTOP}(e, d) \land \text{CLEAR}(e) \land \\
\text{ONTABLE}(d) \land \text{CLEAR}(f) \land \text{ONTABLE}(f) \land \\
\text{CLEAR}(g) \land \text{ONTABLE}(g) \land \text{EMPTY}(\text{arm})
\]

The first argument in the head specifies the optimal operator application, the second specifies the goal as a list of conjuncts, and the third specifies that the state contains a robot arm and the set of blocks \(\{d, e, f, g\}\). Again the constraint specifies the relations (ground atoms) that hold among the objects in the state.
In general a $CIG_{\geq \Sigma_5}$ of these two simple constrained atoms may contain a minimal
$\Sigma_5$-generalization for each of one or more of the matchings of blocks in the two states.
One matching is particularly interesting: the one that results from re-ordering the set in
the second simple constrained atom to $\{e, d, f, g\}$. The resulting least $\Sigma_5$-generalization
for this ordering is in the $CIG_{\geq \Sigma_5}$ and is the following.

\[
apply(\text{unstack}(x, y), [\text{ontop}(y, x), \text{clear}(y)], \text{domain}(\text{arm}, \{x, y, z, W\})) / \text{ontop}(x, y) \land \\
\text{clear}(x) \land \text{ontable}(z) \land \text{empty}(\text{arm})
\]

This simple constrained atom says that $\text{unstack}(x, y)$ is an optimal action for the goal
$\text{ontop}(y, x) \land \text{clear}(y)$, from a state that has blocks $x, y, z$, and possibly others, where
$x$ is on top of $y$, $x$ is clear, $z$ is on the table, and the robot arm is empty. This $\Sigma_5$-
egeneralization could not be obtained without re-ordering the elements of a set.

It is worth noting that throughout the dissertation we have ignored the issue of what
constitutes a good or appropriate constraint theory for a given application, such as the
Blocks World or U. S. politics. Rather, we have identified properties of constraint theories
that lead to particular results such as the pac-learnability of simple constrained atoms.
But in this section we actually have specified a particular part of the constraint theory,
namely the equational part $\Sigma_5$, and it is therefore proper to ask whether $\Sigma_5$ is appropriate
for structural domains. Rather than $\Sigma_5$, or in addition to $\Sigma_5$, we could encode in $\Sigma$ a
number of facts about sets and our representation of sets from our intended model. We
could encode an axiomatization of set theory, or some part of such an axiomatization.
For example, we could encode information about the union or intersection operations
on sets. The question of what information about sets to include is in some respects a
subjective one. The arguments at the beginning of this section indicate that $\Sigma_5$ captures
the aspect of sets that is important for structural domains, that two $\text{set}$-terms denote
the same set in the intended model if and only if one $\text{set}$-term results from re-ordering
the elements of the other $\text{set}$-term as one would re-order the elements of a list. Other
choices could be made, but it would be necessary to revise the proof of Theorem 115,
which follows.

172
We would like to use partitioned constrained $E$-anti-unification (Chapter 7) to solve this learning problem. But we have seen that partitioned constrained $E$-anti-unification requires exponential time and may produce a $CIG_{2^\Sigma}$ with multiple members. Therefore, it would seem that such an attempt will be unsuccessful.

In order to obtain a learning algorithm, we take some hints from Haussler [32]. To learn in structural domains, Haussler (1) restricts the number of objects in any example of the structural domain, and (2) uses subset queries. Following this lead, we place a bound $k$ on the size of any set-term and use subset queries. In addition, we incorporate the bounds and restrictions we have used in previous sections. It can be verified from the proof of Theorem 97 that restriction (1) reduces the factorial/exponential $((\frac{n}{m})!)^{2m+1}$ in the time for computing partitioned constrained $E$-anti-unification, where $n$ is the greatest head size among the input simple constrained atoms and $m$ is the number of these atoms, to $(k!)^{2m+1}$. We will use this operation in such a way that $m$ is always 2, so since $k$ and $m$ are constants, $(k!)^{2m+1}$ is a constant, and therefore partitioned constrained $E$-anti-unification can be computed efficiently. This approach leads to the positive result for polynomial learning by equivalence queries in Theorem 115 (below); the learning algorithm is a simple modification of the algorithm we used in Theorem 114. Afterward, we compare this result with Haussler’s original result.

It is worth noting also that with a bound on $k$, concept membership can be tested efficiently, given a constraint theory $\Sigma$ such that constraint queries regarding the equality-free part $\Sigma'$ of $\Sigma$ can be answered efficiently. To determine whether a given simple constrained set-atom is $\Sigma$-more general than another, simply test whether any $E$-transform (relative to $\Sigma$) of the second simple constrained set-atom is a $\Sigma'$-instance of the first simple constrained set-atom. For each $E$-transform, this test can be performed as for simple constrained atoms, in polynomial time (Section 6.3). Furthermore, the $E$-transforms are obtained from the $k!$ orderings of the set in the second simple constrained set-atom. Because $k$ is bounded, $k!$ is bounded by a constant, and clearly these $E$-transforms can be computed efficiently. Because concept membership can be computed efficiently, the positive result in Theorem 115 implies that simple constrained set-atoms can be pac-
learned, with the additional help of subset queries, under the same conditions required by Theorem 115.

