A Generalized Approach to Equational Unification

by

Katherine Anne Yelick

This research was supported in part by the National Science Foundation under grant MCS-8119846-A01 and by the Defense Advanced Research Projects Agency monitored by the Office of Naval Research under contract number N00014-83-K-0125.

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Massachusetts 02139

A Generalized Approach to Equational Unification

by

Katherine Anne Yelick

Abstract

Given a set of equational axioms and two terms containing function symbols and variables, the equational unification problem is to find a uniform replacement of terms for the variables that makes the terms provably equal from the axioms. In the variable-only case, the two terms contain only variables and function symbols from the axioms. In the general case, the terms may contain symbols not appearing in the axioms, there may be more than one instance of a set of axioms, and there may be more than one set of axioms.

This thesis presents a method for combining equational unification algorithms to handle the terms with "mixed" sets of function symbols. For example, given one algorithm for unifying associative-commutative operators, and another for unifying commutative operators, our algorithm provides a method for unifying terms containing both kinds of operators. It is based on a general strategy for decomposing terms and combining unifiers. We restrict our attention to sets of axioms whose function symbols are pairwise disjoint.

A simplifying assumption is that we are working only with confined regular equational theories, a class of theories defined in this thesis. We present a unification algorithm that solves the general case unification problem for any combination of these theories, given variable-only case algorithms for the theories. The algorithm is proven totally correct. The termination proof is a generalization of Fages' proof of termination for associative-commutative unification.

Our algorithm has been implemented as part of a larger system for generating and reasoning about equational term rewriting systems.

Thesis supervisor: John V. Guttag
Title: Associate Professor of Computer Science and Engineering
Keywords: Unification, Equational Unification, Equational Theories, Confined Regular Theory, Unification Algorithm, Automatic Theorem Proving, Term Rewriting, Resolution

This thesis was submitted to the Department of Electrical Engineering and Computer Science on August 1, 1985 in partial fulfillment of the requirements for the Degrees of Bachelor of Science and Master of Science in Computer Science.

Acknowledgments

I would like to thank my advisor, John Guttag, for his support and encouragement and for his unfailing confidence in this work. John's comments on organization and presentation have helped make this thesis readable, and his prompt and careful reading of my drafts has been invaluable in the completion of this thesis.

In the early stages of this work, Randy Forgaard taught me the basics of term rewriting and unification, and Gene Stark showed interest in solving the problems, and contributed ideas that directly influenced the structure of the completeness proof. A discussion with Jean-Pierre Jouannaud, Claude Kirchner, and Helene Kirchner on the common properties of E-unification algorithms spurred the idea of a general unification algorithm. Pierre Lescanne and Paris Kanellakis answered a stream of questions related to theory and logic, and Val Breazu encouraged me to address the problem of proving termination, which would otherwise have been left for future work. Since taking over the task of developing and maintaining REVE-2, Dave Detlefs has often lent a sympathetic ear and technical advice.

My friends have been a great help in keeping my spirits up. I commend the members of the graduate crew team for their unique sense of humor and enthusiasm for rowing. Joe Zachary, Brian Coan, Julie Lancaster, and Ron Kownacki served as an audience for both technical and non-technical discussions. I would especially like to thank Jennifer Lundelius, Jim Restivo, and Tom Wanuga, who provided an important mix of inspiration, sympathy, and unfailing good humor.

Finally, I am very grateful to my parents, who have been patient and supportive during my undergraduate and graduate studies.

Table of Contents

Chapter One: Introduction 7

1.1 Organization of the Thesis 8
1.2 Definitions 9
1.3 Classical Unification 10
1.4 Equational Unification 12
1.4.1 Applications 12
1.4.2 Equational Theories 14
1.4.3 A Problem Statement 16
1.4.3.1 Properties of Unification Algorithms 16
1.4.3.2 Classifying Solutions to the Unification Problem 18
1.5 Related Work 19
1.5.1 Single Theory Algorithms 19
1.5.2 Narrowing 24
1.5.3 Combining Theories 24

Chapter Two: A Generalized Unification Algorithm 26

2.1 A Generalized Approach 26
2.1.1 Partitioning Equational Theories 26
2.1.2 Some Basic Functions 28
2.1.2.1 Homogeneous Terms 29
2.1.2.2 Unification of Substitutions 30
2.1.3 Restrictions 31
2.1.3.1 Confined Theories 31
2.1.3.2 Regular Theories 31
2.1.3.3 Strict and Strongly Complete Theories 32
2.2 The Algorithm 34
2.3 An Example 36
2.4 Difficulties in Exending *CR-unify* 38

Chapter Three: Proof of Total Correctness 42

3.1 An Overview 42
3.1.1 Definitions for Terms in E* 44
3.1.2 Properties of Confined and Regular Theories 46
3.1.2.1 Confined Theories 46
3.1.2.2 Regular Theories 48
3.2 Consistency 52
3.3 Completeness 54
3.3.1 A New Homogenizing Operation 54
3.3.1.1 Homogeneity Using U 55
3.3.1.2 The Inverse Substitution 59
3.3.2 The Completeness Lemmas 61
3.3.3 A Proof of Completeness 63
3.4 Proof of Termination 66
3.4.1 Noetherian Induction 67

3.4.2 A Noetherian Ordering for E-Unification 67
3.4.3 Some Properties of the Ordering 68
3.4.4 The Proof 73

Chapter Four: Conclusions 79

4.1 Contributions 79
4.2 Future Work 83
4.2.1 Efficiency Issues 83
4.2.2 Removing Restrictions on the Theories 85

Appendix A: Protection of Variables in *CR-unify* 86

Appendix B: Glossary of Terms 91

Appendix C: Special Symbols 95

Table of Figures

Figure 1-1: A Classical Unification Algorithm 11
Figure 1-2: A Prolog Program with Commutativity 13
Figure 1-3: Some Common Equational Theories 22
Figure 1-4: Known Unification Algorithms 23
Figure 2-1: A Partitioned Presentation 28
Figure 2-2: Procedure *map-unify* for Unification of Substitutions 30
Figure 2-3: The *CR-unify* Procedure for Equational Unification 35
Figure 3-1: Diagram Exemplifying Correctness Properties 43
Figure 3-2: A Term and its Significant Subterms 45
Figure 3-3: Commuting Diagram for the Universal E-preserving Substitution 60
Figure 3-4: A Diagram of the Completeness Lemma 62
Figure 4-1: A Careful Description of the *CR-unify* Procedure 88
Figure 4-2: A Careful Description of the *map-unify* Procedure 89

Theorems, Lemmas, and Propositions

Lemma 1: E is confined if and only if E* is confined 47

Lemma 2: $t \underset{E}{=} s \Rightarrow t.head \underset{\pi}{=} s.head$ 47

Theorem I: $t.head \underset{\pi}{\neq} s.head \Rightarrow t$ and s are not unifiable 48

Lemma 3: E is regular if and only if E* is regular 48

Lemma 4: $t \underset{E}{=} s \Rightarrow \forall\, v \in V$, $ParSets(v, t) = ParSets(v, s)$ 49

Lemma 5: $t \underset{E}{=} s \Rightarrow \forall\, t' \preceq_{\mathcal{J}} t\ \exists\, s' \preceq_{\mathcal{J}} s$ such that $t' \underset{E}{=} s'$ 50

Lemma 6: $t \preceq r \prec_{\mathcal{J}} s \Rightarrow t \underset{E}{\neq} s$ 51

Theorem II: $r \prec_{\mathcal{J}} s$ & $v \in \mathfrak{V}(r) \Rightarrow v$ and s are not unifiable 52

Lemma 7: Consistency Lemma: $\rho\hat{t} \underset{E_i}{=} \rho\hat{s}$ & $\sigma \circ \rho \underset{E}{=} \sigma \circ \gamma \Rightarrow \sigma t \underset{E}{=} \sigma s$ 52

Theorem III: *CR-unify* is Consistent 53

Proposition 1: The *U-HomogMap* form of the Preserving Substitution Maps $\hat{t}$ to $\bar{t}$ 56

Lemma 8: *U-Homog* Commutes with Application of Substitutions 56

Lemma 9: $\overline{\sigma_1 \circ \sigma_2} = \bar{\sigma}_1 \circ \bar{\sigma}_2$ 57

Lemma 10: $t \underset{E}{=} s \Rightarrow \bar{t} \underset{E_i}{=} \bar{s}$ 58

Proposition 2: μ Preserves Homogenized Terms Within E* 59

Lemma 11: $\sigma \circ \mu \underset{E}{=} \mu \circ \bar{\sigma}$ 60

Lemma 12: If t and s are E-unifiable then $\hat{t}$ and $\hat{s}$ are E_i-unifiable. 61

Lemma 13: Completeness Lemma: $\sigma \circ \mu \circ \rho \underset{E}{=} \sigma \circ \mu \circ \gamma$ 62

Theorem IV: *CR-unify* is Complete 63

Lemma 14: The $\prec_{\mathcal{C}}$ Ordering is Noetherian 68

Lemma 15: Complexity Decreases for Non-variable Significant Subterms 68

Lemma 16: Complexity Decreases for Variable with More than One Parent 69

Lemma 17: Elementary Non-Increasing Substitution do not Increase Complexity 70

Lemma 18: Non-increasing Substitutions do not Increase Complexity 72

Lemma 19: Non-increasing Substitutions on Subterms are Non-increasing 72

Theorem V: *CR-unify* Terminates 74

Chapter One

Introduction

The unification problems are a class of problems involving a general form of pattern matching. As such, they occur independently in many contexts that involve symbol manipulation, the most pronounced of these being automated theorem proving. A unification problem can be formulated as a problem on strings, graphs, or algebraic objects such as sets or groups.

The classical unification problem is: given two terms containing function symbols and variables, find a uniform replacement of terms for the variables that makes the two terms syntactically identical. Equational unification, or E-unification, extends the classical problem to solving an equation in an equational theory. That is, given a set of equational axioms, find a substitution for the variables in the two terms that makes them provably equal from the set of axioms describing the theory.

In this thesis, we develop a framework for automatically combining E-unification algorithms for independent sets of operators by carefully analyzing the E-unification process in general and studying a number of equational theories in particular. Our approach is generalized in the following sense: Given a unification algorithm for E_1, and a unification algorithm for E_2, we can in some cases automatically generate an algorithm for the combined theory of E_1 and E_2, such that the resulting algorithm will unify terms with mixed set of operators.

Unification was first described by Herbrand in 1930, and was first put to practical use by Robinson as a basic step in resolution [Robinson 65], an inference rule used as a complete proof system for first order predicate calculus. Because of its simplicity and power, the resolution rule is often used as the basis for automatic theorem provers and is also exploited in implementing the logic programming language Prolog [Kowalski 74, Clocksin 81]. In Prolog, unification acts as a procedure call mechanism, allowing procedures to be invoked when the arguments fit the pattern given in the procedure head.

Unification is also used in type inference algorithms for languages such as ML [Milner 78], in which type inference is used as a compromise between strictly typed and typeless languages. These languages gain expressive power over explicitly typed languages that enforce strong type checking, because the type inference provides the programmer with a mechanism for a certain kind of polymorphism [Mitchell 84].

Unification is also an important operation in *term rewriting systems.* These systems perform reasoning by compiling equations into a set of *rewrite rules.* This compilation, known as the completion process, involves ordering each equation into a directed rule, finding pairs of rules that could apply to a single term, and sometimes adding new rules when such critical pairs are found. Unification is used in finding the pairs of overlapping rules and in generating the additional rules. If the completion process terminates successfully, the resulting system is called *convergent*, meaning the rules, applied in any order to a given input, will always result in a unique answer [Knuth 70]. A convergent rewriting system is a complete and terminating decision procedure for determining whether or not an equation is implied by the original set of equations. Term rewriting systems can be used as a basis for automatic theorem provers [Huet 82, Kapur 84, Goguen 80, Hsiang 82]. These theorem provers have been used for applications such as checking formal specifications [Goguen 79, Guttag 83, Kownacki 84], interpreting logic programming languages [Dershowitz 83a, Fribourg 84], reasoning about relational databases [Cosmadakis 85], and checking properties of petri nets [Choppy 85].

The unification algorithm described in this thesis was initially motivated by the need to extend the domain of applications for which the REVE term rewriting system generator [Lescanne 83, Forgaard 84a] is useful. Because unification problems occur in many different applications, there are both theoretical and pragmatic reasons for developing a better understanding of the problem and its solutions. A condensed version of the work described in this thesis appeared previously in [Yelick 85].

1.1 Organization of the Thesis

The remainder of this chapter is devoted to giving the background necessary for understanding unification, both classical and equational, and discussing related work in the field. The rest of the thesis is divided into three chapters. Chapter 2 presents our generalization of the problem along with some some restrictions, gives our algorithm, and goes through a non-trivial example. Chapter 3 presents a proof of total correctness for the generalized algorithm. Chapter 4 presents a summary of our conclusions, a description of the implementation, and ideas for future work. The reader can get of clear picture of our approach, ignoring the question of why the algorithm works, by reading Chapters 2 and 4. Three appendices are included. Appendix A gives some additional technical details of the proof of correctness, Appendix B gives a glossary of some terms used in this thesis, and Appendix C gives a list of special symbols and there uses.

1.2 Definitions

The following definitions are consistent with the definitions of [Fages 84] and [Huet 80a]. We begin with basic definitions of terms and functions on terms.

Let **V** be a countable set of variables and **F** be a family of function symbols with associated arity such that V and F are disjoint. We recursively define the set of terms, T(F, V), as either a variable or a function symbol of arity n, followed by n terms. We assume the sets **V** and **F** to be fixed and, thus, use **T** in place of T(F, V) without ambiguity. Function symbols of arity zero, called *constants*, will be denoted by the letters a, b, c, d, and numerals 0, 1, to be distinguished from variables, denoted by the letters u, v, w, x, y, z. For readability, we will use the symbols, $+$, $*$, and $\bullet$ as binary infix operators. Examples of terms are $f(x, a)$, $h(x * 0)$, and $y + 1$.

Given a term, t, let $\mathcal{V}(t)$ be the set of variables in t and $\mathcal{F}(t)$ be the set of function symbols in t. The root symbol of the graph representation of a term, t, will be denoted *t.head*—*t.head* is a variable if t is a variable, and a function symbol if t is not a variable. Terms formed from function symbols alone, i.e., containing no variables, are called *ground terms* and are denoted by **G**.

An *occurrence* in a term names a node within the tree structure of the term; occurrences are represented by strings of integers, including the empty string, ε. The set of occurrences of a term, denoted $\mathcal{O}(t)$, is defined as follows:

1. If t is a variable or constant, then $\mathcal{O}(t) = \{\varepsilon\}$.
2. If $t = f(t_1,...,t_n)$, then $\mathcal{O}(t) = \{\varepsilon\} \cup \{i.o \mid 1 \leq i \leq n \ \&\ o \in \mathcal{O}(t_i)\}$

An occurrence can be used to index into a term as follows:

1. $t/\varepsilon = t$
2. $f(t_1,...,t_n)/i.o = t_i/o$

An occurrence is said to be *proper* if it is not the empty occurrence and *strict* if the subterm at that occurrence is not a variable.

A *substitution* is a mapping from variables to terms, extended to an endomorphism (a homomorphism from a set to itself) on terms. I.e., if σ is a substitution then $\sigma f(t_1,...,t_n) = f(\sigma t_1,...,\sigma t_n)$. A substitution will be denoted by a set of variable to term mappings, $\{v_1 \leftarrow t_1, v_2 \leftarrow t_2, ...\}$, where all variables outside the set are implicitly mapped to themselves. The identity substitution, i.e., the substitution mapping every variable to itself, will be written ι. The universe of substitutions will be denoted **S**. We define the *domain*, $\mathcal{D}$, of a substitution, σ, as follows: $\mathcal{D}(\sigma) = \{v \mid \sigma v \neq v\}$. Note that this differs from the usual notion of a function's domain, since it contains only those variables that are not mapped to themselves. The *range*, $\mathfrak{R}$, of a substitution is defined as $\mathfrak{R}(\sigma) = \bigcup_{v \in \mathfrak{D}(\sigma)} \{\sigma v\}$, and the *range variables*, $\mathfrak{I}$, as $\mathfrak{I}(\sigma) = \bigcup_{t \in \mathfrak{R}(\sigma)} \mathfrak{V}(t)$. A substitution, σ, can be *restricted* to a set of variables, V, written $\sigma|_V$, by mapping all variables outside V to themselves, $\sigma|_V = \{v \leftarrow \sigma v \mid v \in V\}$.

Thus, if $\sigma = \{x \leftarrow a, y \leftarrow f(z)\}$, then $\sigma f(x) = f(a)$, $\sigma g(z,y) = g(z,f(z))$ and $\sigma|_{\{x,z\}} = \{x \leftarrow a\}$. The domain, range, and range variables of σ have the following values: $\mathfrak{D}(\sigma) = \{x,y\}$, $\mathfrak{R}(\sigma) = \{a, f(z)\}$, and $\mathfrak{I}(\sigma) = \{z\}$.

A term, t, is said to be *an instance of* a term, s, if and only if there exists a substitution, σ, such that $t = \sigma s$. When the domain of σ is restricted to the variables in $\mathfrak{V}(s)$, σ is unique and is called the *match* of t by s. Substitutions may be *composed* using functional composition, i.e., for any term, t, $(\sigma_1 \circ \sigma_2)\, t = \sigma_1(\sigma_2 t)$. In the same sense that one term may be an instance of another, a substitution, σ_1, is an instance of a substitution, σ_2, if and only if there exists a third substitution, τ, such that $\sigma_1 = \tau \circ \sigma_2$; in this case σ_2 is said to be *more general than* σ_1. We denote this partial ordering by $\sigma_1 \leq \sigma_2$.

1.3 Classical Unification

This section presents the classical unification problem and provides an example of a simple algorithm for solving the problem. This classical algorithm will serve as a framework for our generalized algorithm.

Definition. Given two terms, t and s, a substitution, σ, *unifies* t and s if and only if:

$$\sigma t = \sigma s.$$

In general, there is more than one substitution that will unify two given terms. For example, if $t = f(g(x), g(y))$ and $s = f(y,z)$ then $\sigma = \{y \leftarrow g(x), z \leftarrow g(g(x))\}$ is a unifier, as well as $\sigma' = \{y \leftarrow g(g(w)), z \leftarrow g(g(g(w))), x \leftarrow g(w)\}$, and an infinite number of other substitutions. Observe in the preceding example that σ' is instance of σ: $\sigma' = \{x \leftarrow g(x)\} \circ \sigma$. In fact, all unifiers of t and s can be written as some substitution composed with σ, so we call σ the *most general unifier*. The most general unifier of two terms is analogous to the least common multiple of two natural numbers; every multiple of two numbers is divisible by the least common multiple, just as every unifier of two terms is an instance of the most general unifier.

Definition. The *most general unifier* of two terms t and s, is a unifier, σ, such that $\forall\ \varphi$

$$\varphi t = \varphi s \Rightarrow (\exists \tau, \varphi = \tau \circ \sigma).$$

In classical unification, there is at most one most general unifier up to variable renaming. A simple recursive unification algorithm based on [Robinson 71] is given below.

```
unify = proc ($t$: term, $s$: term) returns (substitution)
    case
        is_variable($t$) and is_variable($s$) ⇒
            return($\{t \leftarrow s\}$)
        is_variable($t$) ⇒
            if $t \in \mathcal{V}(s)$
                then failure: cycle
                else return($\{t \leftarrow s\}$)
        is_variable($s$) ⇒
            if $s \in \mathcal{V}(t)$
                then failure: cycle
                else return($\{s \leftarrow t\}$)
        $t.head \neq s.head$ ⇒
            failure: clash
        $t.head = s.head$ ⇒
            $\sigma := \iota$
            for i ← 1 to arity($t.head$) do
                $\sigma :=$ unify($\sigma t/i, \sigma s/i) \circ \sigma$
                end
                return($\sigma$)
        end
    end unify
```

Figure 1-1: A Classical Unification Algorithm

The algorithm points out two cases in which it is not possible to unify the input terms. If, at any point in the unification process, it is necessary to unify two terms in which the head operator symbols are not the same, a *clash* occurs and the terms are not unifiable. The second case is called a *cycle* and is succinctly shown by trying to unify the terms x and $f(x)$. The unifier, $\{x \leftarrow f(f(f(f(\ldots))))\}$ is infinite, although it can be finitely represented by a cyclic graph. In some applications, infinite unifiers are allowed or even desired [Filgueiras 82]; in many other applications infinite unifiers would lead to non-terminating program behavior. We will include the cyclicity test to disallow infinite unifiers, but note that a unification algorithm can be easily modified to allow infinite unifiers by removing the cyclicity test.