**Theorem 115** Let $\mathcal{C}$ be the class of conjunctions of universally-closed simple constrained set-atoms in which (1) the size of the set-term in any atom is at most $k$, and (2) the maximum of the arities of the constraint predicates used is at most $r$. Let $\mathcal{D}$ be the class of universally-closed simple constrained set-atoms subject to the same restrictions. Let $\mathcal{T}$ be the class of Skolem Normal Form partitioned constraint theories, $\Sigma$, such that (1) $\Sigma$ has an initial model, (2) the equational part of $\Sigma$ is $\Sigma_5$, (3) the set function does not appear in the equality-free part of $\Sigma$, (4) constraint queries regarding the equality-free part of $\Sigma$ can be answered efficiently, (5) $\Sigma$ uses only a finite set of constraint predicates, and (6) the arity of any constraint predicate in $\Sigma$ is at most $r$. Then $\mathcal{C}$ is polynomially learnable by equivalence queries and subset queries, relative to $\mathcal{D}$ and $\mathcal{T}$.

**Proof:** The following algorithm polynomially learns $\mathcal{C}$ by equivalence queries and subset queries with respect to $\mathcal{D}$ and any constraint theory $\Sigma$ in $\mathcal{T}$. The algorithm first predicts $\perp$. If it is correct, the algorithm is finished. If it is incorrect, the only counterexamples are simple constrained set-atoms in $\mathcal{D}$ that are labeled as positive examples. One positive example, $\overline{\forall}(\phi/C)$, is received in response to the prediction. From this point on, the algorithm maintains a set of simple constrained set-atoms $S = \{\phi_1/C_1, ..., \phi_m/C_m\}$; initially, $m = 1$ and $\phi_1/C_1$ is $\phi/C$. At any point in the run, the algorithm’s next conjecture is the concept $\overline{\forall}(\phi_1/C_1) \land ... \land \overline{\forall}(\phi_m/C_m)$. If this conjecture is correct, the algorithm is finished. If it is incorrect, it receives a new counterexample $\overline{\forall}(\phi'/C')$; later, we will see that this must be a positive example. Then for all $1 \leq i \leq m$ the algorithm uses the obvious algorithm for partitioned constrained $E$-anti-unification (Chapter 7) to compute the $CIG_{\geq 2} G_i$ of $\phi_i/C_i$ and $\phi'/C'$. Each $G_i$ may contain at most $k!$ simple constrained set-atoms, none of which has a set-term of size greater than $k$ or a constraint predicate of arity greater than $r$. Because $k$ is fixed, this bound on the size of $G_i$ is a constant. For each $1 \leq i \leq m$ the algorithm then issues a subset query about each $\phi'_i/C'_i$ (more precisely, $\overline{\forall}(\phi'_i/C'_i)$) in $G_i$; if the answer to any subset query regarding a $\phi'_i/C'_i$ in $G_i$ is
yes, the algorithm replaces $\phi_i/C_i$ in $S$ by the first such $\phi'_i/C'_i$ in $G_i$. If the answer to every subset query for every $G_i$ is no, the algorithm adds $\phi'/C'$ to $S$. The algorithm is then prepared to make its next conjecture.

We now show by induction that any conjecture by the algorithm is logically weaker than the target, with respect to $\Sigma$ (the target $\Sigma$-entails the conjecture); it follows that each counterexample is a positive example. Clearly the first conjecture, $\bot$, is weaker than the target. Also for the basis, because the second conjecture, $\overline{\forall} (\phi/C)$, is also a positive example, it is logically weaker than the target with respect to $\Sigma$. Now assume, by the inductive hypothesis, that the most recent conjecture, $\overline{\forall}(\phi_1/C_1) \land ... \land \overline{\forall}(\phi_m/C_m)$, is logically weaker than the target with respect to $\Sigma$. Recall that the strong compactness property of simple constrained atoms (Theorem 25, Chapter 2) holds even when $\Sigma$ contains equality, provided $\Sigma$ is in Skolem Normal Form and has an initial model, so it applies here. Recall also that any simple constrained atom that is $\Sigma$-more general than a simple constrained set-atom is itself a simple constrained set-atom. It follows from strong compactness that, for all $1 \leq i \leq m$, the target contains a (universally-closed) simple constrained set-atom that is $\Sigma$-more general than $\phi_i/C_i$. In addition, because the conjecture is weaker than the target with respect to $\Sigma$, the counterexample $\overline{\forall}(\phi'/C')$ to the conjecture is a positive example. Again, from strong compactness, the target contains at least one simple constrained set-atom $\phi''/C''$ that is $\Sigma$-more general than $\phi'/C'$. If any such $\phi''/C''$ is also $\Sigma$-more general than some simple constrained set-atom(s) $\phi_i/C_i$ in the most recent conjecture, then the $CIG_{\Sigma} G_i$ of $\phi_i/C_i$ and $\phi'/C'$ by definition contains a simple constrained set-atom $\phi'_i/C'_i$ that is a $\Sigma$-instance of $\phi''/C''$. Replacing $\phi_i/C_i$ by any such $\phi'_i/C'_i$ will yield a new conjecture that is weaker than the target with respect to $\Sigma$. If no such $\phi''/C''$ is $\Sigma$-more general than any simple constrained set-atom in the most recent conjecture, then $\phi'/C'$ is added to $S$, so $\overline{\forall}(\phi'/C')$ is added to the conjecture to obtain the new conjecture. Because $\phi'/C'$ is a $\Sigma$-instance of a simple constrained set-atom in the target, the new conjecture is weaker than the target with respect to $\Sigma$.

Next, observe that if a new simple constrained set-atom $\phi'/C'$ is added to $S$, then $\phi'/C'$ is a $\Sigma$-instance of some simple constrained set-atom in the target of which no other
member of $S$ is a $\Sigma$-instance. It follows that $S$ contains at most the number of simple
constrained set-atoms in the target, so any conjecture contains at most the number of
simple constrained set-atoms in the target.

Now consider the time the algorithm requires between conjectures. The algorithm
for partitioned constrained $E$-anti-unification runs in time polynomial in the sizes of the
simple constrained set-atoms it receives as input, because the arity of constraint predi-
cates is at most $r$, the size of any set-term is at most $k$, exactly two simple constrained
set-atoms are provided as input, constraint queries regarding $\Sigma$ are answered efficiently,
and $\Sigma$ has a finite set of constraint predicates. Furthermore, the partitioned constrained
$E$-anti-unification algorithm produces simple constrained set-atoms that use constraint
predicates of arity at most $r$ and set-terms of size at most $k$. (Recall that the size of the
set-term in one simple constrained set-atom is no greater than the size of any set-term in
its $\Sigma$-instances.) To update $S$ requires at most $m$ calls to the algorithm for partitioned
constrained $E$-anti-unification and at most $m(k!)$ subset queries, where $m$ is the number
of conjuncts in the target. Therefore, because $k$ is fixed, the time required to update a
conjecture is polynomial in the sizes of the counterexamples seen thus far and the size of
the target.

Finally, we show that where $m$ is the number of simple constrained set-atoms in the
target, $K$ is the number of constraint predicates in $\Sigma$, $n$ is the maximum of the head
sizes of the counterexamples, and $L$ is the maximum number of constraint predicates in
some counterexample but not in $\Sigma$, the number of conjectures that the algorithm makes is
$m(Kn^r + Ln^r + n + 1)$. (Interestingly, this part of the proof is precisely the same as in the
proof of Theorem 114.) This will complete the proof, since for any particular $\Sigma$ this value
is polynomial in the size of the target and the sizes of the counterexamples. The maximum
number of times that a new simple constrained set-atom can be added to $S$ is $m$. The
maximum number of times that any of the (at most $m$) simple constrained set-atoms in
$S$ can be replaced by a $\Sigma$-more general simple constrained atom is $m(Kn^r + Ln^r + n)$,
since from Theorem 106 the maximum size of an ascending chain of simple constrained
set-atoms, relative to $\geq_{\Sigma}$, as described is $Kn^r + Ln^r + n + 1$.  

The theorem implies the pac-learnability of $\mathcal{C}$ relative to $\mathcal{D}$ and $\mathcal{T}$, with the additional help of subset queries.

How does this result relate to that of Haussler for structural domains? The examples and concepts in Haussler's result may be viewed as simple constrained set-atoms, and the background information may be viewed as a constraint theory, with the following additional restrictions beyond those we have used.

1. The head of any simple constrained set-atom is an atom built from a unary predicate.

2. The members of the set-term in the head of any example are all distinct constants.

3. The members of the set-term in the head of any concept description are all variables.

4. The set-terms in the heads of the simple constrained set-atoms are all the same size.

5. The constraint predicates in the constraint of the simple constrained set-atoms have arity at most 2.

6. The constraint predicates in the constraint theory have arity at most 1; that is, the constraint predicates are monadic predicates, or sorts, so the constraint theory is a sort theory.

7. The background theory meets the monomorphic tree restriction.

This leads us to observe four primary advantages of the present result. First, the algorithm for the present result learns a conjunction of simple constrained set-atoms (universally-closed), rather than a single simple constrained set-atom. Thus, in cases where no single simple constrained set-atom is consistent with the examples, the present algorithm may still succeed. Oddly enough, such concepts as this typically are called disjunctive concepts in machine learning research. The utility of learning disjunctive concepts is widely recognized. The second advantage is that constraint predicates may
have an *arbitrary* fixed arity. Thus the constraint theory may use binary predicates such as **ABOVE** in the Blocks World, for example. The third advantage is related, that the constraint theory may be much more expressive. For example, the theory \( \Sigma_6 \) about U. S. politics and the theory \( \Sigma_7 \) defining the **ABOVE** predicate for the Blocks World are not allowed in Haussler’s scheme. The fourth advantage is that arbitrary terms, rather than constants only, may be used in the heads of the examples.