1.4 Equational Unification

As suggested at the outset, we will use the word "unification" to stand for not the single problem of section 1.3, but for a class of problems which differ from classical unification according to the desired notion of equality. In particular, we will be using equality defined by an equational theory, although we could also imagine using non-equational or even higher order logics. The equational unification problems use a form of equality that is weaker than syntactic equality and are therefore relevant to applications in which a less rigid matching process is needed.

1.4.1 Applications

Until recently, most applications made use only of classical unification, however, the need for equational unification is clear. A number of operators that occur frequently in practice have properties described by equational theories, and equational unification provides at least one technique for reasoning about operators with these properties. Furthermore, other common reasoning techniques, such as resolution and term rewriting, do not handle a number of useful equational properties that can be handled by incorporating equational unification.

For example, the properties of associativity and commutativity, called the AC-theory, can be described by the equational axioms:

1. $x \bullet (y \bullet z) = (x \bullet y) \bullet z$
2. $x \bullet y = y \bullet x$.

The *integer* operations of *plus* and *times* are only two of the many examples of associative and commutative functions about which we would like to be able to reason automatically. Despite the prevalence of AC operators, basic term rewriting systems and resolution systems run into difficulty handling this theory.

Another example of an interesting equational theory, less familiar to the mathêmatician, occurs in data type specifications for *sets*; the *insert* operation is "commutative" in the sense that the order of inserting elements into a set is not important. This property can be axiomatized by the equation: *insert*(*insert*(s, e_1), e_2) = *insert*(*insert*(s, e_2), e_1), which is, again, a problematical axiom for term rewriting and resolution based systems.

The difficulty with both the AC-theory and the *insert* operation comes from the symmetry of the axioms, which allows them to be used repetitively. A simple example using the commutative axioms in a Prolog program will exemplify the problem. Consider the Prolog program in Figure 1-2, which contains the user-defined *sibling* relation. The first three lines are input by the programmer: line one asserts that *joe* and *mary* are *siblings*. line two asserts that the *sibling* relation is commutative, and line three is a query, asking for the *siblings* of *mary*. In response to the query, the Prolog interpreter returns the value *joe* for X, since *joe* was asserted to be a *sibling* of *mary*. When the interpreter is prompted for any other *siblings* of *mary*, the interpreter again returns *joe*. The program will continue to loop in this manner, and in general there is no way to determine that all distinct answers have been found so that the process can stop. For similar reasons, the associative and commutative axioms in a term rewriting system cannot be oriented into rewrite rules without losing the termination property of the system.

```
sibling(mary, joe).
sibling(X, Y) :- sibling(Y, X).
?-sibling(mary, X).
        X = joe;
        X = joe;
        X = joe;
            .
            .
            .
```

Figure 1-2: A Prolog Program with Commutativity

In both resolution and term rewriting, a solution to the problem is to build the symmetric axioms into the system, i.e., into the unification process, so that the axioms are not explicitly needed. [Plotkin 72] describes the extension of resolution to resolution with equational unification and [Peterson 81, Dershowitz 83b. Jouannaud 84] describe extensions of term rewriting systems to *equational term rewriting systems*. Resolution and term rewriting systems are two of the current uses of equational unification, but applications are by no means limited to these two. A review of some ideas for using equational unification is given in [Siekmann 84].

1.4.2 Equational Theories

The equational unification problem will be defined in Section 1.4.3. For that purpose, we will need an understanding of equational theories and some related definitions. This section gives the necessary formal background. There are two approaches to presenting equational theories: proof theoretic and semantic. The proof theoretic approach, based on syntactic inference rules, is presented first and the semantic, or algebraic approach, is presented second. The key result in the study in equational logic is the work of Birkhoff, who proved that the two characterizations of equational theories are equivalent [Birkhoff 35, Grätzer 78].

An equation is a pair of terms, $t = s$. A *congruence* relation is an equivalence relation, $\sim$, closed under the *equality rule*:

$$t_i \sim s_i,\ 1 \leq i \leq n \Rightarrow f(t_1,\ldots,t_n) \sim f(s_1,\ldots,s_n) \text{ for all } f \in \mathrm{F} \text{ of arity n.}$$

Given a set of equations E, the *equational theory presented by* E is the set of equations E* formed by the finest congruence over **T** that contains E and is closed over instantiation. We will denote this congruence relation by $t \underset{\mathrm{E}}{=} s$, meaning $t \underset{\mathrm{E}}{=} s \in \mathrm{E}^*$. E* is exactly the set of equations derivable from E by a finite proof, using the following inference rules given by Birkhoff:

1. *Reflexity*: $x \underset{\mathrm{E}}{=} x$ is alway an axiom.

2. *Symmetry*: From $t \underset{\mathrm{E}}{=} s$ deduce $s \underset{\mathrm{E}}{=} t$.

3. *Transitivity*: From $t \underset{\mathrm{E}}{=} r$ and $r \underset{\mathrm{E}}{=} s$ deduce $t \underset{\mathrm{E}}{=} s$.

4. *Equality*: From $t_i \underset{\mathrm{E}}{=} s_i$, $1 \leq i \leq n$ deduce $f(t_1,\ldots,t_n) \underset{\mathrm{E}}{=} f(s_1,\ldots,s_n)$, f of arity n.

5. *Instantiation*: From $t \underset{\mathrm{E}}{=} s$ deduce $\sigma t \underset{\mathrm{E}}{=} \sigma s$.

We will consistently use E and E*, respectively, as a set of axioms and the equational theory presented by those axioms. Note that E* is uniquely defined from E, but there may be more than one presentation, E, given a theory E*. Even if E is an irredundant presentation it may not be unique; for example, group theory has a number of distinct irredundant presentations. This fact will, in some cases, force us to fix the presentations of theories so that proof theoretical arguments can be based on a well-defined set of axioms.

The following discussion of algebras is needed for giving the semantics of equational theories. An *algebra*, $\mathcal{A}$, is a pair $(\tilde{\mathrm{A}}, \tilde{\mathrm{F}})$, where $\tilde{\mathrm{A}}$ is a set of elements called the *carrier* of $\mathcal{A}$ and each $f \in \tilde{\mathrm{F}}$ is a function from $\tilde{\mathrm{A}}^{\mathrm{n}}$ to $\tilde{\mathrm{A}}$ for some arity, n. A mapping, ν, from **V** to $\mathcal{A}$, (i.e., to the carrier of $\mathcal{A}$) extended as a homomorphism from **T** to $\mathcal{A}$, is called an $\mathcal{A}$-assignment. It is important in this discussion to distinguish between the semantic and syntactic objects. For example, if $\tilde{\mathrm{A}}$ is the set of integers and $\tilde{\mathrm{F}}$

a set of integer operations, then addition is a function in $\tilde{F}$ and the number "one" is an element of $\tilde{A}$, whereas " + " and "1" are syntactic objects for which we may choose any interpretation.

One of the simplest models that exists for any equational theory is the *term algebra*, $\mathcal{T}$. The term algebra has as its carrier the set of terms, $T = T(F, V)$, and as its set of functions a set of term constructors, $\tilde{F}$, one for each function symbol in F. For example, if f is a unary operator in F, then there exists a corresponding unary function, $\tilde{f}$ in $\tilde{F}$, such that $\tilde{f}$ maps any term, t, in T to the term $f(t)$. Because the carrier of $\mathcal{T}$ is exactly the set of terms, the identity map is one example of a $\mathcal{T}$-assignment.

If $\nu t = \nu s$ for all $\mathcal{A}$-assignments, ν, then $\mathcal{A}$ is called a *model* of the equation $t = s$, and $t = s$ is said to be *valid* in $\mathcal{A}$; we denote this condition by $\mathcal{A} \models t = s$. Validity can be extended to a set of equations by: $\mathcal{A} \models E$ if and only if $(t = s) \in E \Rightarrow \mathcal{A} \models (t = s)$. Given a set of equations, E, we denote the class of all models of E by $\mathcal{M}o(E)$ and the set of equations valid in a class of models, $\tilde{M}$, by $Eq\ \tilde{M}$. Given a set of axioms, E, we can semantically define an equational theory as the set of equations valid in all models of E. To reiterate the equivalence between the algebraic and proof theoretic characterizations of equational theories, note that the soundness and completeness of the above inference rules, as proved by Birkhoff, can be written $E^* = Eq\ \mathcal{M}o(E)$.

If there exists a non-trivial model of E, i.e., $\mathcal{A} = (\tilde{A}, \tilde{F})$, and $\mathcal{A} \models E$ and $|\tilde{A}| > 1$, then the theory presented by E is said to be *strictly consistent*. Syntactically, an equational theory has only the trivial model if and only if $x \underset{E}{=} y$, since any equation is a substitution instance of this one. The unification problem in an inconsistent theory is always trivial. By assumption, we will work with only strictly consistent equational theories.

The equivalence relation on terms defined by an equational theory can be extended to an equivalence relation on substitutions: $\sigma_1 \underset{E}{=} \sigma_2$ if and only if $\forall\ \nu \in V\ (\sigma_1 \nu \underset{E}{=} \sigma_2 \nu)$. In many cases we are interested only in the effect of a substitution on a particular set of variables, V. We extend our definition as follows: $\sigma_1 \overset{V}{\underset{E}{=}} \sigma_2$ if and only if $\forall\ \nu \in V\ \sigma_1 \nu \underset{E}{=} \sigma_2 \nu$. Furthermore, we say that σ_1 is *more general than* σ_2 *modulo* E *over* V, written $\sigma_1 \overset{V}{\underset{E}{\leq}} \sigma_2$, if and only if:

$$\text{There exists } \tau \text{ such that } \tau \circ \sigma_1 \overset{V}{\underset{E}{=}} \sigma_2.$$

The equivalence relation defined by $\sigma_1 \overset{V}{\underset{E}{\leq}} \sigma_2$ and $\sigma_2 \overset{V}{\underset{E}{\leq}} \sigma_1$ will be denoted $\sigma_1 \overset{V}{\underset{E}{\equiv}} \sigma_2$ or, if V = V, by $\sigma_1 \underset{E}{\equiv} \sigma_2$. The relation $\underset{E}{\equiv}$ corresponds to our intuitive notion of *equivalent modulo E up to variable renaming*, and $\overset{V}{\underset{E}{\equiv}}$ corresponds to same relation where only the domain of V is considered.

The following properties hold for the equivalence relation on substitutions. They will be used freely in the proof of correctness in Chapter 3 and are presented here to avoid distraction during those proofs and to aid the reader in developing intuition about $\underset{E}{=}$ on substitutions.

1. $\sigma_1 \overset{V}{\underset{E}{=}} \sigma_2 \Rightarrow \sigma_3 \circ \sigma_1 \overset{V}{\underset{E}{=}} \sigma_3 \circ \sigma_2$, for any σ_3 *instantiation*

2. $(\sigma_1 \circ \sigma_2) \circ \sigma_3 \overset{V}{\underset{E}{=}} \sigma_1 \circ (\sigma_2 \circ \sigma_3)$ *associativity*

3. $\sigma_1 \overset{V}{\underset{E}{=}} \sigma_2 \Leftrightarrow \sigma|_V \underset{E}{=} \sigma|_V$ *restriction*

4. $\sigma_i \underset{E}{=} \varphi_i,\ 1 \leq i \leq n \Rightarrow \sigma_1 \circ \ldots \circ \sigma_n \underset{E}{=} \varphi_1 \circ \ldots \circ \varphi_n$ *equality for composition*

5. $\sigma_1 \circ \sigma_2 \underset{E}{=} \sigma_2 \circ \sigma_1$, if $(\mathfrak{I}(\sigma_1) \cap \mathfrak{I}(\sigma_2) = \emptyset)$ & *limited commutativity*
$(\mathfrak{I}(\sigma_2) \cap \mathfrak{I}(\sigma_1) = \emptyset)$ & $(\nu \in (\mathfrak{I}(\sigma_1) \cap \mathfrak{I}(\sigma_2)) \Rightarrow \sigma_1 \nu \underset{E}{=} \sigma_2 \nu)$

Having presented the basic concepts involved in working with equational theories, we are now ready to examine the unification problem in these theories.

1.4.3 A Problem Statement

Equational unification is the problem of solving an equation of the form $t = s$ in the quotient algebra, $\mathcal{T}/\underset{E}{=}$, whose carrier is the set of congruence classes of terms defined by E. If E* is the empty theory, then we again have the classical unification problem. The problem is distinguished from the problem of *satisfiability* in a first order theory. In unification the interpretation of symbols is fixed as the term algebra interpretation, whereas determining satisfiability of a first order statement is the problem of finding whether there exists any interpretation in which the statement is valid.

Definition. Let t and s be terms and E be a set of equations. A substitution, σ, is said to be an *E-unifier* of t and s if and only if:

$$\sigma t \underset{E}{=} \sigma s.$$

1.4.3.1 Properties of Unification Algorithms

Let U_E denote the set of all E-unifiers of terms t and s, i.e., $U_E(t, s) = \{\sigma \in S \mid \sigma t \underset{E}{=} \sigma\, s\}$ and let $V = \mathcal{V}(t) \cup \mathcal{V}(s)$. As in classical unification, U_E is infinite; we represent it by a *complete set of unifiers* from which set the U_E can exactly be generated by considering all instances of each substitution in the set. If every element of a set of unifiers is necessary for completeness, it is called a *minimal complete set of unifiers*. The following set of definitions formalize these concepts.

Definition. Let Σ be a set of unifiers of t and s and $V = \mathcal{V}(t) \cup \mathcal{V}(s)$. Σ is said to be *complete* if and only if it generates all unifiers:

$$\forall \sigma \in U_E(t, s)\ \exists\, \sigma' \in \Sigma \quad \sigma' \overset{V}{\underset{E}{\leq}} \sigma$$

Definition. Let Σ be a set of unifiers of t and s and $V = \mathcal{V}(t) \cup \mathcal{V}(s)$. Σ is *minimal* if and only if no substitution in Σ is redundant:

$$\forall \sigma,\sigma' \in \Sigma \quad \sigma \overset{V}{\underset{E}{\not\leq}} \sigma'$$

When it exists, a minimal complete set of unifiers is unique up to $\overset{V}{\underset{E}{\equiv}}$ for any E* [Fages 84]. The size of the minimal complete set is bounded for certain values of E. If $E = \emptyset$, there is always a singleton complete set for any two unifiable terms. If E contains only the associative and commutative axioms (the AC theory) then the complete set is always finite. If E contains only the associative axiom, then there are some pairs of terms for which every complete set of unifiers is infinite. If there is a finite complete set then a minimal complete set always exists and can be found by filtering out non-minimal unifiers through matching. For some infinite cases, the properties of minimality and completeness may conflict, so that no minimal and complete set exists [Fages 83a].

For completeness, it may be necessary for an E-unification algorithm to use more variables in the range of the unifiers than occur in the terms being unified. Because unification procedures are often used within a larger system containing variables of its own, it is useful to require an additional property to protect the existing variables from being used as new variables.

Definition. Let Σ be a set of unifiers of t and s, $V = \mathcal{V}(t) \cup \mathcal{V}(s)$, and let W be some set of variables to be protected, where $V \subseteq W$. Σ is *protective* if and only if:

$$\begin{gathered}\forall \sigma \in \Sigma \;\; \mathfrak{D}(\sigma) \subseteq V \;\&\; W - V \cap \mathfrak{I}(\sigma) = \emptyset \\ \& \; \mathfrak{D}(\sigma) \cap \mathfrak{I}(\sigma) = \emptyset.\end{gathered}$$

Without loss of generality, we will assume sets of unifiers are protective, both for the pragmatic reason given above and for the technical reason that it makes unifiers idempotent, (i.e., $\mathfrak{D}(\sigma) \cap \mathfrak{I}(\sigma) = \emptyset \Rightarrow \sigma \circ \sigma = \sigma$) which will be used in the proofs.

The properties on sets of substitutions are extended to properties on a unification procedure; collectively, they constitute partial correctness of a procedure.

Definition. A procedure, *E-unify* is a *partially correct* unification procedure for E* if and only if for all terms t and s and any finite set of variables $W \supseteq \mathcal{V}(t) \cup \mathcal{V}(s)$, if *E-unify* terminates with a set of substitutions Σ, then:

1. *consistency:* $\sigma \in \Sigma \Rightarrow \sigma t \underset{E}{=} \sigma s$.

2. *completeness:* Σ is complete for t and s.

3. *protection:* Σ is protective of t, s, and W.

If, in addition to being partially correct, *E-unify* returns only minimal sets of unifiers, then it is said to be a *minimal* procedure. A procedure which is partially correct and terminating is called *totally* *correct* and is referred to as an algorithm rather than a procedure. Any theory with a terminating unification algorithm has a minimal complete set of unifiers for any pair of terms. However, minimality of the algorithms is not included in the correctness criteria because it is a difficult to guarantee without the costly filtering process, and because it is not necessary in most applications of E-unification.

1.4.3.2 Classifying Solutions to the Unification Problem

Historically, there are a number of ways in which one can *solve* the unification problem for a particular theory, E*. We will classify each kind of solution because it will help to clarify the contributions of our approach and the assumptions behind it. The terminology used here is not well-defined in the literature, but we establish a convention based on common usage for referring to each kind of solution after giving its characterization.

1. For a given equational theory, E*, the simplest unification problem is called the *variable-only case.* The assumption is that there is one set of axioms, E, and terms to be unified contain only function symbols appearing in E and variables.

2. Unification in the *case with free symbols* is, again, unification with a single set of axioms, E, but the terms may contain free function symbols, i.e., unconstrained symbols of any arity, in addition to function symbols in E and variables.

3. The *multiple instance case* unification problem allows more than one instance of a set of axioms, for example, the AC theory for + and the AC theory for *. In this case terms still contain only function symbols from the axioms and variables, but a single term may contain more than one operator with the given properties, i.e., both + and *.

4. The unification problem for *combined* theories is to take sets of unrelated axioms, for example, the AC theory for + and the "commutativity of insert" theory for sets, and allow unification of terms containing function symbols from both of these theories.

It is this last problem, the problem of combining equational unification algorithms, which is studied and partially solved here. This thesis provides an algorithm for combining equational unification algorithms for a restricted class of equational theories and characterizes some of the theories for which the combined problem is not solved. This problem was suggested as an open problem in [Siekmann 84] and [Shostak 84].

1.5 Related Work

This section discusses some of the work that has be done on developing unification algorithms. Section 1.5.1 give a short survey of algorithms that have been designed to solve the unification problem in one particular theory. Section 1.5.2 describes a class of unification procedures that can be automatically generated from an axiomatization of the theory, and Section 1.5.3 looks at some work related to the general problem of designing algorithms for combinations of theories.

1.5.1 Single Theory Algorithms

A number of unification algorithms have been developed for particular equational theories, and a great deal of effort has been devoted to improving and bounding the running time of these algorithms. It is interesting to note that theoretical measures of complexity for these algorithms often do not reflect their relative running times in practice. This is probably because of the small size of terms in the average case, although little work has been done in trying to formally characterize the average case for a unification problem or in measuring the performance of algorithms based on an average case.

For the empty theory, the first algorithm was described in [Robinson 71] and is exponential in the size of the input. It has been modified by representing terms as directed acyclic graphs rather than trees [Corbin 83] to give an n^2 algorithm. The algorithm of [Paterson 78] runs in linear time and those of [Martelli 82] and [Baxter 73] run in nearly linear time. [Martelli 82], while theoretically slower than the linear algorithms, runs faster on some typical examples. Also, the modified algorithm of [Corbin 83] is fast in practice and has the additional advantage that the structure of the algorithm is simple and intuitive; one disadvantage of the [Corbin 83] approach is that it depends heavily on a data structure for terms that may or may not be appropriate within an application.