It should be noted that the result of Theorem 115 is *not strictly* more general than Haussler’s result, because we have omitted the use of *linear* or *range* sorts. There are infinitely many *range* sorts, such as 1.5...2.3 that define ranges of real numbers. Haussler allows these, but we have omitted them from consideration throughout most of this dissertation by requiring a finite set of sorts or constraint predicates in \( \Sigma \). Chapter 9 discusses the issue of *range* sorts and how they might be added to our framework.

Finally, it should be noted that all of the positive learnability results we have obtained in this section also hold if \( D \) is the set of ground atoms. The proofs of the results remain the same in this case.

### 8.2 Speed-Up Learning: Natarajan’s Framework

#### 8.2.1 Defining the Framework

An algorithm for speed-up learning attempts to improve the performance (in run-time) of a problem solver over time. B. K. Natarajan [49] presents a formalization of speed-up learning that is analogous to the pac-learning model of inductive learning. In this model, we assume a set of states, a set of goals, and a finite set of operators. States and goals have associated sizes (representation size). Operators are functions that map states to states. We assume a program \( P_O \) that takes as input a state and an operator, and produces as output the next state (the state to which the input operator maps the input state), in time polynomial in the size of the input state. We also assume a program \( P_G \) that takes as input a state and a goal, and specifies whether the state *satisfies* the goal; this program also runs in time polynomial in the sizes of the input state and goal.
Together the sets of states, operators, and goals, and the programs $P_O$ and $P_G$ constitute a problem solving domain $M$.

Given a state and a goal, a solution is a sequence of operators that, when applied to the initial state, results in a state that satisfies the goal. An optimal solution is a shortest such sequence of operators. Examples in the model are all positive and are $\langle\text{state, goal, solution}\rangle$ tuples, where the solution is an optimal solution for the given state and goal. The step-size of a solution is the maximum of the sizes of the states generated by beginning with the initial state and applying the operators as specified in the solution. An algorithm for a particular goal, $D$, takes as input any state and produces a solution if one exists.

Each example, each optimal solution, consisting of $m$ operator applications provides $m$ tuples of the form $\langle\text{state, goal, operator}\rangle$, where the operator is an optimal first action in order to reach a state that satisfies the goal from a given state. In other words, operator is the first operator in some optimal solution. Natarajan shows that in any problem solving domain, if for any given goal, control rules specifying when an operator should be applied—more specifically, when an operator application is an optimal action—can be pac-learned from these examples in such a way that the rules have one-sided error, then with high probability an accurate, efficient planning algorithm can be learned efficiently. By one-sided error, Natarajan means that a rule never mistakenly asserts that a particular operator should be applied—its only mistakes are in failing to recognize some appropriate situations for the operator to be applied. As we might imagine, anti-unification of the $\langle\text{state, goal, operator}\rangle$ tuples for each operator can in some cases lead to the desired control rules. We first describe precisely what is meant by a control rule and then finish describing the learning model.

A control rule for a goal $D$ and an operator $o$ is a set of states $x$ such that $o$ is the first operator in some optimal solution given state $x$ and goal $D$.\footnote{Natarajan calls a control rule a projection.} We require that membership of a state in this set can be determined in time polynomial in the size of the state. Alternatively, we may view the control rule for a given goal and operator as a
mapping from states to \textit{positive} if the state is in this set and \textit{negative} otherwise. A set of control rules for a goal \( D \) contains one control rule for each operator. An approximation \( C' \) to a control rule \( C \) exhibits one-sided error if \( C' \) classifies a given state \( x \) as \textit{positive} only if \( C \) classifies \( x \) as \textit{positive}.

Examples to the learning algorithm are provided by a subroutine \textsc{Instance}, which Natarajan describes as a \textit{black box with a button}, such that at each push of the button, \textsc{Instance} outputs a randomly chosen solved instance of the input problem (goal) \( D \). The randomly chosen solved instance of \( D \) is a pair \( \langle x, \text{solution} \rangle \), where \( x \) is a state and \textit{solution} is an optimal solution to achieve \( D \) from state \( x \). The state \( x \) is chosen randomly according to some arbitrary but fixed probability distribution \( P \) over all states. Then \textit{solution} is a \textit{randomly chosen optimal solution for} \( x \) (with goal \( D \)), being the null-sequence if \( x \) is not solvable or ... is solved as it is. By randomly chosen optimal solution, Natarajan means that at any stage in the solution of \( x \), the next operator application used by \textsc{Instance} is picked randomly, according to the uniform distribution, from among those that lead to a shortest solution.

We now present Natarajan’s precise specification of what is meant by \textit{with high probability an accurate, efficient planning algorithm can be efficiently learned} for a given problem solving domain. We say that \textit{with high probability an accurate, efficient planning algorithm can be efficiently learned} for a problem solving domain \( M \) if and only if there exists a \textit{meta-algorithm} for \( M \). Natarajan defines \textit{meta-algorithm} as follows (modified slightly here for our more general definitions).

\textbf{Definition 116} An algorithm \( A \) is a meta-algorithm for a problem solving domain \( M \) if there exists an integer \( k \) such that:

1. \( A \) takes as input an integer \( h \) (the error parameter) and the specification of a problem (goal) \( D \). Let \( l \) be the string length of this input.