Some of the currently known complete E-unification algorithms are for commutative operators [Siekmann 79], AC operators [Stickel 81, Livesey 76] (with termination in the multiple instance case proved in [Fages 84]), signed trees [Kirchner 81], one-sided distributivity [Arnborg 85], and transitivity [Kirchner 85]. There are variations on the AC algorithm [Livesey 76, Fages 84] for AC with idempotence, and AC with a unit element, AC with both idempotence and a unit. An algorithm for the variable-only case of free abelian groups is given in [Lankford 84], and more generally for finitely presented abelian groups in [Kandri-Rody 85]. [Kandri-Rody 85] also gives unification algorithms for finitely presented boolean rings and finitely presented boolean rings of polynomials (i.e., with idempotence).

The decision problem for unification in the associative theory, also known as *string unification*, has been shown decidable [Makanin 77]. The associative theory has, in general, an infinite set of most general unifiers, so a terminating algorithm cannot exist. However, a complete procedure for enumerating unifiers of an associative operator is described in [Plotkin 72]. This procedure is given for one associative operator with unconstrained symbols of any arity. Not all equational theories have decidable unification problems. [Szabo 78] shows that unification in the associative distributive theory is undecidable and [Arnborg 85] shows that combining associativity with one-sided distributivity and a unit element gives a theory with an undecidable unification problem.

For theories in which a unification algorithm is known, the execution times of many have been disappointingly high. These observations are explained by some recent results classifying the complexity of different unification problems. The unification problem in the commutative theory is known to be NP-complete [Garey 79], and in the AC theory to be NP-hard [Kapur 85, Chandra 84]. Unification in the theory of right and left identity is NP-hard while the theory of one-sided distributivity can be done in polynomial time [Arnborg 85]. A restricted case of unification is the matching problem, in which a substitution is applied to one term to make it equal to another. [Benanav 85] shows that even this simpler problem has an NP-complete decision problem in both the AC and commutative theories.

Much of this past work in equational unification has made use of simplifying assumptions on the structure of terms, i.e., algorithms are usually developed to handle terms whose operators all belong to a single set of axioms. In most cases, the above unification algorithms were designed for the variable-only case, possibly with constants. Contrary to many claims, we show in Section 2.4 that extensions to the case with free symbols is often non-trivial.

Under the current approach, every time a new axiom is added to the theory, a new unification algorithm must be found and implemented for the new set of axioms. The work of [Fages 83b, Fages 84] takes steps toward remedying this situation by extending the unification algorithm for AC to handle terms containing a mix of theories including empty, commutative, and AC theories. However, his approach is still *ad hoc* rather than generalized. Adding another theory to his algorithm, for example the theory of left distributivity, would require modification of his algorithm. The modified algorithm would have to consider terms containing all the possible combinations of operators. In contrast, this thesis describes a method for automatically combining theories.

Figure 1-4 summarizes some previous results on developing equational unification algorithms, with careful attention paid to whether an algorithm is a solution to the variable-only case, the case with free symbols, or the multiple instance case. Algorithms that permit constants as well as variables are recorded in the variable-only case column; the constants are added in some theories to keep the decision problem from becoming trivial; they play an uninteresting role in the problem of generating a complete set of unifiers, except to eliminate those unifiers that equate two constants. Algorithms from [Hullot 80] and [Jouannaud 83] are based on narrowing, a process for unification described in the following section.

Abbreviation	Axioms	Theory
Ø	no axioms	*empty theory*
A	$x + (y + z) = (x + y) + x$	*associative*
C	$x + y = y + x$	*commutative*
I	$x + x = x$	*idempotent*
D_l	$x * (y + z) = (x * y) + (x * z)$	*left distributive*
D_r	$(x + y) * x = (x + y) * z$	*right distributive*
D	D_l and D_r	*distributive*
U	$x + 0 = 0 + x = x$	*right and left identity*
H	$h(x * y) = h(x) + h(y)$	*homomorphism*
M	$-(-x) = x$ $-(x * y) = (-y) * (-x)$	*minus*
SBT	$-(-x) = x$ $-(x + y) = (-y) + (-x)$ $(x + y) + (-y) = x$ $(-x) + (x + y) = y$	*signed binary trees*
T	$f(g(x, y), g(y, z)) = f(g(x, y), g(x, z))$	*transitivity*
C_r	$f(f(x, y), z) = f(f(x, z), y)$	*right commutativity,* e.g., insert on sets
C_l	$f(x, f(y, z)) = f(y, f(x, z))$	*left commutativity*
QG	$x * (x \backslash y) = y$ $(x / y) * y = x$ $x \backslash (x * y) = y$ $(x * y) / y = x$	*quasi group*
AG	$x * i(x) = e$ $x * e = x$ $*$ is AC	*abelian group*
FPAG	AG axioms with finite set of linear polynomials	*finitely presented abelian group*
FPBR	$+$ is $*$ of an AG, $*$ is ACI with U and a finite set of boolean polynomials	*finitely presented boolean ring*

Figure 1-3: Some Common Equational Theories

Theory	Variable-Only	Free Symbols	Multiple Instance
Ø	[Robinson 71] and others		
A	[Plotkin 72] procedure to enumerate unifiers	[Plotkin 72]	*open*
C	[Siekmann 79]	[Fages 83b]	[Fages 83b]
AC	[Stickel 81]	[Fages 83b]	[Fages 83b]
I	[Raulefs 78] [Hullot 80]	[Hullot 80]	[Hullot 80]
CI	[Raulefs 78] [Jouannaud 83]	[Jouannaud 83]	[Jouannaud 83]
AI	decidable [Szabo 78]	*open*	*open*
ACI	[Livesey 76]	[Jouannaud 83]	[Jouannaud 83]
ACU	[Livesey 76]	[Jouannaud 83]	[Jouannaud 83]
D_l (or D_r)	[Arnborg 85]	*open*	*open*
D	*open*	*open*	*open*
U	[Arnborg 85] [Hullot 80]	[Hullot 80]	[Hullot 80]
DA	undecidable [Szabo 78]		
D_lAU (or D_rAU)	undecidable [Arnborg 85]		
H	[Vogel 78]	this work [Kirchner 85]	this work [Kirchner 85]
Minus	[Kirchner 84a]	*open*	*open*
ABS	[Kirchner 81]		
T	[Kirchner 85]	this work [Kirchner 85]	this work [Kirchner 85]
TC	[Kirchner 85]	this work [Kirchner 85]	this work [Kirchner 85]
TCC	[Kirchner 85]	this work [Kirchner 85]	this work [Kirchner 85]
C_r (or C_l)	[Jeanrond 80]	this work [Kirchner 85]	this work [Kirchner 85]
QG	[Hullot 80]	[Hullot 80]	[Hullot 80]
AG	[Lankford 84]	*open*	*open*
FPAG	[Kandri-Rody 85]	*open*	*open*
BR	[Kandri-Rody 85]	*open*	*open*

Figure 1-4: Known Unification Algorithms

1.5.2 Narrowing

While most of the existing unification work in unification has required human invention of each algorithm, the unification procedures based on *narrowing* [Slagle 74] are automatically generated. For equational theories representable by a convergent term rewriting system there is method for performing unification in the theory of the rewriting system [Fay 79]. [Hullot 80] gives sufficient conditions for termination of the narrowing process, along with some improvements, and [Jouannaud 83] generalizes this work to equational term rewriting systems. [Rety 85] further improves on the efficiency of the narrowing process and detects cycles in the unifiers.

The approach described in this thesis serves a quite different purpose than the work on unification through narrowing; the narrowing procedure does not assume the existence of equational unification algorithms, but generates a procedure based on its axioms, whereas our approach might use a unification algorithm produced by narrowing as one of the basic pieces to be combined. For example, the work in narrowing has lead to unification algorithms for theories described by a convergent term rewriting system in which all right-hand sides are either a single variable or ground term. These theories include idempotence, identity, and quasi-groups. The algorithm described in this thesis applies to narrowing algorithms for theories in which right and left sides of all axioms are ground terms, e.g., $1 \star 1 = 0$. In other words, given a theory presented by ground equations, we can automatically generate a unification algorithm through narrowing; this algorithm can be combined using our approach with other unification algorithms.

The narrowing process is interesting from a theoretical standpoint, and gives quick positive answers to the question of existence of an E-unification algorithm for theories presented by ground equations or unconfined equations. For the cases in which narrowing gives a unification algorithm, it will also solve the problem with free symbols or multiple instances, but can solve the combining problem for only very limited combinations of theories. Moreover, even when the process terminates, it is too inefficient to be practical as part of a larger system such as a term rewriting completion procedure or resolution system.

1.5.3 Combining Theories

Although the problem of combining unification algorithms is a known open problem [Shostak 84, Siekmann 84], the problem of combining decision procedures for first order theories has been studied. Nelson and Oppen provide a procedure for deciding whether a formula is a theorem in combination of first order predicate calculus theories [Nelson 79]. Their algorithm uses a set of decision procedures for the theories being combined, much in the same way we will use unification algorithms for the theories being combined. [Shostak 84] improves on the algorithm of [Nelson 79] by localizing the information shared between algorithms. This yields an improvement in the algorithm's efficiency and extendibility.

The similarity between the structure of our unification algorithm and the decision procedure of [Nelson 79] is apparent when unification is considered in the Martelli and Montanari style of propagating equalities [Martelli 82]. [Kirchner 84a] gives an algorithm based on this style for unification in the *decomposable theories*, a class of theories in which a natural decomposition process occurs during unification. For example, if f is a symbol that does not appear at the head of either side of any equations in E, then the problem of E-unifying terms of the form $f(s_1,...,s_n)$ and $f(t_1,...,t_n)$, modulo E, is proved equivalent to unifying all pairs s_i, t_i, $1 \leq i \leq n$.

[Kirchner 85] has independently developed an algorithm for combining unification algorithms. He generalizes his earlier work by defining the notion of a decomposition process for theories in which the natural decomposition does not occur. For example, in the the commutative and AC theories there is a finite set of possible decompositions related to the set of possible unifiers.

The algorithm of [Kirchner 85] is based on two main phases: simplification of the unificands, and formation of substitutions. His algorithm is correct for a slightly smaller class of theories than the algorithm we will be defining. In particular, if a theory contains a ground equation in an irredundant presentation, then Kirchner's approach does not work. We will discuss the exact details of his restrictions on equational theories in Section 2.1.3.3 after our own restrictions are presented. [Kirchner 85] reports that his implementation works faster than ours on some typical examples using the AC theory. This may be because of earlier discovery of clashes using his approach, since he detects all clashes during the simplification phase before beginning the more expensive phase which involves cyclicity testing.

Chapter Two

A Generalized Unification Algorithm

This chapter presents our approach to generalized unification. The presentation is bottom-up. In Section 2.1 we state some assumptions on the equational theories, Section 2.2 presents a description of the algorithm, and Section 2.3 gives a detailed example.

2.1 A Generalized Approach

In equational unification, a unification algorithm must be discovered and implemented for each equational theory of interest, and with some notable exceptions, (see Section 1.5) this process is not automatic. As we will see, the problem of combining algorithms is also non-trivial. Our approach is to break a combined unification problem into pieces that we know how to solve with the sub-theory algorithms, and then to combine the answers for each these sub-problems to get a solution to the whole problem.

Our algorithm is recursive; a top-level procedure performs the steps in unification that are common to all equational theories and then invokes an appropriate equational unification algorithms for sub-problems particular to one theory.

2.1.1 Partitioning Equational Theories

Our first underlying assumption is that the sets of operators handled by each unification procedure are disjoint. Consider the following example: we are given a unification algorithm for AC-unification and an algorithm for unification with an idempotent constant (IK-unification). If the + operator is known to be AC and have an idempotent constant, a, (i.e., $a + a = a$), our technique will not automatically generate an algorithm for AC-IK-unification because the AC and IK axioms interact through the shared symbol +. On the other hand, if + were AC and $*$ were IK with a, and there were no other axioms in E, our approach would generate an algorithm for this theory.

We will treat each unification algorithm for an equational theory as a "black-box," invoking it with certain inputs, but never examining the operation within the box. The above problem with + being both AC and IK can be eliminated by considering only sets of axioms with disjoint operator sets. Formally, we define a partitioning on the axioms presenting E*.

Definition Let $\pi = \{E_1, E_2, \ldots, E_n\}$, where each E_i is a set of equations. π is a *partitioned presentation* of an equational theory E* if and only if:

1. $\mathcal{F}(E_i) \cap \mathcal{F}(E_j) = \emptyset, \forall i \neq j \leq n$

2. $\bigcup_{i \leq n} E_i$ is a presentation of E*, and

3. $\emptyset \in \pi$.

Each of the E_i's presents a theory, E^*_i, called a *sub-theory* of E*. The empty set of equations in (3) represents the empty equational theory, which is a sub-theory of any theory. Semantically, the union of a set of equational theories corresponds to taking the intersection of their models, $\mathcal{M}(E) = \bigcap_{i \leq n} \mathcal{M}(E_i)$.

The partition, π, naturally defines an equivalence relation on function symbols. Let $f_1 =_\pi f_2$ if and only if either:

1. There exists $E_i \in \pi$ such that $f_1 \in \mathcal{F}(E_i)$ and $f_2 \in \mathcal{F}(E_i)$.

2. Or, for all $E_i \in \pi$, $f_1 \notin \mathcal{F}(E_i)$ and $f_2 \notin \mathcal{F}(E_i)$.

The equivalence class of symbols containing f will be denoted $[f]$, e.g., if one of the sub-theories is ACZ with + as the AC operator and 0 as the identity constant, then $[+] = \{+, 0\}$. Function symbols that do not appear in any of the E_i's are called *uninterpreted* and all belong to one equivalence class.

This equivalence relation will provide a convenient way of naming unification algorithms of sub-theories. It is not quite correct to refer to an operator as being 'in' a particular sub-theory, since each sub-theory has the same fixed signature, F. We therefore fix the partitioned presentation π for E* and refer to the set of function symbols appearing in E_i as the *constrained* function symbols for E^*_i. For technical reasons, we will let the set of uninterpreted symbols be the constrained symbols for the empty theory. The unification algorithm for the theory constraining F will be denoted E_F*-unify* or $E_{[f]}$*-unify*, if $f \in F$.

An example should help clarify our definitions. Let E* be presented by the axioms in Figure 2-1 and let the signature, F, be the set $\{+, *, a, b, \bullet, f, g\}$ with appropriate arities. $\pi = \{E_1, E_2, E_3, E_4\}$ is a partitioned presentation of E. π is said to be a *minimal partition* because none of the elements of π can be divided further without losing the disjointness property on function symbols. In general we will use minimal partitions because this results in the simplest sub-theories, although minimality of partitions is not required.

There is a final technicality to clarify before beginning the discussion of our algorithm. Although

$\pi = \{E_1, E_2, E_3, E_4\}$

$$
\begin{array}{ll}
E_1: & x + y = y + x \\
 & (x + y) + z = x + (y + x) \\
E_2: & x * y = y * x \\
 & a * a = a \\
 & (x * y) * z = x * (y * z) \\
E_3: & x \bullet y = y \bullet x \\
 & (x \bullet y) \bullet z = x \bullet (y \bullet x) \\
E_4: & \emptyset
\end{array}
$$

Classes of F: $\{F_1, F_2, F_3, F_4\}$

$$
\begin{array}{l}
F_1 = \mathcal{F}(E_1) = \{+\} \\
F_2 = \mathcal{F}(E_2) = \{*, a\} \\
F_3 = \mathcal{F}(E_3) = \{\bullet\} \\
F_4 = \{b, f, g\}
\end{array}
$$

Figure 2-1: A Partitioned Presentation

we speak of an E-unification algorithm for a particular equational theory, each algorithm is really for an isomorphism class of equational theories. For example, if both + and • are AC, as in Figure 2-1, we can use the same algorithm for unifying a pair of terms containing + or a pair of terms containing •. The two equational theories, + AC and • AC, are not equal theories, but the isomorphism is so natural that we would normally consider them to be the same. A difficulty arises when the two theories are combined, i.e., when terms to be unified contain more than one operator with the same equational properties. To resolve this issue, each E-unification algorithm is parameterized over the set of names of its constrained operators. In this example, AC-unification for + and AC-unification for • are both instances of the same E-unification algorithm. For the purpose of this discussion, we will assume a different unification procedure exists for each instance of a theory, although in the implementation we do not duplicate the actual code.

2.1.2 Some Basic Functions

Our algorithm begins by transforming the input terms into simpler terms containing only operators from a subset of the axioms, a subset for which there is a known E-unification algorithm. The information lost in the transformation is saved in the form of a substitution. This substitution is combined, through E-unification of substitutions, with each sub-theory unifier of the transformed terms. Section 2.1.2.1 describes this transformation process on terms and section 2.1.2.2 describes a procedure for unifying substitutions.

2.1.2.1 Homogeneous Terms

A term, t, is called *homogeneous* with respect to a set of function symbols, F, if and only if $\mathcal{F}(t) \subseteq$ F. We define a *homogenizing function*, *Homog*, to convert an inhomogeneous term (i.e., a term containing operators that are not in F) into a homogeneous term. The basic operation of *Homog* is to replace all maximal subterms whose top function symbol is outside F with a new variable.

Definition. Let F be a set of function symbols and t be a term, then *Homog*(t, F) is defined as:

1. If t is a variable, then *Homog*(t, F) $= t$.

2. If $t = f(t_1,...,t_n)$ and $f \in$ F, then *Homog*(t, F) $= f$ (*Homog*(t_1, F),...,*Homog*(t_n, F)).

3. If $t = f(t_1,...,t_n)$ and $f \notin$ F, then *Homog*(t, F) $= v$, where v is a new variable.

As defined, *Homog*(t, F) is not a function but is unique for t up to names of the new variables. Technically, we should be more precise about the new variables that are used, for example, in case (2) we assume any new variables in *Homog*(t_i) are disjoint from both the old variables in t_j and new ones in *Homog*(t_j), for i≠j. To assure our algorithm is protective, these new variables must also be disjoint from the set of protected variables. We formalize the naming of new variables in Appendix A.

Taking F $= \{a\ ,*\}$, we have the following values for *Homog*:

$$\begin{aligned}&Homog(x*(a+y), \mathrm{F}) = x*v_1\\ &Homog(x*(a*b), \mathrm{F}) = x*(a*v_2)\\ &Homog(x+y, \mathrm{F}) = v_3.\end{aligned}$$

In general, homogenization of a term is done with respect to some equivalence class of F as defined by π, usually the equivalence class of the root symbol. We use the notation $\hat{t}$ to denote *Homog*(t,[t.*head*]).

In forming a homogeneous term, part of the structure of the original term is lost. To take a homogeneous term back to the term from which it was formed, we find a *preserving substitution*. Notice that t is an instance of $\hat{t}$ and we can therefore find the match of t for $\hat{t}$, in this case called a preserving substitution, by *Preserve*(t, $\hat{t}$). *Preserve*(t, $\hat{t}$) maps each new variable in $\hat{t}$ to the term it replaced in t. We distinguish the preserving substitutions from normal matching because the preversing substitution is unique for t within variable names in its domain.

Definition. Let t be a term and $\hat{t}$ be its homogenous form. The *preserving substitution* for t and $\hat{t}$ is a substitution γ such that:

$$\mathfrak{D}(\gamma) \subseteq \mathcal{N}(t)\ \&\ \gamma\hat{t} = t.$$

2.1.2.2 Unification of Substitutions

This section describes our method for combining sub-problem unifiers, which involves unification of substitutions.