2. \( A \) may call \textsc{Instance}. \textsc{Instance} returns examples for \( D \), chosen according to some unknown distribution \( P \). Let \( n \) be the maximum step-size and \( m \) the maximum length of the solutions provided by \textsc{Instance}. Let \( t(n) \) be the sum of the running
times of the programs \( P_O \) and \( P_G \) on inputs of length \( n \). A computes in time \( O(\text{thm}(n))^k \), that is, in time polynomial in the length of its input \( l \), the error parameter \( h \), and the time required to evaluate \( P_O \) and \( P_G \) on the examples seen. A may be a randomized algorithm.

3. For all goals \( D \) and all probability distributions \( P \) over the states, A outputs a (possibly randomized) program \( H \) that runs in time \( O(t(n)^k) \) on inputs of length \( n \) and approximates an algorithm for \( D \) in the sense that

\[
\sum_{\{x: H \text{ is correct on } x\}} P(x) \leq \frac{1}{h}
\]

Since \( H \) may be randomized, by “\( H \text{ is correct on } x \)” is meant that \( H \) generates a solution from initial state \( x \) with probability greater than \( \frac{1}{h} \), if a solution exists.

Natarajan shows that a problem solving domain has a meta-algorithm if for each goal a class \( C \) of control rules exists that (1) contains the control rules for every goal \( D \) and (2) is pac-learnable with one-sided error. The method used is to use the examples returned by INSTANCE to generate examples of the form \( \langle \text{state}, \ \text{goal}, \ \text{operator} \rangle \), and then use these examples to pac-learn the control rules with one-sided error. The control rules are then used to direct a problem solver.

### 8.2.2 An Application to Symbolic Integration

As an example of his framework, Natarajan describes a learning program, similar to LEX [46; 63; 62], that learns control rules for solving simple indefinite integrals. (The goal is always the same, to solve the integral.) These rules are obtained by a method that amounts to using sorted anti-unification to compute the \( LGG_{\geq2} \)s of the descriptions of states, as sorted atoms, in which particular operators are applied. \( LGG_{\geq2} \)s are computed with respect to a concept hierarchy that is equivalent to a sort theory \( \Sigma \) meeting the monomorphic tree restriction (see the end of Section 5.3). For example, suppose the
operator that rewrites any integral of the form $\int k f(x) \, dx$, where $k$ is a constant and $f$ is a function, to $k \int f(x) \, dx$ is the first operator in optimal solutions of both $\int 5 \cos x \, dx$ and $\int 3 \sin x \, dx$. Then the $LGG_{\Sigma}$ of these examples is a rule that says this operator should be applied to any integral of the form $\int \text{INTEGER TRIG} \, dx$. $LGG_{\Sigma}$s are guaranteed, even though the method used to compute them is sorted anti-unification, because $\Sigma$ meets the monomorphic tree restriction.

Interestingly, the background theory that Natarajan uses is much more restricted than the background theory used in LEX. Is this just to make the example easier to understand? In fact, such a more restricted theory is necessary; we will now see, based on our study of sorted anti-unification, that the theory used in LEX is not guaranteed to yield $LGG_{\Sigma}$s, but may in fact yield $CIG_{\Sigma}$s that can have exponentially many members (exponential in the sizes of the input sorted atoms). Therefore, Natarajan’s example does not directly imply any kind of learnability result for LEX.

The background theory for LEX is a context-free grammar that is used as a sort theory. Figure 8.1 shows the part of this theory that is relevant to our discussion.

An expression such as $(\text{op expr expr})$ may be viewed as a sort. Its members are denoted by terms of the form $(\neg \text{expr expr})$, $(+ \text{expr expr})$, $(\times \text{expr expr})$, $(/ \text{expr expr})$, $(\uparrow \text{expr expr})$. Notice that not every member of $\text{monom}$ is a member of $(\text{op expr expr})$, and not every member of $(\text{op expr expr})$ is a member of $\text{monom}$, but the two do have some members in common. Therefore, some sorted terms may have multiple minimal variable generalizations, and therefore multiple minimal $\Sigma$-generalizations. For example, the terms $(\star x 2)$ and $(\uparrow x 3)$ have two minimal $\Sigma$-generalizations: $\text{monom}$ and $(\text{op expr expr})$. In addition to multiple minimal variable generalizations, $\star$ and $\uparrow$ may be viewed as function symbols denoting polymorphic functions. For example, $\uparrow$ is polymorphic because a term such as $(\uparrow x 3)$ denotes a member of the sort $\text{monom}$, but not all terms built from $\uparrow$, such as $(\uparrow x 3.5)$, denote members of $\text{monom}$. The full theory used by LEX contains other interesting examples of function polymorphism, but the ones involving $\uparrow$ and $\star$ are the simplest. We use the first example, of multiple minimal variable generalizations rather than polymorphism, to show how a set of minimal generalizations $(CIG_{\Sigma})$ can
be exponentially large. Some of the cases of function polymorphism in LEX can be used
in a similar manner.