Definition. Given a set of equations, E, a substitution, σ, is said to *E-unify* two substitutions, φ_1 and φ_2 if and only if:

$$\sigma \circ \varphi_1 \underset{E}{=} \sigma \circ \varphi_2.$$

We need an effective procedure, call it *map-unify*, for finding unifiers of two substitutions, φ_1 and φ_2. In looking for a unifier of two substitutions, as in testing for equality of substitutions, we restrict ourselves to the domain of the variables $V = \mathcal{D}(\varphi_1) \cup \mathcal{D}(\varphi_2)$. A *corresponding pair of terms* is defined to be a pair, $\langle t_1, t_2 \rangle$, where $t_1 = \varphi_1 v$ and $t_2 = \varphi_2 v$ for some $v \in V$. If we can unify each corresponding pair of terms in substitutions sequentially, accumulating the unifiers, and applying the results to remaining pairs, the end result will be a set of unifiers of the substitutions. The routine in Figure 2-2 performs the desired function. *map-unify* assumes the existence of our main algorithm, *CR-unify*, because the two procedures, *map-unify* and *CR-unify*, are mutually recursive. (For technical reasons that will be made clear in the termination proof, the variables in $\mathcal{D}(\varphi_2)$ are unified after the variables in $\mathcal{D}(\varphi_1)$.)

```
map-unify = proc ($\varphi_1$, $\varphi_2$:subst) returns($\Sigma$:subst_set)
    $V_1 := \mathcal{D}(\varphi_1) - \mathcal{D}(\varphi_2)$
    $V_2 := \mathcal{D}(\varphi_2)$
    $\Sigma_0 := \{\iota\}$
    $i := 0$
    for $j = 1$ to 2 do
        for $v$ in $V_j$ do
            $i := i + 1$
            $\Sigma_i := \{\omega_i \circ \sigma_{i-1} \mid \sigma_{i-1} \in \Sigma_{i-1}$ & $\omega_i \in CR\text{-}unify(\sigma_{i-1}\varphi_1 v, \sigma_{i-1}\varphi_2 v)\}$
        end
    return($\Sigma_i$)
    end map-unify
```

Figure 2-2: Procedure *map-unify* for Unification of Substitutions

2.1.3 Restrictions

The *CR-unify* procedure presented in this thesis is correct for only a restricted class of equational theories. Two syntactic restrictions on the axioms in π, *confinement* and *regularity*, are sufficient to show correctness.

2.1.3.1 Confined Theories

The first restriction will be to eliminate those equational theories in which two E-equal terms have head symbols that are constrained by different sub-theories. Because we assume the axiom sets are disjoint, this case can occur only if there is a equation in the theory of the form $t = s$, where either t or s is a variable and the other term is a non-variable. If this kind of equation is in one sub-theory, then there will also be instances of this equation in the entire theory, where the heads are not equivalent modulo π. Equations of this form, between a variable and a non-variable, will be referred to as *non-confining*, because they provide a means of deducing equations whose roots are not confined to the same equivalence class of F. A set of equations containing no non-confining equations is called *confined*; this terminology also applies to theories since they are closed sets of equations.

An example of an unconfined theory is the theory of idempotence. Let E be the theory presented by $\pi = \{\{x \bullet x = x\}, \varnothing\}$, and let f be a function symbol in F. The equation $f(x) \bullet f(x) \underset{E}{=} f(x)$ is in E, even though $\bullet$ and f are in different equivalence classes. The problem caused by having roots in different equivalence classes will be apparent in the description of *CR-unify*, where we begin the unification process by unifying in the sub-theory of the roots. Lemma 1 will state that this restriction can be made on the axioms, rather than the theory, and Theorem I will give the desired property on head symbols of equations. We postpone the presentation of these proofs until Chapter 3.

2.1.3.2 Regular Theories

The second restriction is on the sets of variables in equations of the theory. The problem comes from variables that occur on one side of the equation, but not on the other. An equation, $t = s$, is *regular* if and only if $\mathcal{V}(t) = \mathcal{V}(s)$. We extend this to sets of equations by: E is *regular* if and only if all equations in E are regular. As in the case of confined theories, we can restrict ourselves to reqular theories by restricting the set of axioms. Lemma 3 states the equivalence of these two properties and Theorem II gives sufficient conditions under which a variable and a term containing that variable are not unifiable.

We have eliminated some interesting theories, such as idempotence, identity and minus, however, there are still many interesting theories that are both confined and regular. The distributive theory, the homomorphism theory, and any theories presented by ground equations are confined and regular. In addition, any theory in which the right side of each equation is identical to the left side within a permutation of variables, i.e., a *permutative theory*, is confined and regular.

Recall from Section 1.4.1 that one of our reasons for studying E-unification was to avoid termination problems in systems that use equations as oriented rules. A permutative equations oriented in such a manner will always a program to loop, since they can be applied repeatedly. An unconfined equation, if oriented into a rule, will not lead to termination problems since the variable side of the equation is simpler than the non-variable side. Consequently, the equations that most often lead to termination problems in an application can be handled by our unification algorithm, whereas equations that cannot be handled by our unification algorithm will often not cause termination problems in an application.

2.1.3.3 Strict and Strongly Complete Theories

In this section we digress in order to give a careful distinction between our restriction to confined regular theories and the restrictions of [Kirchner 85]. As discussed in Section 1.5.3, Kirchner's algorithm is based on two main phases: simplification of unificands and formation of substitutions. His algorithm for simplification is correct and terminating for the confined theories (he calls them *non-potent* theories.) The simplification algorithm produces a simplified system of unificands and is *complete* in a sense defined in that work.

The second phase, formation of substitutions, involves forming substitutions from simple pairs of unificands, e.g., one variable and one non-variable. In general, discovery of all unifiers may require another phase of simplification after some substitutions have been formed, and this makes it difficult to guarantee termination. To avoid having to alternate between simplification and substitution formation, Kirchner places further restrictions on the equational theories so that a single phase of simplification followed by a single phase of substitution formation gives a complete set of unifiers.

The correctness of his unification algorithm depends on the theories being *strongly complete* and *strict* as well as confined.

Definition. A theory E* is said to be *strongly complete* if and only if for all variables, x, and terms, t, there exists Σ, a complete set of unifiers of x and t such that:

$$\forall \sigma \in \Sigma, \mathfrak{D}(\sigma) = \{x\}.$$

As an example of a theory that is not strongly complete, consider the theory presented by $a + b = a$. This theory is not strongly complete because the terms $x + y$ and x are unifiable by the substitution

$\{x \leftarrow a, y \leftarrow b\}$. but not by any substitution having only x in its domain. All theories presented by ground equations that have at least one operator of arity two or higher in the presentation, will fail the strong completeness test.

The notion of strictness of an equational theory depends on the following ordering on pairs of terms.

Definition. $\langle t, s\rangle \prec_{\mathcal{H}} \langle t', s'\rangle$ if and only if:

1. t is variable, t' is not a variable & $t \in \mathcal{V}(t')$, or
2. t is variable, s' is not a variable & $t \in \mathcal{V}(s')$, or
3. 1 or 2 is true with s for t.

Consider a set of unificands. i.e., pairs of terms to be unified, for which we want to find a single substitution that unifies all pairs in the set. If such a unifier exists, then we take the transitive closure of $\prec_{\mathcal{H}}$ on the set and determine whether or not the transitive closure is strict. (A strict partial order is one which is irreflexive and asymmetric.) A theory is strict only if the transitive closure of $\prec_{\mathcal{H}}$, denoted $\prec_{\mathcal{H}}{}^*$, is strict. More precisely:

Definition. A theory E* is *strict* if and only if for all sets of pairs of terms, P:

$$\exists\sigma \text{ such that } (\forall\langle t, s\rangle \in \mathrm{P}, \sigma t \underset{\mathrm{E}}{=} \sigma s)$$
$$\Rightarrow \prec_{\mathcal{H}}{}^* \text{ is strict on P.}$$

The simplest examples of theories that are not strict are those in which a variable is unifiable with a term containing that variable. For example, in the theory presented by $g(g(x)) = g(x)$, $g(y)$ and y are unifiable, but $\langle g(x), x\rangle$ is less than itself by $\prec_{\mathcal{H}}$, so $\prec_{\mathcal{H}}$ is not strict on $\mathrm{P} = \{\langle g(y), y\rangle\}$. Any theory with an equation in which one side is a subterm of the other will be non-strict.

Kirchner's restrictions on the equational theories are less general than ours because both algorithms require confinement and because strictness implies regularity. For evidence of the latter, we construct a pair of unifiable terms that cause a trivial cycle in the $\prec_{\mathcal{H}}{}^*$ ordering in any non-regular theory. Given any non-regular equation in E*, it will be of the form $t \underset{\mathrm{E}}{=} s$, where $\exists v$, $v \in t$ & $v \notin s$. The pair of terms v and t are unifiable in E* by the substitution $\{v \leftarrow s\}$, but $\prec_{\mathcal{H}}{}^*$ is not strict on $\{\langle v, t\rangle\}$ since $v \in t$.

Note that the restrictions of strong completeness and strictness are conditions on the infinite set of equations in the theory, whereas our restrictions are syntactic checks on any presentation of the theory. Moreover, strictness implies regularity but not the reverse, so his algorithm is slightly less general than ours. Examples of theories handled by our algorithm but not by Kirchner's include most theories presented by ground equations, e.g., $a + b = a$, and the confined regular theories containing equations with one side a subterm of the other, e.g., $g(g(x)) = g(x)$.

2.2 The Algorithm

The main algorithm, an algorithm for generalized equational unification of terms, is presented below. The basic assumptions are summarized here. E* is a strictly consistent equational theory, with a fixed partitioned presentation, $\pi = \{E_1, \ldots E_n\}$. For each E_i^*, there is a known unification algorithm, called E_i-*unify*, which returns a complete set of unifiers, given any two terms that are homogeneous in the constrained symbols of E_i^*. We also assume that E* is confined and regular, although the reason for these last two requirements will not become apparent until the proof of correctness in Chapter 3.

The *CR-unify* procedure is given in Figure 2-3. If *t* and *s* are variables or if *t* is a variable not occurring in *s*, then the terms are unifiable by the substitution $\{t \leftarrow s\}$. If *t* and *s* are both non-variables with root symbols from different equivalence classes of F, then any substitution instance of *t* and *s* will also have root symbols with this property, so *t* and *s* are not unifiable (see Theorem I).

If both *t* and *s* are non-variables with root symbols in the same equivalence class, then we form homogeneous terms and determine the preserving substitutions, *Preserve*(t, $\hat{t}$) and *Preserve*(s, $\hat{s}$). The union of these substitutions is well-defined because the domain of each contains only new variables from $\hat{t}$ and $\hat{s}$, and these two variable sets are disjoint by construction. We will refer to their union, γ, as the *combined preserving substitution.* P is found by unifying homogeneous terms in the appropriate sub-theory, and the preserving substitution is combined with each $\rho \in$ P by unification of substitutions. The set of unifiers, Σ, is returned after restricting the domain to the variables in *t* and *s*. (Note: for readability, we have extended the notation for restricting substitutions to denote restriction of a set of substitutions, i.e., $\Sigma|_V = \{\sigma|_V \mid \sigma \in \Sigma\}$.)

```
CR-unify = proc (t, s: term) returns (subst_set)
    case
        is_variable(t) and is_variable(s) $\Rightarrow$
            return($\{\{t \leftarrow s\}\}$)                % case 1
        is_variable(t) and ~is_variable(s) $\Rightarrow$
            return(CR-variable-unify(t, s))          % case 2
        is_variable(s) and ~is_variable(t) $\Rightarrow$
            return(CR-variable-unify(s, t))          % case 3
        t.head $\neq_\pi$ s.head $\Rightarrow$
            return ($\emptyset$)                           % case 4
        t.head $=_\pi$ s.head $\Rightarrow$                       % case 5
                $\gamma :=$ Preserve($t$, $\hat{t}$) $\cup$ Preserve($s$, $\hat{s}$)
                $P := E_{[t.head]}$-unify ($\hat{t}$, $\hat{s}$)
                $\Sigma := \bigcup_{\rho \in P}$ map-unify($\rho$, $\gamma$)
                return($\Sigma|_{\mathcal{V}(t) \cup \mathcal{V}(s)}$)
        end
    end CR-unify

CR-variable-unify = proc (v: variable, s: term) returns (subst_set)
    $\gamma :=$ Preserve($s$, $\hat{s}$)
    case
        $v \notin \mathcal{V}(s) \Rightarrow$                                  % case A
            return ($\{\{v \leftarrow s\}\}$)
        $v \in \mathcal{V}(s)$ & $v \notin \mathcal{I}(\gamma) \Rightarrow$                     % case B
            $P := E_{[s.head]}$-unify ($v$, $\hat{s}$)
            $\Sigma := \bigcup_{\rho \in P}$ map-unify($\rho$, $\gamma$)
            return($\Sigma|_{\{v\} \cup \mathcal{V}(s)}$)
        $v \in \mathcal{V}(s)$ & $v \in \mathcal{I}(\gamma) \Rightarrow$                     % case C
            return($\emptyset$)
        end
    end CR-variable-unify
```

Figure 2-3: The *CR-unify* Procedure for Equational Unification

2.3 An Example

This example shows unification in the equational theory, E, as presented in Figure 2-1. Let the input terms be, $t = b + (x * y)$ and $s = a + z$. Both are non-variable terms and the sub-theory of the root operator (+ in both cases), is presented by E_1. The relevant axioms for this example are E_1, the AC theory for +, E_2 the AC theory for * with idempotent constant a, and E_4, the empty theory with uninterpreted symbols b, f, and g.

Calling *CR-unify*(t, s), we find that case 5 of *CR-unify* is appropriate for two terms with roots constrained by the same sub-theory. Following this branch, we compute the homogeneous terms, $\hat{t}$ and $\hat{s}$. The set of constrained symbols for E_1 is { + }.

$$\hat{t} = v_1 + v_2 \qquad \hat{s} = v_3 + z,$$

The preserving substitutions are:

$$Preserve(t, \hat{t}) = \{v_1 \leftarrow b, v_2 \leftarrow x * y\} \qquad Preserve(s, \hat{s}) = \{v_3 \leftarrow a\}$$

and the combined preserving substitution $\gamma = Preserve(t, \hat{t}) \circ Preserve(s, \hat{s})$ is:

$$\{v_1 \leftarrow b, v_2 \leftarrow x * y, v_3 \leftarrow a\}$$

The homogeneous terms are unified in the sub-theory E_1^*, the AC theory for +. AC-unifying $\hat{t}$ and $\hat{s}$ results in a complete set of AC-unifiers. This set will contain two unifiers that are within $\underset{E}{\overset{V}{\equiv}}$ of:

$$\rho_1 = \{v_3 \leftarrow v_1, z \leftarrow v_2\}$$

$$\rho_2 = \{v_3 \leftarrow v_2, z \leftarrow v_1\}.$$

We proceed by calling *map-unify* with ρ_1 and γ. (Both ρ_1 and ρ_2 will be considered eventually and the choice of which unifier to look at first is arbitrary.)

$$map\text{-}unify(\rho_1, \gamma) = map\text{-}unify(\{v_3 \leftarrow v_1, z \leftarrow v_2\}, \{v_1 \leftarrow b, v_2 \leftarrow x * y, v_3 \leftarrow y\}$$

Recall that a substitution maps each variable outside its domain to itself. The corresponding pairs of terms are thus:

from ρ_1:	$\rho_1 z = v_2$	$\rho_1 v_1 = v_1$	$\rho_1 v_2 = v_2$	$\rho_1 v_3 = v_1$
from γ:	$\gamma z = z$	$\gamma v_1 = b$	$\gamma v_2 = x * y$	$\gamma v_3 = a$

In this example, each recursive call to *CR-unify* will yield a singleton set of unifiers, although this will not be true in general. (If, at any point, more than one unifier was returned, we would proceed with each in depth-first fashion.) We will show the inputs to each call to *CR-unify*, the resulting unifier set, and the effect of applying the unifier on the pairs of terms. The pair of terms for z is unified first, because it is the only element of V_1; the order of the rest of the calls is arbitrary.

1. *CR-unify*(z, v_2) returns $\{\{z \leftarrow v_2\}\}$

from ρ_1:	v_2	v_1	v_2	v_1
from γ:	v_2	b	$x * y$	a

2. *CR-unify*(b, v_1) returns $\{\{v_1 \leftarrow b\}\}$

from ρ_1:	v_2	b	b	b
from γ:	v_2	b	$x * y$	a

3. *CR-unify*$(b, x * y)$ fails, since b and $*$ are not in the same equivalence class.

This ends the call to *map-unify* for ρ_1.

We call *map-unify* again, this time with ρ_2 and γ.

$$map\text{-}unify(\rho_2, \gamma) = map\text{-}unify(\{v_3 \leftarrow v_2, z \leftarrow v_1\}, \{v_1 \leftarrow b, v_2 \leftarrow x * y, v_3 \leftarrow y\}$$

The corresponding pairs of terms are:

from ρ_2:	$\rho_2 z = v_1$	$\rho_2 v_1 = v_1$	$\rho_2 v_2 = v_2$	$\rho_2 v_3 = v_2$
from γ:	$\gamma z = z$	$\gamma v_1 = b$	$\gamma v_2 = x * y$	$\gamma v_3 = a$

The following sequence of calls to *CR-unify* results.

1. *CR-unify*(z, v_1) returns $\{\{z \leftarrow v_1\}\}$

from ρ_1:	v_1	v_1	v_2	v_2
from γ:	v_1	b	$x * y$	a

2. *CR-unify*(b, v_1) returns $\{\{v_1 \leftarrow b\}\}$

from ρ_1:	b	b	v_2	v_2
from γ:	b	b	$x * y$	a

3. *CR-unify*$(x * y, v_2)$ returns $\{\{v_2 \leftarrow x * y\}\}$

from ρ_1:	b	b	$x * y$	$x * y$
from γ:	b	b	$x * y$	a

4. *CR-unify*$(x * y, a)$ calls E_2-*unify* since both terms are homogeneous in $\{*, a\}$. This results in the singleton set of unifiers $\{\{x \leftarrow a, y \leftarrow a\}\}$

from ρ_1:	b	b	$a * a$	$a * a$

from γ: b b $a*a$ a

Composing the unifiers from steps 1 through 4, we get the substitution:

$$\{x\leftarrow a, y\leftarrow a, z\leftarrow b, v_1\leftarrow b, v_2\leftarrow a*a\}$$

or restricting to the variables in t and s:

$$\{x\leftarrow a, y\leftarrow a, z\leftarrow b\}.$$

We check that this is indeed a unifier of t and s by applying and testing for E-equality.

$$b + (a*a) \underset{E}{=} a + b$$

2.4 Difficulties in Exending *CR-unify*

The restrictions of confinement and regularity were carefully chosen. Two arguments will make this point clear. Most importantly, we will prove in Chapter 3 that the restrictions are sufficient, i.e., that *CR-unify* is consistent, complete, and terminating for the confined regular theories. This section is devoted to showing the necessity of our restrictions. We will show that *CR-unify* is not complete for theories that are either unconfined or non-regular. A great deal of effort was put into the design of the *CR-unify* algorithm, and a number modifications attempted for the purpose of weakening the restrictions of confinement and regularity. Although none of these modifications solve the more general problems, the incorrectness was not always obvious. We include some examples in this section that point to problems in the modified algorithms.

The modified algorithms will be within the framework of our approach, i.e., E* will have a partitioned presentation defining the sub-theories and these sub-theories will be assumed to have complete and terminating unification algorithms. Since the problem with *CR-unify* is completeness, the modifications will involve changing a failure case to a non-failure case.

The first example shows that *CR-unify* is not complete for unconfined theories because case 4 of the procedure signals failure when the head symbols of the two terms are constrained by different sub-theories.

Example 1.

$E_1: f(x) = x$ $t = f(x)$ $s = g(y)$

$E_2: g(x) = x$

Case 4 of *CR-unify* applies because $f \underset{\pi}{\neq} g$, so *CR-unify* returns the empty set. However, the substitution $\{x\leftarrow y\}$ is a unifier of t and s.

Note that if we replace case 4 of *CR-unify* with two recursive steps, one for each sub-theory, the problem in Example 1 will cause the procedure to loop.

Example 2.

$\hat{t}_1 = f(x) \qquad \hat{s}_1 = v_1$

$\gamma_1 = \{v_1 \leftarrow g(y)\}$

$\hat{t}_2 = v_2 \qquad \hat{s}_1 = g(y)$

$\gamma_2 = \{v_2 \leftarrow f(x)\}$

E_1-*unify*($f(x)$, v_1) returns the single unifier

$\rho = \{v_1 \leftarrow f(x)\}$.

Map-unify(γ_1, ρ) call *CR-unify* on :

$t' = g(y) \qquad s' = f(x)$

which is the original problem.

The procedure will similarly loop on the second recursive call that starts with E_2-unifying $\hat{t}_1$ and $\hat{t}_2$.

Example 3 is another example in which a unifier exists, but the *CR-unify* algorithm will fail to find it. In this example case C of *CR-variable-unify* is the incomplete step.

Example 3.

E_1: $f(0, x) = x \qquad t = f(x, g(y, z)) \qquad s = z$

E_2: $g(1, x) = x$

This example fails in case C of *CR-variable-unify*, because z occurs below an inhomogeneous subterm of s, although the substitution $\{x \leftarrow 0, y \leftarrow 2\}$ is a unifier of t and s.

Again, if we replace case C of *CR-variable-unify* with a recursive step as in case B of *CR-variable-unify* and case 5 of *CR-unify*, the procedure will loop.

Example 4.

$\hat{t} = f(x, v_1) \qquad \hat{s} = z$

$\gamma = \{v_1 \leftarrow g(y, z)\}$

E_1-*unify*($f(x, v_1)$, z) returns a single unifier

$\rho = \{z \leftarrow f(x, v_1)\}$.

Map-unify(γ, ρ) will call *CR-unify* on:

$t_1 = v_1 \qquad s_1 = g(y, z)$ and

$t_2 = f(x, v_1) \qquad s_2 = z$.

CR-unify(v_1, $g(y, z)$) returns a single unifier:

$\sigma_1 = \{v_1 \leftarrow g(y, z)\}$.

Applying this to the second pair of terms gives:

$\sigma_1 \hat{t}_2 = f(x, g(y, z)) \qquad \sigma\, \hat{s}_2 = z$,

which is the original problem.

The unification problem in Examples 3 and 4 are correctly solved by replacing x with a constant or by E-matching t by x as suggested by [Tiden 85]. However, Example 5 shows that these methods do not work in general.

Example 5.

E_1: $f(x, x) = x$ $\quad\quad t = h(f(x, y)) \quad\quad s = x$

E_2: $h(h(x)) = h(x)$

$\quad\quad \hat{t} = h(v_1) \quad\quad s = c_1$

where c_1 is a new constant.

The terms $h(v_1)$ and c_1 are not E_2-unifiable, so this procedure will fail to find the unifier, $\{x \leftarrow h(v_2), y \leftarrow h(v_2)\}$.

Intuitively, the problem illustrated in Examples 4 and 5 is that the evidence of z in t is lost when t is homogenized. If, instead of replacing $g(x, z)$ with a new variable, we replace it with z, then the algorithm finds the correct unifier. Unfortunately, this technique will not work in general for non-regular theories as shown in Example 6. Like Example 5, Example 6 is also a counter-example to the idea of replacing a variable to be unified with a constant. In Example 5 the theory is regular by not confined, and in Example 6 the theory is confined but not regular.

Example 6.

E_1: $f(x, 0) = 0$ $\quad\quad t = h(f(x, y)) \quad\quad s = y$

E_2: $\varnothing$ (h unconstrained)

$\quad\quad t' = h(y) \quad\quad s = y$

where t' is the homogenized form of t, but y is used in place of a new variable because it appears in $f(x, y)$ and it is the term with which t is being unified.

Unifying $h(y)$ and y in the empty theory will fail, so the E-unifier, $\{y \leftarrow 0\}$, will not be found.

Examples corresponding to 3 and 4 also exist for non-regular theories. The unification problem in Example 6 shows that case C of *CR-variable-unify* is not complete, and that modifying case C to perform recursion will make the process loop.

The examples themselves are interesting, but a general conclusion can also be drawn, namely, that the properties of completeness and termination conflict. These examples also motivate the careful proofs in Chapter 3, since some of the completeness problems would not arise in the intuitive arguments of the lemmas, but would arise in the careful inductive proofs of the theorems.

We also note that the trivial failure cases in classical unification, i.e., clashes and cycles, are no longer failure cases in many equational unification algorithms. While the relaxation of clash detection has been treated in the literature, the difficulty of detecting cycles in theories such as idempotence, where unifying $f(x, x)$ and x does not cause a cycle, has been underestimated.

Chapter Three

Proof of Total Correctness

This chapter presents the proof of total correctness for the *CR-unify* procedure. The correctness proof is divided into a consistency theorem, a completeness theorem, and a termination theorem. The consistency and completeness theorems, given in Sections 3.2 and 3.3, respectively, are proved by induction on the depth of recursion and are therefore dependent on the termination theorem in Section 3.4. The proof of termination is by induction on a noetherian ordering on pairs of input terms; it is a generalization of Fages' proof of termination for AC-unification [Fages 84]. Before presenting the three correctness results, we will begin in Section 3.1 with some definitions for the proofs and some important theorems having to do with our restriction to confined regular theories.

The correctness proofs depend on *CR-unify* being protective. For the sake of thoroughness, we present a more careful description of the naming of new variables in Appendix A, along with a discussion of protection. Throughout this chapter we will assume each new variable is one that has not occurred previously and thus, for example, the new variables used in forming $\hat{t}$ and $\hat{s}$ in *CR-unify* are disjoint from new variables generated within an invocation of a sub-theory unification algorithm. The formal discussion of protection and naming of new variables is relegated to an appendix because it is more technical than interesting.

3.1 An Overview

Throughout this chapter we will assume a given set of equations, E, defining a strictly consistent equational theory. E* is partitioned by $\pi = \{E_1, E_2,..., E_n\}$, which also defines a partition on the signature, F, of E*. For each sub-theory there exists a complete, consistent, protective, terminating unification algorithm. We make no other global assumptions on E* and will be careful to point out cases in which E* is assumed to be confined or regular.

Each of the three main theorems, consistency, completeness, and termination, is proved by induction. We could imagine combining the three proofs into a single inductive proof of total correctness. This is not necessary because the termination proof does not depend on the recursive calls returning complete sets of unifiers, but depends instead only on the termination of the recursive call and some technical properties of the returned substitutions. However, the proposed single correctness proof will provide a useful structure for demonstrating the overall structure of the three proofs.

First, at each step some progress must be made towards finding an answer so that the next unification problem to be solved is, by some measure, easier than the current one; this property, along with the fact that our measure of complexity does not decrease infinitely, implies termination. Second, each step must generates some piece of what could turn out to be a legitimate unifier, i.e., we cannot generate any incorrect pieces. This is the criterion for consistency. Finally, every possibility leading to a good answer must be considered. The set of partial answers generated at each step must be complete in the sense that every most general unifier can be formed from one of the elements of the set. This property will give us completeness. By separating these three independent properties into three theorems, we are able to focus our attention on one of the correctness properties at a time.

In proving the inductive hypotheses in the consistency and completeness proof, we need to show that the basic approach to breaking down the problem and building up the solutions is correct. Intuitively, we would like to show the diagram of Figure 3-1 is correct. Although we will not prove this diagram commutes, it will help motivate the technical lemmas in the proof of completeness, and is very close to the correctness diagram that will be proved. There are two levels of detail at which the diagram should be viewed. Consider first only the mappings, i.e., the labels on edges, and note that the right and left halves are mirror images, each showing $\sigma \circ \rho \underset{E}{=} \sigma \circ \gamma$. Now consider the labels on vertices. $\hat{t}$ and $\hat{s}$ are homogeneous terms and t and s are some possibly inhomogeneous instances of these terms, ρ is an E_i-unifier of $\hat{t}$ and $\hat{s}$, and σ is an E-unifier of t and s.

$$\begin{tikzcd}
\hat{s} \arrow[r, "\rho"] \arrow[d, "\gamma"'] & \rho \hat{s} \underset{E_i}{=} \rho \hat{t} \arrow[d, shift right=4, "\sigma"'] \arrow[d, shift left=4, "\sigma"] & \hat{t} \arrow[l, "\rho"'] \arrow[d, "\gamma"] \\
\gamma \hat{s} = s \arrow[r, "\sigma"'] & \sigma s \underset{E}{=} \sigma t & \gamma \hat{t} = t \arrow[l, "\sigma"]
\end{tikzcd}$$

Figure 3-1: Diagram Exemplifying Correctness Properties

The consistency argument says roughly that if γ is the preserving substitution for $\hat{t}$, t, and $\hat{s}$, s, then for any sub-theory unifier, ρ, of the homogeneous terms, all E-unifiers of ρ and γ are E-unifiers of t and s. The completeness argument says roughly that for any E-unifier, σ, there exists some sub-theory unifier, ρ, of the homogeneous terms, such that σ is an E-unifier of ρ and γ.

3.1.1 Definitions for Terms in E*

Because E* is partitioned, the terms in $\mathcal{T}/\underset{E}{=}$ have some interesting properties related to the partitioning. This section establishes some new concepts for describing these properties.

There is a partial order on terms defined as $t \preceq s$ if and only if t is a subterm of s. We use this to define another partial order on terms that is contained in $\preceq$ but takes into account the equivalence classes of function symbols defined by π.

If o is a proper occurrence, let *prefix*(o) be the string o minus the last number, i.e., for all $o \neq \varepsilon$, there exists i such that o = *prefix*(o).i. In the tree representation of terms, *prefix*(o) indexes the parent of the node at occurrence o.

Definition. Given a partition, $\underset{\pi}{=}$, on F, a term, t, and an occurrence, o $\in \mathcal{O}(t)$, o is said to be *significant* in t if and only if either:

1. o $= \varepsilon$, in which case t/o $= t$,
2. or, o is not strict in t, i.e., t/o is a variable,
3. or, (t/o).*head* $\underset{\pi}{\neq}$ (t/*prefix*(o)).*head*.

In other words, an occurrence in a term t, is significant if the subterm at the occurrence has a head symbol in a different partition of F than the symbol it occurs under. In addition, the empty occurrence and all variable occurrences in a term are significant.

Definition. The term s is a *significant subterm* of t, denoted $s \preceq_{\mathcal{J}} t$, if and only if:
$\exists$ o$\in\mathcal{O}(t)$ such that t/o $= s$ and o is a significant occurrence in t.

Note that s may appear at both significant and insignificant occurrences within t, but if at least one occurrence is significant, then s is significant in t. If s is proper in t as well as significant, we write $s \prec_{\mathcal{J}} t$.

Let F be partitioned in to $\{+\}$, $\{*, a\}$, $\{\bullet\}$ and $\{b, f, g\}$ as in Figure 2-1. Figure 3-2 shows a term with all of its significant subterms outlined. As noted, a is significant in its first and third occurrences, but not in its second occurrence.

A suggestion of the relevance of significant subterms comes from considering an arbitrary term, t,

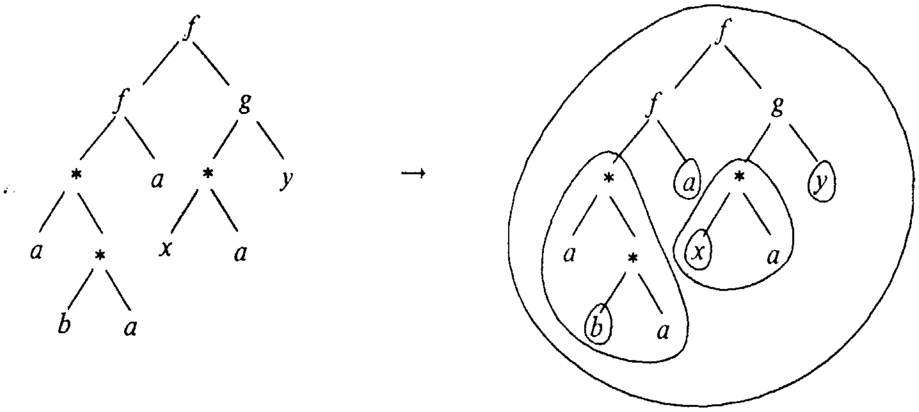

Figure 3-2: A Term and its Significant Subterms

its homogeneous form, $\hat{t}$, and the preserving substitution, $\gamma = Preserve(t, \hat{t})$. All terms in $\mathcal{R}(\gamma)$ are significant in t. The only significant subterms of the homogeneous term, $\hat{t}$, are $\hat{t}$ itself and the variables in the $\hat{t}$.

Next we define the notion of the *parents* of a term t in s, denoted $Parents(t, s)$, where a parent is an operator symbol in s having t as an argument. A special null operator, n, is included denote the parent of a term within itself.

Definition. The *parents* of t in s, written $Parents(t, s)$ are:

1. if $t = s$, then $Parents(t, s) = \{n\}$,
2. otherwise, $Parents(t, s) = \{f \mid \exists f(s_1,\ldots,s_n) \preceq s \ \& \ (\exists i, s_i = t)\}$.

This definition is extended to equivalence classes of F by considering only those classes with a representative in $Parents(t, s)$:

Definition. The *parent sets* of t in s, written $ParSets(t, s)$ are:

1. If $t = s$, then $ParSets(t,s) = \{\{n\}\}$,
2. otherwise $ParSets(t, s) = \{[f] \mid f \in Parents(t,s)\}$.

Taking t to be the term in Figure 3-2 and using the same partitioning on F ($\{+\}$, $\{*, a\}$, $\{\bullet\}$ and $\{b, f, g\}$), we get the following values of parents and parent sets in t.

$Parents(x, t) = \{*\}$ $\qquad$ $ParSets(x, t) = \{\{*, a\}\}$

$$\mathit{Parents}(a, t) = \{f, \bullet\} \qquad \mathit{ParSets}(a, t) = \{\{\bullet, a\}, \{b, f, g\}\}$$
$$\mathit{Parents}(x \bullet a, t) = \{g\} \qquad \mathit{ParSets}(x \bullet a, t) = \{\{b, f, g\}\}$$
$$\mathit{Parents}(z, t) = \{\} \qquad \mathit{ParSets}(z, t) = \{\{\}\}$$
$$\mathit{Parents}(t, t) = \{n\} \qquad \mathit{ParSets}(t, t) = \{\{n\}\}$$

3.1.2 Properties of Confined and Regular Theories

As stated in Section 2.1.3, we will limit the set of theories in E* by two syntactic restrictions: confinement and regularity. In this section we give some of the lemmas pertaining to confined and regular theories, characterizing the ways in which unification is simplified in these theories. For both confinement and regularity, we first prove that the restriction on the axioms is equivalent to the same restriction on the theory.

3.1.2.1 Confined Theories

Recall the definition of confined theories limits the equational theories to those containing no equations with a variable equal to a non-variable. Lemma 1 shows that we eliminate exactly those theories that are unconfined by eliminating those with unconfined presentations. Lemma 2 will show that confined theories contain only equations with head symbols constrained by the same E_i, and Theorem I relates this property to the unification problem.

Lemma 1 and a number of other proofs involving equational theories will be done by induction on the length of proof. By the completeness of the inference rules in Section 1.4.2, we know that if $t \underset{E}{=} s$ then there exists a finite proof of $t \underset{E}{=} s$, starting from the axioms in E and using only the five listed inference rules. We use this fact in the proof of Lemma 1, for example, where we want to show there are no equations in E* that are unconfined if there are no equations in E that are unconfined. To show any equation $t \underset{E}{=} s$ is confined, we need to show that the inference rules only prove confined equations, when they start with confined axioms. If we measure the proof as the number of steps, where each step requires application of one rule of inference, then the basis case is a proof of length zero. If the equation is proved without any inference rules, then it must be in the set of axioms.

The inductive step proves that a proof of length n + 1 gives only confined equations, assuming proofs of length n give only confined equations. Proving the induction step is done by examining each possible inference rule that could take us from the n^{th} to the $n+1^{st}$ step. If the transitive rule was the last step in a proof of $t \underset{E}{=} s$, then there must have been some other term, r, such that $t \underset{E}{=} r$ and $r \underset{E}{=} s$ were proved in n steps or less. We can therefore assume the inductive hypothesis on $t \underset{E}{=} r$ and $r \underset{E}{=} s$, which says both equations are confined. Combining this with the fact that the term r occurs in both equations, we know that either t, s, and r are all variables, or t, s, and r are all non-variables; in either case the equation $t \underset{E}{=} s$ will be confined, and the induction step for transitivity is proved.

Lemma 1: E is confined if and only if E* is confined.

Proof. One direction ($\Leftarrow$) is obvious, the other will be proved by induction on the length of proof in E*. Let E be confined and show that any equation, $t \underset{E}{=} s$, in E* must be confined.

1. Basis: $t \underset{E}{=} s$ is an axiom in E, so $t \underset{E}{=} s$ is confined because E is confined.

2. Inductive step: If $t \underset{E}{=} s$ has a proof of length n, then consider each possible inference rule for the last step in the proof:

   A. Reflexive: Trivial.

   B. Symmetric and transitive: Follow directly from the induction hypothesis.

   C. Equality: Yields only pairs of non-variable terms.

   D. Instantiation: From $t' \underset{E}{=} s'$ deduce $\sigma t' \underset{E}{=} \sigma s'$, where t is $\sigma t'$ and s is $\sigma s'$ and $t' \underset{E}{=} s'$ has a proof of length n-1 or less. By the induction hypothesis, either t' and s' are both non-variables or both variables. If they are both non-variables, t and s will be also. If they are both variables, then they are the same variable (by consistency of E*), so t and s are either both the same variable or both the same non-variable term.

□

One condition used in our proof of correctness of *CR-unify* is that E* is a confined theory. Lemma 2 shows that in confined theories there are no equations whose right and left head symbols are constrained by different sub-theories. This gives us the invariant, stated in Theorem I, that two non-variable terms whose heads symbols are constrained by different sub-theories (from π) of E* are not E-unifiable. We make use of this fact in case 4 of *CR-unify* where an empty set of unifiers is returned.

Lemma 2: If E* is a confined theory, then $t \underset{E}{=} s \Rightarrow t.head \underset{\pi}{=} s.head$.

Proof. By definition, π presents the same theory as E, although the two may contain different axioms. We will make use of this fact by performing induction on the length of proof that $t \underset{E}{=} s$ starting from the axioms in π, rather than the axioms in E.

1. Basis: If $t \underset{E}{=} s$ is an axiom in some $E_i \in \pi$, then because π defines the partition on F, $t.head \underset{\pi}{=} s.head$.

2. Inductive step: Consider each possible inference rule for the last step in the proof.

   A. Reflexive: Trivial.

   B. Symmetric and Transitive: Follow directly from the induction hypothesis.

C. Equality: Yields only terms with the same head symbol.

D. Instantiation: From $t' \underset{E}{=} s'$ deduce $\sigma t' \underset{E}{=} \sigma s'$, where $\sigma t' = t$ and $\sigma s' = s$.

i) If t' and s' are non-variable terms, then by the induction hypothesis $t'.head \underset{\pi}{=} s'.head$ and hence $t.head \underset{\pi}{=} s.head$.

ii) If either t' or s' is a variable, then they are both variables and both the same variable. Therefore t and s are the same term and the property holds trivially.

□

Theorem I: If E* is a confined equational theory, and t and s are non-variable terms such that $t.head \underset{\pi}{\neq} s.head$, then t and s are not E-unifiable.

Proof. Any instance of t and s will have the same non-equivalent head symbols, and by Lemma 2 these two terms cannot be equal in E*. □

3.1.2.2 Regular Theories

The restriction to regular theories is needed for case C of *CR-variable-unify*, a failure case. I.e., because C is a failure case, *CR-unify* is not complete for all confined theories, but regularity along with confinement of E* is sufficient for completeness. (See Section 2.4 for an example showing incompleteness of *CR-unify* for unconfined theory and a non-regular theory.) The key result of this section is Theorem II. It states that a variable is not unifiable with a term, t, containing that variable if the variable occurs below the homogeneous part of the term, i.e., at an occurrence not in $\hat{t}$. Lemma 4 is used to prove Lemma 5 which in turn is used to prove Lemma 6.

Lemma 3: E is regular if and only if E* is regular.

Proof. Again, one direction (⇐) is obvious, the other will be proved by induction on the length of proof in E*. Let E be regular and show that any equation, $t \underset{E}{=} s$, in E* must be regular.

1. Basis: $t \underset{E}{=} s$ as an axiom in E, so $t \underset{E}{=} s$ is regular by the hypothesis on E.

2. Inductive step: Consider each possible inference rule for the last step in the proof:

A. Reflexive: Trivial.

B. Symmetric and transitive: Follow directly from the induction hypothesis.

C. Equality: From $t_i \underset{E}{=} s_i$, $1 \leq i \leq n$, deduce $f(t_1,\dots,t_n) \underset{E}{=} f(s_1,\dots,s_n)$, where t is $f(t_1,\dots,t_n)$ and s is $f(s_1,\dots,s_n)$. Since t contains the union of all variables in the t_i's and s contains those in the s_i's, these two sets are equal because they are each the union of n pairwise equal sets.

D. Instantiation: From $t' \underset{E}{=} s'$ deduce $\sigma t' \underset{E}{=} \sigma s'$, where t is $\sigma t'$ and s is $\sigma s'$. By the induction hypothesis, $\mathcal{V}(t') = \mathcal{V}(s')$, so consider each v in this set. If v is in the domain of σ, then the variables in σv will occur in both t and s and otherwise v itself will occur in both t and s.