Consider a pair of terms such as

\[(+ (+ (+ (+ x_1 2) (↑ x_2 5)) (* x_3 6)) (* x_4 4)) (↑ x_5 3))\]
\[(+ (+ (+ (↑ y_1 3) (* y_2 11)) (↑ y_3 8)) (↑ y_4 2)) (* y_5 9))\]

These of course represent the arithmetic expressions \(x_1^2 + x_2^5 + 6x_3 + 4x_4 + x_5^3\) and
\(3y_1 + 11y_2 + y_3^8 + y_4^2 + 9y_5\). The terms \((↑ x 2)\) and \((* x 3)\) have two minimal \(Σ\)-
generalizations: \(monom\) and \((op expr expr)\). Similarly, four other pairs of terms have two
minimal \(Σ\)-generalizations each. But computing the minimal \(Σ\)-generalizations of terms
built from the function \(+\) involves taking cross-products of these other \(Σ\)-generalizations,
so the pair of terms in the example yields 32 minimal \(Σ\)-generalizations, as follows.

\[(+ (+ (+ (op expr expr) (op expr expr)) (op expr expr)) (op expr expr)) (op expr expr))\]
\[(+ (+ (+ (+ monom (op expr expr)) (op expr expr)) (op expr expr)) (op expr expr)) (op expr expr))\]
\[
(+ (+ (+ (+ \text{expr \ expr} \ \text{monom} \ \text{monom} \ \text{monom} \ \text{monom}) \ \text{monom} \ \text{monom} \ \text{monom})) \ \text{monom}) \ \text{monom})
\]

It is easy to see that for such expressions the number of minimal $\Sigma$-generalizations is exponential in the number of occurrences of the + function in both of the expressions being generalized. Thus even longer examples can give far more minimal $\Sigma$-generalizations. And the situation is worse yet, because in LEX each of these $\Sigma$-generalizations in turn must be be generalized with new examples.

Nevertheless, our study of anti-unification reveals that Natarajan’s framework in fact can be applied to sort theories as expressive as the one used by LEX. The solution, simply said, is to use extended sorted anti-unification rather than sorted anti-unification, and to represent control rules as extended sorted atoms rather than as sorted atoms. We know that extended sorted anti-unification can be used to pac-learn extended sorted atoms. And by the definition of anti-unification, the extended sorted atoms that are learned in this way exhibit one-sided error. Of course, it remains to verify that the theory LEX uses is rich enough for the expression of control rules that provide optimal solutions to all the symbolic integration problems it sees; we shall not embark on that venture.

The general message of this example is that, where we have a pac-learning algorithm based on any of the forms of anti-unification we have studied, we more specifically have a pac-learning algorithm that exhibits one-sided error. This follows directly from the correctness of the anti-unification algorithms and the semantic property in Theorem 14. Thus it might be possible to use other anti-unification algorithms for provably successful speed-up learning in Natarajan’s framework. For example, it might be possible to use partial constrained $E$-anti-unification, with simple constrained set-atoms and a theory $\Sigma$ whose equational part is $\Sigma_5$ (about sets), for provably successful speed-up learning in structural domains such as the Blocks World. (See [41] for an experimental study of applying induction to speed-up learning in the Blocks World.)
8.3 Knowledge Base Vivification

Knowledge base vivification is an approach to efficient deduction [8; 20; 42]. 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 [42], suppose our KB includes $age(fred,71) \lor age(fred,72)$. 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 anti-unification to replace $age(fred,71) \lor age(fred,72)$ with $\exists x:LOW-SEVENTIES age(fred,x:LOW-SEVENTIES)$.

Most examples of knowledge base vivification are of this simple form, replacing constants such as 71 or 72 with existentially-quantified variables that belong to some taxonomic class such as LOW-SEVENTIES. But many interesting disjunctions may include non-constant terms, and the terms in the disjuncts may share significant properties other than common taxonomic categories. Therefore, for knowledge base vivification to reach the point of practical application, more sophisticated techniques of vivification are needed that can deal with (1) terms built from non-nullary function symbols, and (2) background information involving higher-arity predicates, such as BIGGER, rather than only unary, taxonomic predicates such as LOW-SEVENTIES. This section shows how our study of anti-unification leads to such techniques.

In general, the vivification problem is to replace a disjunction with some sentence that is non-disjunctive and is logically weaker than the disjunction, relative to some non-disjunctive portion of the KB. (We refer to this non-disjunctive portion of the KB as the background theory.) Furthermore, we would like the logically strongest such sentence, relative to the background theory.