□

The property gained by restricting E* to a regular theory can be seen by considering the set of all significant subterms of a pair of equivalent terms. Lemmas 4 and 5 characterize this property precisely: Lemma 4 states that any variable occurring in two equivalent terms, will occur under the same set of parent sets, i.e., equivalence classes of parents; Lemma 5 shows that given two congruent terms, the sets of all significant subterms of the terms are equal modulo E.

Lemma 4: If E* is confined and regular, and $t \underset{E}{=} s$, then for all $v \in V$, $ParSets(v, t) = ParSets(v, s)$.

Proof. By induction on the length of proof of $t \underset{E}{=} s$ starting from the axioms in π.

1. Basis: If $t \underset{E}{=} s$ is an axiom in some $E_i \in \pi$, then t and s are homogeneous in the constrained symbols of E_i^*, and there are three cases:

   A. If $v \notin \mathcal{V}(t)$ then by regularity of E, $v \notin \mathcal{V}(s)$, so $ParSets(v, t) = ParSets(v, s) = \emptyset$.

   B. $v \in \mathcal{V}(t)$ and t is a non-variable term, then s is a non-variable term and by regularity of E*, $v \in \mathcal{V}(s)$. Therefore, $ParSets(v, t) = ParSets(v, s)$, since equations in π are by definition homogeneous.

   C. If t is a variable and $v = t$, then s is the same variable. Therefore, $ParSets(v, t) = ParSets(v, s) = \{\{n\}\}$.

2. Inductive step: Consider each possible inference rule for the last step in the proof: use one of the following inference rules.

   A. Reflexive, symmetric, and transitive: Obvious.

   B. Equality: From $t_i \underset{E}{=} s_i$, $1 \le i \le n$, deduce $f(t_1,\ldots,t_n) \underset{E}{=} f(s_1,\ldots,s_n)$, where t is $f(t_1,\ldots,t_n)$ and s is $f(s_1,\ldots,s_n)$. If t_i and s_i are non-variables, then by the induction hypothesis, all variables in t_i will occur under the same parent sets in s_i and *vice versa*. If t_i and s_i are variables, then they are the same variable so both will occur under the equivalence class, $[f]$.

   C. Instantiation: From $t' \underset{E}{=} s'$ deduce $\sigma t' \underset{E}{=} \sigma s'$, where $\sigma t' = t$ and $\sigma s' = s$. Consider the following two cases on the occurrence of v in $\sigma t'$.

      i) If the occurrence of v in $\sigma t'$ is also in t', then by the induction hypothesis there is some occurrence in s' under the same parent set. The same is true with t' and s' reversed.

ii) If the occurrence of v is in $\sigma t'$ and not in t', then v must be an element of $\mathcal{I}(\sigma)$ or, restated, there exists $v' \in \mathfrak{I}(\sigma)$ such that $v \in \mathcal{V}(\sigma v')$ and $v' \in t'$. By the induction hypothesis, *ParSets*(v', t') = *ParSets*(v', s'), and if $\sigma v'$ is simply the variable, v, then all occurrences of v will correspond to an occurrence of v' in both t and s. If $\sigma v'$ is a non-variable term containing v, then by regularity, the entire subterm $(\sigma v')$ will occur in both t and s and thus v will occur under the same operators in both.

□

Lemma 5 states that if $t \underset{E}{=} s$, then there is a one-to-one correspondence modulo E between the significant subterms of t and the significant subterms of s. The main purpose of Lemma 5 is to prove Lemma 6, although the result is interesting in its own right as a property of confined regular theories.

Lemma 5: If E* is confined and regular, and $t \underset{E}{=} s$, then for all $t' \preceq_{\mathcal{S}} t$ there exists $s' \preceq_{\mathcal{S}} s$ such that $t' \underset{E}{=} s'$.

Proof. By induction on the length of proof that $t \underset{E}{=} s$, starting from the axioms in π.

1. Basis: If $t \underset{E}{=} s$ is in some $E_i \in \pi$, then t and s are homogeneous with respect to the symbols constrained by E_i^*. Therefore, the only significant subterms in t are t itself and the variables in $\mathcal{V}(t)$. If $t' = t$, we can take s' to be s, and if t' is a variable in t, we can take s' to be the same variable, which must an element of $\mathcal{V}(s)$ by regularity of E*.

2. Inductive step: Consider each possible inference rule for the last step in the proof.

   A. Reflexive, symmetric, and transitive: Obvious.

   B. Equality: From $t_i \underset{E}{=} s_i$, $1 \le i \le n$, deduce $f(t_1,\ldots,t_n) \underset{E}{=} f(s_1,\ldots,s_n)$, where t is $f(t_1,\ldots,t_n)$ and s is $f(s_1,\ldots,s_n)$. By the induction hypothesis, for every $t_i' \preceq_{\mathcal{S}} t_i$ there exists $s_i' \preceq_{\mathcal{S}} s_i$ such that $t_i' \underset{E}{=} s_i'$. Furthermore, by Lemma 2, t_i.*head* $\underset{\pi}{=}$ s_i.*head*, so t_i will be significant in t if and only if s_i is significant in s.

   C. Instantiation: From $t_1 \underset{E}{=} s_1$ deduce $\sigma t_1 \underset{E}{=} \sigma s_1$, where t is σt_1 and s is σs_1. Consider $t' \preceq_{\mathcal{S}} t_1$ such that $\sigma t_1/o = t'$.

      i) If o is a strict occurrence of $\mathcal{O}(t_1)$, then by the induction hypothesis on n, there exists $o' \in \mathcal{O}(s_1)$ such that $t_1/o \underset{E}{=} s_1/o'$. Applying σ to these two equivalent subterms we get $\sigma t_1/o \underset{E}{=} \sigma s_1/o'$, but $\sigma t_1/o$ is t' so we can take s' to be $\sigma s_1/o'$.

      ii) If o is a variable occurrence, then t_1/o is variable, call it v. By Lemma 4, *ParSets*(v, t_1) = *ParSets*(v, s_1). Therefore, t' will itself occur significantly in s if and only if it does in t.

iii) If $o \notin \mathcal{O}(t_1)$, then t' is a proper significant subterm of σv for some $v \in \mathcal{D}(\sigma)$. By regularity of E*, $v \in \mathcal{V}(t_1) \Rightarrow v \in \mathcal{V}(s_1)$, so σv will occur in $s = \sigma s_1$, and hence t' will occur significantly in s.

□

The above lemmas give some general properties about confined regular theories. More specific to our purposes is the following lemma, which gives a sufficient condition under which terms are not equal in the theory. Lemma 6 states that no term is equal modulo E to any subterm of a proper significant subterm of itself and is proved by induction on the structure of terms, i.e., using the subterm ordering. Theorem II relates the equality problem back to the the unification problem.

Lemma 6: If E* is confined and regular, and t, s, and r are terms such that $t \preceq r \prec_{\mathcal{S}} s$, then $t \underset{E}{\neq} s$.

Proof. By structural induction on s.

1. Basis: If s is a variable or constant, then $\nexists\, r \prec_{\mathcal{S}} s$, so the hypothesis is vacuously true.

2. Inductive step: If s is a non-variable, non-constant term, then there are three cases to consider with respect to t.

   A. If t is a variable, then $t \underset{E}{\neq} s$ by the confinement property of E*.

   B. If t is a non-variable term such that $t.head \underset{\pi}{\neq} s.head$ then $t \underset{E}{\neq} s$ by Lemma 2.

   C. If t is a non-variable term such that $t.head \underset{\pi}{=} s.head$, then assume $t \underset{E}{=} s$ and derive a contradiction.

   $\exists\, s'$ such that $t \preceq s' \prec_{\mathcal{S}} s$ and $s'.head \underset{\pi}{\neq} s.head$. (The existence of the significant subterm, r, between t and s implies the existence of s' with inequivalent head.)

   Since $t.head \neq s'.head$, $t \preceq s' \prec_{\mathcal{S}} s$.

   By Lemma 5, $t \underset{E}{=} s$ and $s' \preceq_{\mathcal{S}} s$ implies there is some $t' \preceq_{\mathcal{S}} t$ such that $t' \underset{E}{=} s'$.

   From Lemma 2, $t'.head \underset{\pi}{=} s'.head$.

   Therefore, $t'.head \neq t.head$, and, again, t' must be is proper in t.

   So far we have $t' \prec_{\mathcal{S}} t \prec s' \prec_{\mathcal{S}} s$.

   Apply the induction hypothesis to s', using t' for t, s' for s' and t' for r, respectively.

   By the induction hypothesis, $t' \underset{E}{\neq} s'$, but this is a contradiction, since t' was chosen such that $t' \underset{E}{=} s'$.

□

The main result for confined regular theories is in Theorem II, which shows that case C of *CR-variable-unify* should be a failure case.

Theorem II: If E* is confined and regular and r and s are non-variable terms such that $r \prec_{\mathcal{F}} s$ and $v \in \mathcal{V}(r)$, then v and s are not unifiable.

Proof. If σ is unifier of v and s, then:

$$\sigma v \underset{E}{=} \sigma s$$

and, furthermore, the following property holds:

$$\sigma v \preceq \sigma r \prec_{\mathcal{F}} t.$$

This contradicts Lemma 6, so no such σ can exist. □

Note that the existence of the significant subterm, r, in both Lemma 6 and Theorem II is necessary. Without r the Theorem would state that no variable is unifiable (in any confined regular theory) with a term that contains it. A simple counter-example to this stronger statement is the pair of terms $f(x)$ and x in the theory presented by E = $\{f(u) \underset{E}{=} u\}$; the terms are unifiable though E* is confined and regular.

The important results of this section are Theorems I and II. Theorem I will be used to show the completeness of case 4 of *CR-unify*, where the empty set of unifiers is returned for two non-variable terms with heads constrained by different sub-theories. Theorem II will be used to show the completeness of case C of *CR-variable-unify*, where the empty set of unifiers is returned for one variable and one non-variable term when the variable occurs in the non-variable term below its top homogeneous part.

3.2 Consistency

This section presents the proof of consistency for *CR-unify*. The key lemma for consistency, Lemma 7, can be explained informally as showing that any substitution which unifies a sub-theory unifier, ρ, with the preserving substitution, γ, will also be a unifier of the two terms, t and s (where the notation here is that of the *CR-unify* procedure). This lemma alone is not enough to show consistency, since the consistency of *CR-unify* depends on the consistency of *map-unify*, which in turn depends on the consistency of *CR-unify*. Therefore, we will use an induction on the depth of recursion for proving the consistency of *CR-unify* in Theorem III.

Lemma 7: Let F_i be the set of constrained symbols for $E_i \in \pi$ and let t and s be homogeneous terms in F_i. If γ, ρ, and σ are substitutions then:

$$\rho t \underset{E_i}{=} \rho s \;\&\; \sigma \circ \rho \underset{E}{=} \sigma \circ \gamma \quad \Rightarrow \quad \sigma(\gamma t) \underset{E}{=} \sigma(\gamma s).$$

Proof.

$$\begin{aligned}
\rho t \underset{E_i}{=} \rho s &\Rightarrow \rho t \underset{E}{=} \rho s, \text{ since } E_i \text{ is a sub-theory of E} \\
&\Rightarrow \sigma(\rho t) \underset{E}{=} \sigma(\rho s), \text{ by applying the substitution, } \sigma \\
&\Rightarrow (\sigma \circ \rho) t \underset{E}{=} (\sigma \circ \rho) s, \text{ from the definition of composition} \\
&\Rightarrow (\sigma \circ \gamma) t \underset{E}{=} (\sigma \circ \gamma) s, \text{ substituting } \sigma \circ \gamma \text{ for } \sigma \circ \rho \\
&\Rightarrow \sigma(\gamma t) \underset{E}{=} \sigma(\gamma s)
\end{aligned}$$

□

The consistency theorem is given below. The proof is done on a case-by-case basis, where the cases are those appearing with labels in *CR-unify* algorithm. The basis cases are those in which no recursion is done and the inductive steps are those that involve recursion. Theorem III uses Lemma 7 in proving the inductive steps.

Theorem III: If σ is a substitution produced by *CR-unify*(t, s), then σ is an E-unifier of t and s.

Proof. By induction on the level of recursion of *CR-unify*.

1. Basis: Cases 1 and 4 of the *CR-unify* procedure and cases A and C of *CR-variable-unify*, as called in cases 2 and 3 of the *CR-unify* procedure :

   A. Case 1: t and s are both variables, so $\{t \leftarrow s\}$ is a unifier.

   B. Case 4: Returns the empty set, so the theorem holds vacuously.

   C. Case A: v does not occur in s, so the single returned substitution, $\{v \leftarrow s\}$, is a unifier.

   D. Case C: Returns the empty set, so the theorem holds vacuously.

2. Inductive step: Case 5 of *CR-unify* and case B of *CR-variable-unify* are the inductive steps.

   A. Case 5: t and s are non-variable terms, $\hat{t}$ and $\hat{s}$ are their respective homogeneous forms, and γ is a preserving substitution for both t, $\hat{t}$ and s, $\hat{s}$.

   By consistency of each of the E_i-*unify* procedures, each $\rho \in P$ is a unifier of $\hat{t}$ and $\hat{s}$.

   Choose a value for ρ.

   Each $\sigma \in \Sigma$ is generated by *map-unify*(γ, ρ) for some ρ, and is therefore of the form $\omega_n \circ \ldots \circ \omega_1$, where $\omega_i \in$*CR-unify*($\omega_{i-1}\rho v_i$, $\omega_{i-1}\gamma v_i$). By the induction hypothesis, each value of ω_i is a unifier of $\omega_{i-1}\rho v_i$ and $\omega_{i-1}\gamma v_i$, so σ is an E-unifier of each corresponding pair of terms in the range of ρ and γ and is therefore a unifier of ρ and γ themselves.

γ, ρ, and σ, meet the conditions of Lemma 7 with $\hat{t}$ and $\hat{s}$ for t and s in the lemma and t and s for γt and γs in the lemma, so σ is a unifier of t and s.

Therefore, $\sigma|_{\mathcal{V}(t)\cup\mathcal{V}(s)}$ is also a unifier of t and s.

B. Case B: v is a variable, s is non-variable term, $\hat{s}$ is the homogeneous form of s and γ is a preserving substitution for s, $\hat{s}$.

By the assumption of consistency of the E_i-*unify* procedures, each $\rho \in \mathrm{P}$ is a unifier of v and $\hat{s}$.

By the same argument as in case 5, each $\sigma \in \Sigma$ is an E-unifier of v and s in *CR-variable-unify*, which is exactly the pair of terms t, s or s, t of *CR-unify*.

Therefore, $\sigma|_{\{v\}\cup\mathcal{V}(s)}$ is a unifier of t and s of *CR-unify*.

□

3.3 Completeness

In this section we will prove that *CR-unify* is complete for all confined regular theories. The proof of the completeness theorem, Theorem IV, is done by induction on the depth of recursion, and its structure is similar to the proof of consistency, Theorem III. The proof of Theorem IV is given in Section 3.3.3, and uses a number of lemmas developed in Sections 3.3.1 and 3.3.2. The main lemmas are Lemmas 12 and 13 in Section 3.3.2, the proofs of which rely on the technical definitions and lemmas of Section 3.3.1. The proof of Theorem IV will also use Theorems I and II, since each gives an independent set of sufficient conditions for completeness of a failure case. Section 3.3.1 gives some technical definitions of functions that are used in the proof of completeness but are not needed in the implementation

3.3.1 A New Homogenizing Operation

Our proof of completeness uses a function, *U-Homog*, for forming homogeneous terms. The homogenizing operation in Section 2.1.2.1 is not unique for a given input, but may vary in the names of new variables. The function defined here will use a special set of variables and have an inverse mapping which is a substitution; both the variable set and the inverse substitution are universally defined for E*. In addition, we will extend *U-Homog* to a function on substitutions. The functions defined here are used solely as aids in the proof of completeness and are used in the implementation of *CR-unify*.

The special set of variables used in forming homogeneous terms will be denoted by **U**. Each variable in **U** represents an equivalence class of terms in the theory, E*. There is one for each element of quotient algebra, T(F, V-U)/$\underset{E}{=}$. We will represent each variable in **U** as $u_{[t]}$, where t is some term containing no variables in **U**, and $[t]$ represents the equivalence class of which t is a member. By definition, we know $t \underset{E}{=} s \Rightarrow u_{[t]} = u_{[s]}$, i.e., $u_{[t]}$ and $u_{[s]}$ are two denotations for the same variable. Henceforth, we will assume the existence of this set, **U**, as it is universally defined for E*, and denote **V**-**U**, the complement of **U**, by $\neg$**U**.

3.3.1.1 Homogeneity Using U

The function, *U-Homog*, is similar to *Homog* except that each maximal subterm whose head is not in the set F will be replaced with an element of **U** rather than with an arbitrary new variable. The following definition of *U-Homog* differs from the definition of *Homog* only in case 3.

Definition. Let F be a set of function symbols and t be a term containing no variables in **U**. *U-Homog*(t, F) is defined as follows:

1. If t is a variable, then *U-Homog*(t, F) = t.

2. If $t = f(t_1,...,t_n)$ and $f \in$ F, then
   U-Homog(t, F) = f(*U-Homog*(t_1, F),...,*U-Homog*(t_n, F)).

3. If $t = f(t_1,...,t_n)$, $f \notin$ F then *U-Homog*(t, F) = $u_{[t]}$

We extend the notion of homogeneity and the homogenizing function to substitutions. A substitution, σ, is homogeneous with respect to a set of function symbols, F, if and only if $\bigcup_{t \in \mathfrak{R}(\sigma)} \mathfrak{F}(t) \subseteq$ F. We define a function *U-HomogMap* on substitutions, which is analogous to *U-Homog* on terms.

Definition. Let σ be a substitution containing no variables from **U** (i.e., $[\mathfrak{D}(\sigma) \cup \mathfrak{I}(\sigma)] \cap$ **U** = $\emptyset$) and let F be a set of function symbols. Then *U-HomogMap*(σ, F) is a substitution such that $\mathfrak{D}$(*U-HomogMap*(σ, F)) $\subseteq \mathfrak{D}(\sigma) \cup$ **U** and for all $v \in \mathfrak{D}(\sigma) \cup$ **U**:

1. If $v \in \mathfrak{D}(\sigma)$, then
   U-HomogMap(σ, F)v = *U-Homog*(σv, F).

2. If $v \in$ **U**, then by the definition of **U**, $v = u_{[t]}$ for some t (where $\mathfrak{V}(t) \cap$ **U** = $\emptyset$), and
   U-HomogMap(σ, F)v = $u_{[\sigma t]}$.

U-HomogMap is well-defined because $t \underset{E}{=} s \Rightarrow \sigma t \underset{E}{=} \sigma s$, so picking an arbitrary t from $[t]$ will result in a unique equivalence class, $[\sigma t]$, and thus a unique variable, $u_{[\sigma t]}$. The domain of substitutions formed by *U-HomogMap* may be infinite unlike other substitutions we have used thus far. Extending substitutions in this manner gives no additional most general unifiers, because any term has only a finite number of variables. When the value of F is clear from context, we will use $\bar{t}$ and $\bar{\sigma}$ to denote *U-Homog*(t, F) and *U-HomogMap*(σ, F), respectively.

As noted, the definition of $\hat{t}$ and $\bar{t}$ are identical except in the names of variables used to replace subterms. Furthermore, if F is [*t.head*] as in $\hat{t}$, $\bar{t}$ is an instance of $\hat{t}$ for any t, since $\hat{t}$ uses different variables for each replaced subterm whereas $\bar{t}$ will use the same variable more than once if two replaced subterms are equal modulo E. Therefore, we can relate $\hat{t}$ to $\bar{t}$ by finding the match of $\bar{t}$ by $\hat{t}$. Furthermore, observe that this is the homogeneous form (with respect to F) of the preserving substitution for t and $\hat{t}$.