Sorted anti-unification with a background theory meeting the monomorphic tree restriction may be used to compute the typical, simple examples of knowledge base vivification. In fact, it even allows the incorporation of higher-arity functions. Because
simple constrained anti-unification is more efficient than sorted anti-unification, and uses more expressive languages, in many cases simple constrained anti-unification is even more appropriate for vivification. To illustrate the advantages of simple constrained anti-unification over sorted anti-unification, imagine the following example. Suppose one night you see (or hear) Fred or Joe, you are not sure which, in what you believe to be a domestic quarrel. This is of particular interest because Fred and Joe are family counselors whose wives are professionals (Fred’s wife is a doctor while Joe’s wife is a math teacher). You might represent this observation as the disjunction

\[ \text{argued fred, wife(fred)} \lor \text{argued(joe, wife(joe))} \]

The \(LGG_{\Sigma}\) resulting from simple constrained anti-unification of the disjuncts captures all that is likely to be interesting about the observation, when this \(LGG_{\Sigma}\) is existentially-closed.

\[ \exists (\text{argued}(x, \text{wife}(x))) / \text{FAMILY-COUNSELOR}(x) \land \text{PROFESSIONAL}(\text{wife}(x))) \]

This sentence can also be written as follows.

\[ \exists x \ (\text{FAMILY-COUNSELOR}(x) \land \text{PROFESSIONAL}(\text{wife}(x)) \land \text{argued}(x, \text{wife}(x))) \]

It says “there exists an \(x\) such that \(x\) is a family counselor, the wife of \(x\) is a professional, and \(x\) argued with his wife.” Because of the semantic properties of simple constrained anti-unification (Theorem 14), this statement is the strongest existentially-closed simple constrained atom that logically follows from the observation and the background information. (This result is true provided the KB is equality-free; in the remainder of this section we include this assumption.) Notice that if sorted anti-unification were used instead, it would give two minimal \(\Sigma\)-generalizations. These are

\[ \exists (\text{argued}(x, \text{wife}(x))) / \text{FAMILY-COUNSELOR}(x)) \]
$\exists (\text{argued}(x, y) \land \text{family-counselor}(x) \land \text{professional}(y))$

These may also be written as follows.

$\exists x \ (\text{family-counselor}(x) \land \text{argued}(x, \text{wife}(x)))$

$\exists x \exists y \ (\text{family-counselor}(x) \land \text{professional}(y) \land \text{argued}(x, y))$

Even if both of these are added to the knowledge base, the resulting knowledge base contains less information than if the $LGG_{\Sigma}$ computed by simple constrained anti-unification were added instead. Also, recall that in general the number of minimal $\Sigma$-generalizations that sorted anti-unification produces can be exponential in the size of the smallest disjunct, whereas simple constrained anti-unification always produces an $LGG_{\Sigma}$. This example shows not only the utility of anti-unification operations in knowledge base vivification, but also the importance of choosing the right operation. Thus our study of anti-unification has led to a better operation—simple constrained anti-unification—for knowledge base vivification, in terms of both expressivity and computational complexity, than the standard operation—sorted anti-unification.

We now consider an example involving binary constraint predicates; this is an example that first appeared in Chapter 6. Suppose our knowledge base contains the following sentence.

$$\text{intimidates}(\text{son}(\text{jumbo}), \text{son}(\text{clyde})) \lor \text{intimidates}(\text{son}(\text{fred}), \text{son}(\text{joe}))$$

Then if we also have the information in $\Sigma_4$, we can replace the sentence with the existentially closed $LGG_{\Sigma_4}$ of the disjuncts, which is

$\exists (\text{intimidates}(\text{son}(x_2), \text{son}(y_2)) \lor \text{bigger}(x_2, y_2) \land \text{bigger}(\text{son}(x_2), \text{son}(y_2)))$
This sentence may also be written as

$$\exists (\text{bigger}(x_2, y_2) \land \text{bigger}(\text{son}(x_2), \text{son}(y_2)) \land \text{intimidates}(\text{son}(x_2), \text{son}(y_2)))$$

While some information is lost in the replacement, much of the information is retained, and the disjunction is removed.

But does simple constrained anti-unification necessarily give the desired vivid form of a disjunction? Does it always give the strongest consequent of a disjunction together with specified knowledge from the knowledge base? According to Theorem 14, about the semantic properties of the $\geq_\Sigma$ ordering, it does, if the disjuncts are equivalent to ground or existentially quantified constrained atoms, the specified background knowledge from the KB forms an equality-free constraint theory (and so does not use a predicate that appears in the disjuncts), and the sentence that replaces the disjunction is required to be equivalent to a ground or existentially quantified simple constrained atom. Of course, if we wish to obtain some other form of non-disjunctive sentence, other methods are necessary. The search for efficient anti-unification operations for more expressive languages is certainly an important topic for further research that could contribute to the ability to compute other forms of non-disjunctive sentences. It is also worth noting that other techniques are necessary for vivification if the predicates that appear in the disjuncts may also appear in the background theory.
Chapter 9

Conclusions and Directions for
Further Research

Five years after Alan Robinson introduced unification, Gordon Plotkin and J. C. Reynolds independently introduced its dual, generalization, or ordinary anti-unification [56; 60]. Whereas Robinson developed unification solely for its application to automated deduction, Plotkin posited the utility of ordinary anti-unification in machine learning and Reynolds investigated an application of ordinary anti-unification to automated deduction. But while unification research proliferated over the next twenty years, particularly within the automated deduction and logic programming communities, the dual of unification was largely ignored, in spite of its wide range of potential applications. Therefore, anti-unification lags far behind unification today in both pure research and applications. In particular, unification now comes in a variety of forms, each of which has a dual form of anti-unification, but many of these anti-unification operations have not been studied or even defined.