Proposition 1: If F = [*t.head*] and γ = *Preserve*(t, $\hat{t}$), then

$$\bar{\gamma}\hat{t} = \bar{t}.$$

The definition of *U-Homog* may, at this point, seem somewhat under-motivated. The following lemmas describe some useful properties of *U-Homog* that will be used in the completeness lemmas and are not valid for *Homog*. The two functions, *Homog* of the implementation and *U-Homog* of the lemmas, are related in the completeness theorem using Proposition 1. Lemma 8 shows that *U-Homog* commutes with the application of substitutions to terms, and Lemma 9 extends this property to composition of two substitutions. Lemma 10 is the key result of these lemmas; it justifies the division of the unification problem in E* into unification problems in the sub-theories by showing that the existence of a particular equation in E* implies the existence of the homogeneous form of the same equation in the sub-theories. Henceforth, we will assume that no substitution or term contains a variable from **U**, unless formed from *U-Homog* or *U-HomogMap*.

Lemma 8: For any set of function symbols, F:

$$\bar{\sigma}\,\bar{t} = \overline{\sigma t}.$$

Proof. By induction on the structure of t.

1. Basis: t is a variable, call it v.

   A. If $v \in \mathfrak{D}(\sigma)$ then

   $$\bar{\sigma}\,\bar{t} = \bar{\sigma}\,\bar{v} = \bar{\sigma} v$$

   $$= \overline{\sigma v}, \text{ by the definition of } \textit{U-HomogMap}.$$

B. If $v \notin \mathfrak{D}(\sigma)$, then:

$$\bar{\sigma}\,\bar{t} = \bar{\sigma}\,\bar{v} = \bar{\sigma} v$$

$= v$, since $v \notin \mathfrak{D}(\sigma)$ and $v \notin \mathrm{U} \Rightarrow v \notin \mathfrak{D}(\bar{\sigma})$

$= \sigma v = \overline{\sigma v}$

2. Inductive step: $t = f(t_1, \dots, t_n)$, for some $f \in \mathrm{F}$ of arity $\mathrm{n} \geq 0$.

A. If $f \in \mathrm{F}$, then:

$$\bar{\sigma}\,\bar{t} = \bar{\sigma}\,\overline{f(t_1,\dots,t_n)}$$

$= \bar{\sigma} f(\bar{t}_1,\dots,\bar{t}_n)$, because f is in F

$= f(\bar{\sigma}\bar{t}_1,\dots,\bar{\sigma}\bar{t}_n)$, by homomorphism of substitutions

$= f(\overline{\sigma t}_1,\dots,\overline{\sigma t}_n)$, by the induction hypothesis

$= \overline{f(\sigma t_1,\dots,\sigma t_n)}$, from the definition of *U-Homog*

$= \overline{\sigma f(t_1,\dots,t_n)}$, by homomorphism of substitutions

$= \overline{\sigma t}$

B. If $f \notin \mathrm{F}$, then:

$$\bar{\sigma}\,\bar{t} = \bar{\sigma}\,\overline{f(t_1,\dots t_n)}$$

$= \bar{\sigma} u_{[t]}$, because f is not in F

$= u_{[\sigma t]}$, since $u_{[t]} \in \mathrm{U}$

$= \overline{\sigma t}$, since the head symbol of σt is still f, and therefore not in F

□

Lemma 9 extends the commutativity of *U-HomogMap* with substitution application to the commutativity of *U-HomogMap* with substitution composition. The proof is straightforward.

Lemma 9: If F is any set of function symbols, then:

$$\overline{\sigma_1 \circ \sigma_2} = \bar{\sigma}_1 \circ \bar{\sigma}_2.$$

Proof. Show $\forall\, v \in \mathrm{V}\ \overline{\sigma_1 \circ \sigma_2}\, v = \bar{\sigma}_1 \circ \bar{\sigma}_2 v$. There are two cases:

1. If $v \in \mathrm{U}$, then $v = u_{[t]}$ for some term t, then:

$$\overline{\sigma_1 \circ \sigma_2}\, u_{[t]} = u_{[(\sigma_1 \circ \sigma_2)t]} = u_{[\sigma_1(\sigma_2 t)]} = \bar{\sigma}_1 u_{[\sigma_2 t]} = \bar{\sigma}_1(\bar{\sigma}_2 u_{[t]})$$

$$= \bar{\sigma}_1 \circ \bar{\sigma}_2 u_{[t]}$$

2. If $v \notin U$:

$$\overline{\sigma_1 \circ \sigma_2 v} = \overline{\sigma_1 \circ \sigma_2}\,\bar{v} = \overline{\sigma_1 \circ \sigma_2 v} = \overline{\sigma_1(\sigma_2 v)}$$
$$= \bar{\sigma}_1(\overline{\sigma_2 v}) = \bar{\sigma}_1(\bar{\sigma}_2 \bar{v}) = \bar{\sigma}_1 \circ \bar{\sigma}_2 \bar{v}$$
$$= \bar{\sigma}_1 \circ \bar{\sigma}_2 v$$

□

Lemma 10 is the key to our completeness argument. Unlike Lemmas 8 and 9, which do not assume any relationship between F and π, Lemma 10 will take F to be a set of function symbols constrained by one of the sub-theories. It states that the homogeneous forms (by *U-Homog*) of any two E-equal terms are E_i-equal in the sub-theory constraining F. It is important that F is the set of constrained symbols for the sub-theory, E_i^*, but we do not make any assumptions about head symbols of t and s belonging to F.

Lemma 10: If F is the set of constrained symbols for some sub-theory, E_i^* of E^*, and E^* is confined, then:

$$t \underset{E}{=} s \Longrightarrow \bar{t} \underset{E_i}{=} \bar{s}.$$

Proof. By induction on the length of proof of $t \underset{E}{=} s$, starting from the axioms in π.

1. Basis: If $t \underset{E}{=} s$ is an axiom in some $E_j \in \pi$, then:

   A. If $E_j = E_i$, i.e., $t \underset{E}{=} s \in E_i$, then t and s must be homogeneous in F. Therefore, $\bar{t} = t$, $\bar{s} = s$, and $\bar{t} = \bar{s} \in E_i$, so there is a proof in E_i.

   B. If $E_j \neq E_i$, i.e., $t \underset{E}{=} s \notin E_i$ then assume, since E^* is confined, that both t and s are non-variables. Furthermore, by disjointness of function symbols in elements of π we know $[\mathcal{F}(t) \cup \mathcal{F}(s)] \cap F = \varnothing$. Thus, $\bar{t} = u_{[t]}$ and $\bar{s} = u_{[s]}$ and since $t \underset{E}{=} s$, it follows from the definition of *U-Homog* that $u_{[t]}$ and $u_{[s]}$ are identical and thus $u_{[t]} \underset{E_i}{=} u_{[s]}$ by reflexivity.

2. Inductive step: Consider each possible inference rule for the last step in the proof.

   A. Reflexive, symmetric and transitive rules: Obvious.

   B. Equality: From $t_i \underset{E}{=} s_i$, $1 \leq i \leq n$, deduce $f(t_1,...,t_n) \underset{E}{=} f(s_1,...,s_n)$, where t is $f(t_1,...,t_n)$ and s is $f(s_1,...,s_n)$. By the induction hypothesis, $\bar{t}_i \underset{E_i}{=} \bar{s}_i$ for $1 \leq i \leq n$. There are two cases on f:

      i) If $f \in F$, then $\bar{t} = f(\bar{t}_1,..., \bar{t}_n)$ and $\bar{s} = f(\bar{s}_1,...,\bar{s}_n)$, so $\bar{t} \underset{E_i}{=} \bar{s}$ is implied by the equality rule.

      ii) If $f \notin F$, then $\bar{t} = u_{[t]}$ and $\bar{s} = u_{[s]}$. Since $t \underset{E}{=} s$, $u_{[t]}$ and $u_{[s]}$ are identical, and thus $\bar{t} \underset{E_i}{=} \bar{s}$ by reflexivity.

C. Instantiation: From $t' \underset{E}{=} s'$ deduce $\sigma t' \underset{E}{=} \sigma s'$, where t is $\sigma t'$ and s is σs. By the inductive hypothesis we know $\bar{t}' \underset{E_i}{=} \bar{s}'$ and applying the substitution $\bar{\sigma}$ to this equation we get $\bar{\sigma}\bar{t}' \underset{E_i}{=} \bar{\sigma}\bar{s}'$, which by by Lemma 8 implies $\overline{\sigma t'} \underset{E_i}{=} \overline{\sigma s'}$. But, $\sigma t'$ is t, and $\sigma s'$ is s, so this gives $\bar{t} \underset{E_i}{=} \bar{s}$.

□

Note that the assumption that E* is confined is necessary. Without this assumption, case B in the basis case would not hold. Consider an equation, for example, $v \underset{E}{=} t$, where $t.head \notin F$. In this case, $\bar{v} = v$ and $\bar{t} = u_{[t]}$, but v and $u_{[t]}$ are different variables and are therefore not equal in E_i^*.

While intuitively Lemma 10 is the key to the completeness argument, we must still relate the equality problem back to the unification problem and relate the *U-Homog* forms of terms in the lemma to the *Homog* forms in the algorithm.

3.3.1.2 The Inverse Substitution

For the homogenizing function *Homog*, we were able to define the notion of a preserving substitution, *Preserve*(t, $\hat{t}$), which mapped a homogeneous form of a term back to the original term. For *U-Homog*, such a substitution cannot be defined, since two subterms may be different terms, but equal modulo E, and will thus be replaced by the same variable. We define instead the *universal E-preserving* substitution, μ, which maps each variable in **U** to some element of the equivalence class of terms it represents.

Definition. Let μ be a substitution such that $\mathfrak{D}(\mu) = \mathbf{U}$ and $\forall u_{[t]} \in \mathbf{U}$, $\mu u_{[t]} = t'$, such that $t' \underset{E}{=} t$.

The choice of which term in the equivalence class to use is not important but only serves to take us from an element of **U** back to the set of terms in which we are working, terms that do not contains variables from **U**. The axiom of choice guarantees the existence of such a substitution. By construction of μ and the homomorphism of substitutions, we now have the following property:

Proposition 2: For any term, t, and any set of function symbols, F:

$$\mu(\bar{t}) \underset{E}{=} t.$$

We would like to extend this property directly to substitutions and get $\mu\bar{\sigma} \underset{E}{=} \sigma$, but because the domain of μ contains all variables in **U**, composing μ with the homogeneous form of a substitution yields a substitution with more variables in its domain than the original substitution. Therefore, the extension to substitutions is the following lemma:

Lemma 11: Let F be any set of function symbols.

$$\mu\circ\bar{\sigma} \underset{E}{=} \sigma\circ\mu.$$

Proof. Show $\forall\nu \in \mathbf{V}.\ (\sigma\circ\mu)\,\nu \underset{E}{=} (\mu\circ\bar{\sigma})\nu$. Consider the following cases on ν :

1. If $\nu \in \mathcal{D}(\sigma)$, then:

$$\begin{aligned} \mu\circ\bar{\sigma}\ \nu &= \mu(\overline{\sigma\nu}) \\ &\underset{E}{=} \sigma\nu, \text{ by Proposition 2} \\ &\underset{E}{=} \sigma\circ\mu\nu, \text{ because } \nu \notin \mathbf{U} \end{aligned}$$

2. If $\nu \in \mathbf{U}$, then it is of the form $u_{[t]}$, for some term, t, and we have:

$$\begin{aligned} \mu\circ\bar{\sigma}\ u_{[t]} &= \mu(u_{[\sigma t]}) \\ &\underset{E}{=} \sigma t \\ &\underset{E}{=} \sigma\,(\mu u_{[t]}) \\ &\underset{E}{=} \sigma\circ\mu\ u_{[t]} \end{aligned}$$

3. If $\nu \notin \mathbf{U}\cup\mathcal{D}(\sigma)$, then $\mu\circ\bar{\sigma}\ \nu = \nu = \sigma\circ\mu\ \nu$

□

The property proved in Lemma 11 is expressed as a diagram in Figure 3-3. This diagram will appear as a basic component in Figure 3-4, which illustrates the main completeness lemma, Lemma 13. Figure 3-4 will bring us very close to the preliminary correctness diagram of Figure 3-1.

$$\begin{tikzcd} {} \arrow[r, "\mu"] \arrow[d, "\bar{\sigma}"'] & {} \arrow[d, "\sigma"] \\ {} \arrow[r, "\mu"'] & {} \end{tikzcd}$$

Figure 3-3: Commuting Diagram for the Universal E-preserving Substitution

3.3.2 The Completeness Lemmas

Having built up some background lemmas to use in proofs, we are now ready to prove the main results leading to completeness. The first lemma, Lemma 12 shows that the decomposition process is complete, that every E-unifier is made up only of pieces of E_i-unifiers. The second lemma, Lemma 13, shows that the combination process of unifying substitutions is complete, that all the necessary combinations of E_i-unifier pieces are considered.

In the following lemmas, assume t and s are terms with compatible head symbols, i.e., either one is a variable and the other is not, or they are both non-variables with roots constrained by the same sub-theory. Let γ be the combined preserving substitution for t and s, i.e., $\gamma\hat{t} = t$, $\gamma\hat{s} = s$ and $\mathfrak{D}(\gamma) \subseteq \mathcal{V}(\hat{t}) \cup \mathcal{V}(\hat{s})$. E_i-*unify* will be the sub-theory unification algorithm for the sub-theory constraining the head symbols of t or s (one or both depending on whether they are both non-variables or not).

Lemma 12:

If E* is confined then:

$$\sigma t \underset{E}{=} \sigma s \Rightarrow \overline{\sigma \circ \gamma}\hat{t} \underset{E_i}{=} \overline{\sigma \circ \gamma}\hat{s}$$

Proof.

$$\begin{aligned} \sigma t \underset{E}{=} \sigma s &\Rightarrow \overline{\sigma t} \underset{E_i}{=} \overline{\sigma s}, \text{ by Lemma 10} \\ &\Rightarrow \bar{\sigma}\bar{t} \underset{E_i}{=} \bar{\sigma}\bar{s}, \text{ by Lemma 8} \\ &\Rightarrow \bar{\sigma}\bar{\gamma}\hat{t} \underset{E_i}{=} \bar{\sigma}\bar{\gamma}\hat{s}, \text{ by Proposition 1} \\ &\Rightarrow \overline{\sigma \circ \gamma}\ \hat{t} \underset{E_i}{=} \overline{\sigma \circ \gamma}\ \hat{s}, \text{ by Lemma 9} \end{aligned}$$

□

This proves the existence of an E_i-unifier of the homogenized terms for any E-unifier of the unhomogenized terms, and just as important, gives a way of constructing the E_i-unifier from the E-unifier. This is the key to showing that our approach of dividing E-unification problems into several E_i-unification problems is complete. We still need to show that the manner in which E-unifiers are constructed from the E_i-unifiers in the algorithm is complete.

Recall that in *CR-unify* the E_i-unifiers are combined through unification of substitutions with the preserving substitution. One property sufficient to show completeness would be that any E-unifier unifies the preserving substitution and the constructed E_i-unifier. However, this property does not hold in general for the following reason: some of the variables in the range of the sub-theory unifier may be elements of U, whereas none of the variables in the range of γ can be in U, and furthermore, σ does not contain any variables from U in its domain, so σ will not unify the two substitutions.

Instead of the property in Figure 3-1, we show a weaker but still sufficient result which states that σ is part of a substitution that unifies ρ and γ. The substitution μ will solve the problem of the variables in $\mathfrak{I}(\rho)$ being in U since it maps every variable in U to some term containing no variables in U.

Lemma 13 shows that the substitution $\sigma \circ \mu$ is an E-unifier of γ and the constructed ρ; and Figure 3-4 gives the pictorial representation of the proof. The proof of Lemma 13 starts from the result in Lemma 11, just as the left-hand diagram of Figure 3-4 shows two instances of Figure 11. Similarly, the right-hand diagram represents the last step in the proof of Lemma 13, namely the statement of the Lemma. Note that the right-hand diagram of Figure 3-4 is almost identical to the diagrams in Figure 3-1, our original goal for a correctness diagram; the only difference is that the σ arrows in Figure 3-1 are $\sigma \circ \mu$ arrows in Figure 3-4.

$$\begin{tikzcd}
{} \arrow[r, "\mu"] \arrow[d, "\bar{\gamma}"'] & {} \arrow[d, "\gamma"] & & & \\
{} \arrow[r, "\mu"] \arrow[d, "\bar{\sigma}"'] & {} \arrow[d, "\sigma"] \arrow[r, phantom, "\Longrightarrow"] & {} & {} \arrow[r, "\gamma"] \arrow[d, "\rho"'] & {} \arrow[d, "\sigma \circ \mu"] \\
{} \arrow[r, "\mu"] & {} & & {} \arrow[r, "\sigma \circ \mu"'] & {}
\end{tikzcd}$$

Figure 3-4: A Diagram of the Completeness Lemma

Lemma 13: If σ is an E-unifier of t and s, and $\rho = \overline{\sigma \circ \gamma}|_{\mathfrak{I}(\sigma \circ \gamma)}$, then:

$$\sigma \circ \mu \circ \rho \underset{E}{=} \sigma \circ \mu \circ \gamma.$$

Proof. Show the right-hand diagram of Figure 3-4 frollows from the left-hand diagram.

$\mu \circ (\overline{\sigma \circ \gamma}) \underset{E}{=} (\sigma \circ \gamma) \circ \mu$, from Lemma 11

$\Rightarrow \sigma \circ \mu \circ (\overline{\sigma \circ \gamma}) \underset{E}{=} \sigma \circ \sigma \circ \gamma \circ \mu$. by applying σ

$\Rightarrow \sigma \circ \mu \circ (\overline{\sigma \circ \gamma}) \underset{E}{=} \sigma \circ \gamma \circ \mu$. by idempotence

$\Rightarrow \sigma \circ \mu \circ (\overline{\sigma \circ \gamma}) \underset{E}{=}^{\neg U} \sigma \circ \gamma \circ \mu$. since $U \subseteq V$

$\Rightarrow \sigma \circ \mu \circ (\overline{\sigma \circ \gamma}) \underset{E}{=}^{\neg U} \sigma \circ \gamma$. since $\mathfrak{I}(\mu) = U$

$\Rightarrow \sigma \circ \mu \circ (\overline{\sigma \circ \gamma}) \underset{E}{=}^{\neg U} \sigma \circ \mu \circ \gamma$. since $\mathfrak{I}(\gamma) \cap U = \emptyset$ and $\mathfrak{I}(\mu) \cap \neg U = \emptyset$

$\Rightarrow \sigma \circ \mu \circ \rho \underset{E}{=}^{\neg U} \sigma \circ \mu \circ \gamma$, since $\mathfrak{I}(\overline{\sigma \circ \gamma}) \cdot \mathfrak{I}(\rho) \subseteq U$

$\Rightarrow \sigma \circ \mu \circ \rho \underset{E}{=} \sigma \circ \mu \circ \gamma$, since $(\mathfrak{I}(\gamma) \cup \mathfrak{I}(\rho)) \cap U = \emptyset$

□

3.3.3 A Proof of Completeness

This section proves the completeness theorem for *CR-unify*. The proof works in the reverse direction from the *CR-unify* algorithm, showing that any "good" answer was made from pieces that must be considered by *CR-unify*. We assume the input terms have a unifier, and by observing some properties of the unifier, show that either it or a more general unifier will be produced by *CR-unify*. The proof may also be somewhat confusing because the result seems too weak. At every step we are showing the there is some partially formed unifier that is more general than any actual unifier, but we never show that the pieces form a unifier. Recall, however, that this condition was proved separately in the proof of consistency, Theorem III.

One of the non-obvious steps in the proof is that the minimal set of E_i-unifiers is sufficiently large to find a complete set of E-unifiers. Note that this does not follow directly from the Lemma in Section 3.3.2, since they only show that the E-unifier can be constructed (through substitution unification) from *some* E_i-unifier and not that it can be constructed from a minimal E_i-unifier. This is important to the completeness argument since the E_i-unification algorithms are only assumed to return minimal complete sets of unifiers.

Theorem IV: If σ is an E-unifier of *t* and *s*, then there exists $\theta \in$ *CR-unify*(*t*, *s*) such that $\theta \overset{V}{\underset{E}{\leq}} \sigma$.

Proof. By induction on the depth of recursion in *CR-unify*. Proving the induction step will require a second induction on the number of calls to *CR-unify* made from *map-unify*.

1. The basis for induction on the depth of recursion are those cases in *CR-unify* for which *map-unify* is not called, namely, cases 1 and 4 of *CR-unify* and case A and C of *CR-variable-unify* under either 2 or 3 of *CR-unify*.