This dissertation has argued that anti-unification, in its various forms, should not be ignored, because its properties suit it to several of the most important reasoning tasks studied in artificial intelligence: inductive learning of concepts in first-order logic (inductive logic programming), speed-up learning, and efficient automated deduction (by
knowledge base vivification). The dissertation has focused on forms of anti-unification operating on subsets of constraint logic.

In fact, the application of anti-unification is increasing already. In the last five years, concurrent with the work on this dissertation, other inductive logic programming researchers [47; 48; 10] have revived the work on anti-unification by Plotkin and Reynolds, applying it to the problem of learning definite clause theories. In particular, some of the recent work on learnability in inductive logic programming uses ordinary anti-unification in this way [12; 19]. In addition, Peter Idestam-Almquist has investigated a form of anti-unification that applies to a restricted class of recursive Horn clauses [33; 34], based on related work by Lapointe and Matwin [37]. Rob Hasker is investigating anti-unification in higher order logic and its potential applications for programming languages [30; 31]. The semantic properties of anti-unification indicate that it may be useful for database systems as well, particularly for systems that incorporate constraints [36; 38; 59]. In general, the study of the theory and applications of anti-unification is a promising direction for further research.

In addition to further research into the theoretical properties and the applications of various anti-unification operations, the dissertation has indicated several other areas for further research. These of course are also related to anti-unification. We mention these briefly now.

First, all the results presented in Chapter 8 for inductive logic programming are on the learnability of non-recursive definite clauses or definite clause theories. The learnability of definite clause theories that may exhibit recursion is a fascinating area for further research. Several recent papers have begun the investigation into this topic [12; 11; 19; 21; 22].

Second, applying partitioned constrained $E$-anti-unification to speed-up learning for structural domains, in Natarajan’s framework, could yield some of the first theoretical results for speed-up learning in structural domains. In general, Natarajan’s framework provides an intuitively-appealing formalization of the speed-up learning task, and anti-unification operations appear uniquely suited for use with this formalization. The search
for new positive results in this framework, based on anti-unification, is an interesting area for further research. Even more generally, are there other formalizations of speed-up learning that should be considered as well, and is anti-unification useful for these?

Third, the anti-unification algorithms and the positive learnability results presented in the dissertation assume a constraint theory that uses only a finite set of constraint predicates. This assumption disallows the use of infinitely many range sorts, such as 2.4...3.5 that include exactly the real numbers in a given interval. If range sorts are allowed, it is impossible to construct algorithms that learn the various classes of constrained atoms by equivalence queries. Nevertheless, perhaps pac-learning is possible using anti-unification operations. While infinite ascending chains may exist when range sorts are allowed, there may be other approaches to proving that the VC-dimension of a restricted class of constrained atoms grows only polynomially with the size of the constrained atoms allowed. Such proofs could lead to pac-learning results in the presence of range sorts.
Appendix

A Short Glossary

A.1 Restricted Constrained Atoms

**Extended sorted atom:** A constrained atom whose constraint is a conjunction of atoms built from *unary* constraint predicates and from terms in the head.

**Simple constrained atom:** A constrained atom whose constraint is a conjunction of atoms built from constraint predicates and from terms in the head.

**Simple constrained *set*-atom:** A simple constrained atom meeting the following restrictions: at most one argument in the head contains the *set* function symbol, this argument is a *set*-term, and the *set* function symbol does not appear in the constraint. A *set*-term is recursively defined to be any of: *nil*, any variable, or any term of the form \( \text{set}(t_1, t_2) \), where \( t_1 \) and \( t_2 \) are *set*-terms.

**Sorted atom:** A constrained atom whose constraint is a conjunction of atoms built from unary constraint predicates and from variables in the head, such that each variable in the head appears at most once in the constraint.
A.2 Anti-Unification Operations

For most of the following anti-unification operations, additional restrictions are sometimes imposed to ensure efficient computation. The restrictions are not given in the general definitions, but are explicitly stated in the dissertation wherever they apply.

E-Anti-Unification: Anti-unification of ordinary atoms relative to an equational theory.

Extended Sorted Anti-Unification: Anti-unification of extended sorted atoms relative to a sort theory (a constraint theory using only unary constraint predicates, which by default is equality-free).

Ordinary Anti-unification: Anti-unification of ordinary atoms relative to the empty constraint theory.

Partitioned Constrained E-Anti-Unification: Anti-unification of simple constrained atoms relative to a constraint theory that is partitioned into an equational part and an equality-free part.

Simple Constrained Anti-Unification: Anti-unification of simple constrained atoms relative to an equality-free constraint theory.

Sorted Anti-Unification: Anti-unification of sorted atoms relative to a sort theory.
Bibliography


198


Vita

Charles David Page Jr. was born on March 22, 1963 in Winston-Salem, North Carolina. He received a B.A. in political science from Clemson University in 1985, where he also received the 1985 James F. Byrnes American Government Award for scholarship in political science. He received an M.S. in computer science from Clemson University in 1987. Since coming to the University of Illinois, his work includes the following papers.