A. Case 1: t and s are both variables. Any unifier of t and s is an instance of $\{t \leftarrow s\}$.

B. Case A: v is a variable not occurring in s. Any unifier is an instance of $\{v \leftarrow s\}$.

C. Case C: v is a variable occurring in s and in $\mathfrak{I}(\gamma)$. Therefore, v is a variable in some non-variable proper significant subterm of s, and by Theorem II, v and s are not unifiable. The empty set is a complete set of unifiers.

D. Case 4: t and s are non-variable terms such that $t.head \underset{\pi}{\neq} s.head$. By the Theorem I, t and s are not unifiable. Again, the empty set is complete.

2. The inductive step includes cases 5 of *CR-unify* and case B of *CR-variable-unify* when called from either 2 or 3 of *CR-unify*.

   A. Case 5: t and s are non-variables such that $t.head \underset{\pi}{=} s.head$. Let E_i^* be the sub-theory of E^* constraining $t.head$ and $s.head$ and let F be the set of function symbols constrained by E_i^*.

   We will first show that the substitution ρ constructed in Lemma 12 bears a useful relation to some sub-theory unifier found in *CR-unify*.

   Let $\rho = \overline{\sigma \circ \gamma}\,|_{\mathfrak{D}(\sigma \circ \gamma)}$.

   From Lemma 12 we know:

$$\rho \hat{t} = \rho \hat{s}.$$

   By the completeness of E_i-*unify*, there exists $\rho' \in E_i$-*unify*$(\hat{t}, \hat{s})$ such that

$$\rho' \overset{V_1}{\underset{E_i}{\leq}} \rho, \text{ for } V_1 = \mathcal{V}(\hat{t}) \cup \mathcal{V}(\hat{s}).$$

   Therefore, $\exists \varphi$ such that $\varphi \circ \rho' \overset{V_1}{\underset{E_i}{=}} \rho$ and since E_i^* is a sub-theory of E^*,

$$\varphi \circ \rho' \overset{V_1}{\underset{E}{=}} \rho.$$

   Without loss of generality, assume

$$\mathfrak{D}(\varphi) \subseteq \mathfrak{D}\rho \cup \mathfrak{I}(\rho').$$

   By protectiveness of φ' we know

$$(\mathfrak{D}(\rho) \cup \mathfrak{I}(\rho) \cup \mathfrak{D}(\rho') \cup \mathfrak{I}(\rho')) \cap (V_1 - V_2) = \emptyset.$$

   Let $V_2 = \mathcal{V}(t) \cup \mathcal{V}(s)$ and $V_3 = V_1 \cup V_2$ and from the previous step we have

$$\varphi \circ \rho' \overset{V_3}{\underset{E}{=}} \rho.$$

   Using these values of ρ, ρ', and φ, we can show the the inductive step for case 5, i.e., that there exists $\theta \in$ *map-unify*(ρ', γ), such that $\theta \overset{V_2}{\underset{E}{\leq}} \sigma$. By examination of *map-unify* we know that θ would have to be a substitution of the form $\omega_n \circ \ldots \circ \omega_0$, where n is the size of $\mathcal{V}(\rho')\cup\mathcal{V}(\gamma)$ and ω_i is the unifier of the i^{th} pair of terms. Let σ_i be the accumulated unifier for the i^{th} iteration, i.e., $\sigma_i = \omega_i \circ \omega_{i-1} \circ \ldots \circ \omega_0$. Show by induction on i that $\sigma_n \overset{V}{\underset{E}{\leq}}{}^{2} \sigma$.

i) Basis: If $i=0$, then $\sigma_0 = \omega_0 = \iota$. Therefore, $\sigma_0 \overset{V}{\underset{E}{\leq}}{}^{2} \sigma$. (Note: $n = 0$ would correspond to both t and s being homogeneous and equal modulo E.)

ii) Inductive step: If $i>0$, then let v_i be the i^{th} variable considered in $\mathcal{V}(\rho')\cup\mathcal{V}(\gamma)$ for forming corresponding pairs of terms.

Show there exists $\omega_i \in$ *CR-unify*$(\sigma_{i-1}\rho' v_i,\ \sigma_{i-1}\gamma v_i)$ such that $\sigma_i = \omega_i \circ \sigma_{i-1} \overset{V}{\underset{E}{\leq}}{}^{2} \sigma$.

$\sigma_{i-1} \overset{V}{\underset{E}{\leq}}{}^{2} \sigma$, by the induction hypothesis on i

$\sigma_{i-1} \overset{V}{\underset{E}{\leq}}{}^{2} \sigma \circ \mu$, since $\mathcal{V}(\mu)\cap\mathcal{V}(\sigma) = \emptyset$ & $\mathcal{V}(\mu)\cap V_2 = \emptyset$

$\exists \psi$ such that $\psi \circ \sigma_{i-1} \overset{V}{\underset{E}{=}}{}^{2} \sigma \circ \mu$, by definition of $\overset{V}{\underset{E}{\leq}}{}^{2}$

$\sigma \circ \mu \circ \rho \underset{E}{=} \sigma \circ \mu \circ \gamma$, from Lemma 13

$\sigma \circ \mu \circ \rho \overset{V}{\underset{E}{=}}{}^{3} \sigma \circ \mu \circ \gamma$

$\sigma \circ \mu \circ \varphi \circ \rho' \overset{V}{\underset{E}{=}}{}^{3} \sigma \circ \mu \circ \gamma$, since $\varphi \circ \rho' \overset{V}{\underset{E}{=}}{}^{3} \rho$ was shown above

$\exists \varphi'$ such that $\varphi' \circ \sigma \circ \mu \circ \rho' \overset{V}{\underset{E}{=}}{}^{3} \sigma \circ \mu \circ \gamma$, since $\forall x \in \mathcal{V}(\varphi)$, $x \notin \mathcal{V}(\varphi')$ and:

a) If $x \in V_2$, then $x \in \mathcal{V}(\sigma)$:

$$\begin{aligned} \mu \circ \varphi \circ \rho' x &= \mu \circ \varphi x \\ &= \mu \circ \bar{\sigma} x \\ &= \sigma x. \end{aligned}$$

b) If $x \in V_1 - V_2$, then $x \in \mathcal{V}(\gamma)$:
$x \notin \mathcal{V}(\mu)$ & $x \notin \mathcal{V}(\sigma)$.

c) If $x \notin V_3 V_1 \cup V_2$, then x is a new variable from ρ':
$x \notin \mathcal{V}(\mu)$ & $x \notin \mathcal{V}(\sigma)$.

Using this new substitution, φ', we have:

$\varphi' \circ \varphi' \circ \sigma \circ \mu \circ \rho' \overset{V}{\underset{E}{=}}{}^{3} \varphi' \sigma \circ \mu \circ \gamma$,

$\varphi' \circ \sigma \circ \mu \circ \rho' \overset{V}{\underset{E}{=}}{}^{3} \varphi' \circ \sigma \circ \mu \circ \gamma$,

$\varphi' \circ \psi \circ \sigma_{i-1} \circ \rho' \overset{V}{\underset{E}{=}}{}^{3} \varphi' \circ \psi \circ \sigma_{i-1} \circ \gamma$, by replacing $\sigma \circ \mu$ with $\psi \circ \sigma_i$

Therefore, $\varphi' \psi$ is an E-unifier of $t_i = \sigma_{i-1} \rho' v_i$ and $s_i = \sigma_{i-1} \gamma v_i$.

By the induction hypothesis on t and s, there exists a substitution, $\omega_i \in$ *CR-unify*(t_i,s_i). such that:

$$\omega_i \overset{V}{\underset{E}{\leq}}{}^{4} \varphi' \circ \psi, \text{ where } V_4 = \mathfrak{V}(t_i) \cup \mathfrak{V}(s_i).$$

Since $\mathfrak{I}(\varphi')$ contains variables from V_1 only if they are in $\mathfrak{I}(\sigma)$,

$$\varphi' \circ \sigma \overset{V}{\underset{E}{=}}{}^{1} \sigma.$$

Furthermore, $\omega_i \circ \sigma_{i-1} \overset{V}{\underset{E}{\leq}}{}^{1} \varphi' \circ \psi \circ \sigma_{i-1}$, since by protectiveness of *CR-unify*, $x \in \mathfrak{I}(\omega_i) \cap \mathfrak{J}(\sigma)$ only if $x \in V_4$.

We also have $\varphi' \circ \psi \circ \sigma_{i-1} \overset{V}{\underset{E}{=}}{}^{1} \varphi' \circ \sigma \overset{V}{\underset{E}{=}}{}^{1} \sigma \overset{V}{\underset{E}{=}}{}^{1} \sigma \circ \mu$.

Therefore. $\sigma_i = \omega_i \circ \sigma_{i-1} \overset{V}{\underset{E}{\leq}}{}^{1} \sigma$. which which completes the induction proof on i.

The induction on i shows $\sigma_n \overset{V}{\underset{E}{\leq}}{}^{2} \sigma$, which proves the inductive step on t and s in case 5 and therefore proves the existence of some θ produced by *CR-unify*(t, s) such that $\theta \overset{V}{\underset{E}{\leq}}{}^{2} \sigma$.

B. Case B: The proof follows Case 5, except v replaces both t and $\hat{t}$.

□

3.4 Proof of Termination

If recursive calls from *CR-unify* were made only to subterms of the original inputs, then termination would be obvious. However, at each iteration within *map-unify*, the substitution accumulated up to a given point is applied to the next corresponding pair of terms, so the terms of a recursive call are not necessarily subterms of the original inputs, and may be larger than the inputs. The proof of termination in Section 3.4 uses noetherian induction; we define a noetherian ordering on terms which is proved to be strictly decreasing with each level of recursion. The ordering is a generalization of the ordering used by Fages to show termination of AC-unification [Fages 84]. The proof of termination is a generalization of the termination proof for AC-unification [Fages 84]. We consistently extend his definitions to handle the general case for unification in equational theories. With minor exceptions, our definitions would be identical to Fages' if we restricted ours to only the associative-commutative and empty theories. Noetherian induction is discussed briefly in Section 3.4.1. Section 3.4.2 describes the noetherian ordering that is the basis of our termination proof and then gives some lemmas on the ordering.

3.4.1 Noetherian Induction

Classical induction is based on a total ordering, typically the "less than" ordering on the natural numbers. Noetherian induction is more general in that it is based on a partial ordering; the partial ordering is additionally required to have no infinite decreasing paths. The reader is referred to [Cohn 65] for a justification of noetherian induction and to [Huet 80b] for some abstract properties of orderings.

A typical class of noetherian orderings that are not total are those formed as lexicographic extensions of two or more total orderings. In fact any lexicographic extension of noetherian orderings is itself a noetherian ordering. We will use this fact in the definition of the noetherian ordering for our termination proof.

3.4.2 A Noetherian Ordering for E-Unification

The input to *CR-unify* is a pair of terms, and we define a noetherian ordering on pairs of terms to perform the induction. The partial ordering on terms defined by the subterm property is not acceptable as a basis for our induction because, as mentioned, recursive calls are made to terms formed by applying substitutions to subterms of the inputs and not just to subterms of the original input terms. Therefore, a recursive call from *CR-unify* is not necessarily made to arguments that are strictly less than the inputs by the subterm ordering. We instead define an ordering that is contained in the subterm ordering, but does not increase when certain substitutions are applied to inputs. This ordering is shown to decrease with each level of recursion. It is a lexicographic extension of two orderings on the size of sets τ and ν, which are defined below.

Because each E_i-unification procedure is assumed to terminate, it is is appropriate for the current discussion to think of the unification of two homogeneous terms from the same sub-theory as being a single computation. Expanding on this idea, it is more difficult to unify a term with many inhomogeneous subterms than one that is close to being homogenous, even if the more homogeneous term has a larger actual size. This should help motivate the definition of the first measure of complexity based on the set of distinct terms having strict and significant occurrences in the input terms.

Definition. $\tau(t, s) = \{r \mid r \text{ is non-variable } \& \ (r \preceq_{\mathcal{S}} t \text{ or } r \preceq_{\mathcal{S}} s)\}$.

The intuition behind the second complexity measure is more difficult, but is related to τ in the following sense: If a variable occurs under more than one operator set, where by operator set we mean an equivalence class of F, then any substitution of a non-variable term for the variables will result in at least one new strict significant occurrence in the resulting term. Therefore, ν is the set of variables occurring under more than one operator set in t and s.

Definition. Let $\nu(t, s) = \{x \in \mathbf{V} \mid size(ParSets(x, t) \cup ParSets(x, s)) > 1\}$

We will measure the complexity of a particular unification problem by considering the cardinalities of ν and τ, and use this measure of complexity to define an ordering on the pairs of terms comprising the inputs. The ordering will be denoted $\prec_{\mathcal{C}}$ and is defined as the lexicographic extension of the cardinalities of ν and τ:

Definition. $\langle t', s'\rangle \prec_{\mathcal{C}} \langle t, s\rangle$ if and only if:

1. $size(\nu(t', s')) \leq size(\nu(t, s))$

2. and, $(size(\nu(t', s')) = size(\nu(t, s)) \Rightarrow size(\tau(t', s')) < size(\tau(t, s)))$.

We will use the notation $\preceq_{\mathcal{C}}$ to denote a reflexive ordering containing $\prec_{\mathcal{C}}$, i.e., $\langle t', s'\rangle \preceq_{\mathcal{C}} \langle t, s\rangle$ if and only if:

1. $\langle t', s'\rangle \prec_{\mathcal{C}} \langle t, s\rangle$

2. or, $(size(\nu(t', s')) = size(\nu(t, s))$ & $size(\tau(t', s')) = size(\tau(t, s)))$.

Note that the t and s can be commuted in the ordering, since $\tau(t, s) = \tau(s, t)$ and $\nu(t, s) = \nu(s, t)$. This is consistent with our expectations of a good measure for the complexity of unification since unification is itself commutative. Lemma 14 states that $\prec_{\mathcal{C}}$ is a noetherian ordering. This will allow us to use it as the ordering for an inductive proof.

Lemma 14: The $\prec_{\mathcal{C}}$ ordering is noetherian.

Proof. It is the lexicographic extension of two instances of the less than total ordering on the natural numbers. □

3.4.3 Some Properties of the Ordering

The inductive hypothesis within the proof of termination can only be applied to pairs of terms that are strictly smaller (in this case by the $\prec_{\mathcal{C}}$ ordering) than the given pair of terms. We will use the following lemmas to show that recursive calls are made to strictly smaller terms than the input terms. Lemmas 15 and 16 give independent conditions that are each sufficient for one pair of terms to be less than another by the $\prec_{\mathcal{C}}$ ordering. Lemma 15 shows that two non-variable proper significant subterms of two terms have strictly smaller complexity than the two terms; it will apply in case 5 of *CR-unify* where the algorithm recurses over non-variable arguments.

Lemma 15: Let t', and s' be non-variable proper significant subterms of t or s, then:

$$\langle t', s'\rangle \prec_{\mathcal{C}} \langle t, s\rangle.$$

Proof. Let $\nu_1 = \nu(t, s)$, $\tau_1 = \tau(t, s)$, $\nu_2 = \nu(t', s')$ and $\tau_2 = \tau(t', s')$. Since t' and s' are subterms of t or s, all variables occurrences in t' or s' correspond to some occurrence in t or s. Since t' and s' are non-variable terms, all variables in t' or s' will occur under an operator they occurred under in t or s, so $\nu_2 \subseteq \nu_1$ and $size(\nu_2) \leq size(\nu_1)$. It is now sufficient to show $\tau_2 \subset \tau_1$, which in this case is independent of whether or not $\nu_2 = \nu_1$.

All significant subterms of t' or s' are significant in t or s (because t' and s' are themselves significant in t or s). Therefore, finding one significant subterm of t or s that is not a subterm of t' or s' will prove $\tau_2 \subset \tau_1$ and thus $size(\tau_2) < size(\tau_1)$. Specifically, we will show that either t is not a subterm of t' and not a subterm of s' or s is not a subterm of t' and not a subterms of s' and, each being a significant subterm of itself, satisfies these conditions to prove $size(\tau_2) < size(\tau_1)$.

Assume, without loss of generality, that $t' \prec t$. We also know that either $s' \prec t$ or $s' \prec s$:

1. If $s' \prec t$, then t is not a subterm of s' and t is not a subterm of t'.
2. If $s' \prec s$, then there are three cases:
    - A. If $s' \prec t'$ then $s' \prec t$, so t is not a subterm of s' and t is not a subterm of t'.
    - B. If $t' \prec s'$ then $t' \prec s$, so s is not a subterm of s' and s is not a subterms of t'.
    - C. If $t' \nprec s'$ and $s' \nprec t'$, then both t and s are not subterms of t' or of s'.

Therefore, $\langle t', s' \rangle \prec_{\mathcal{C}} \langle t, s \rangle$. □

Lemma 16 gives sufficient conditions for a pair of subterms of two terms to be strictly less than the two terms even when one of the subterms is a variable. It will apply in case B of *CR-variable-unify* where the algorithm recurses over a variable and a non-variable term.

Lemma 16: Let s' be a non-variable proper significant subterm of t or s and let x be a variable such that $size(ParSets(x, t) \cup ParSets(x, s)) > 1$, then:

$$\langle x, s' \rangle \prec_{\mathcal{C}} \langle t, s \rangle.$$

Proof. Let $\nu_1 = \nu(t,s)$, $\tau_1 = \tau(t,s)$, $\nu_2 = \nu(x, s')$, and $\tau_2 = \tau(x, s')$. Since s' is a subterm of t or s, all variable occurrences in s' will occur under the same operator sets in either t or s and since x will be in ν, its new occurrence under $\{n\}$ will not place any new variables in ν. Therefore, $\nu_2 \subseteq \nu_1$ and $size(\nu_2) \leq size(\nu_1)$. By the same argument as in Lemma 15, $\tau_2 \subset \tau_1$, and thus $\langle x, s' \rangle \prec_{\mathcal{C}} \langle t, s \rangle$. □

Lemmas 15 and 16 will not be enough to show that the complexity decreases with recursion, since *map-unify* applies the accumulated unifier to the next corresponding pair of terms before calling *CR-unify*. In general, applying a substitution to a term may increase its complexity. Therefore, the following set of definitions and lemmas are used to give sufficient conditions on a substitution and pair of terms such that the substitution will not increase the complexity of the pair of terms.

Each substitution created directly by the *CR-unify* procedure is of the form $\{x \leftarrow r\}$, where x is a variable and r is a term. A substitution of this form has a domain of size one or zero and will be called an *elementary* substitution. The following definition gives sufficient conditions for forming elementary substitutions for a pair of terms to assure that the complexity of the terms will not increase with application of the substitution.

Definition. Let σ be an elementary substitution. σ is said to be *elementary non-increasing* for t and s if and only if it is of one of the following forms:

1. $\{t \leftarrow s\}$, where t is a variable not occurring in s, or similarly, the substitution $\{s \leftarrow t\}$ where s is a variable not occurring in t.

2. $\{x \leftarrow r\}$, where $x \notin \mathcal{V}(r)$ and $(ParSets(x, t) \cup ParSets(x, s)) \cap (ParSets(r, t) \cup ParSets(r, s)) \neq \emptyset$

3. $\{x \leftarrow y\}$, where either y is a new variable or
   $(ParSets(x, t) \cup ParSets(x, s)) \cap (ParSets(y, t) \cup ParSets(y, s)) \neq \emptyset$

4. $\{x \leftarrow r\}$, where r is a non-variable homogeneous term such that $x \notin \mathcal{V}(r)$ and
   $[r.head] \in ParSets(x, t) \cup ParSets(x, s)$ and $\forall\ y \in \mathcal{V}(r)$ either y is a new variable or
   $[r.head] \in ParSets(y, t) \cup ParSets(y, s)$

5. $\{t \leftarrow r\}$, where r is a non-variable homogeneous term such that $t \notin \mathcal{V}(r)$ and
   $[s.head] \in ParSets(t, s)$ and $r.head \underset{\pi}{=} s.head$ and $\forall y \in \mathcal{V}(r)$ either y is a new variable or
   $[r.head] \in ParSets(y, t) \cup ParSets(y, s)$. (Similarly for t and s reversed.)

6. $\{x \leftarrow r\}$, where r is a non-variable homogeneous term such that $x \notin \mathcal{V}(r)$ and there exists a significant subterm, r', of either t or s such that $r.head \underset{\pi}{=} r'.head$ and
   $(ParSets(x, t) \cup ParSets(x, s)) \cap (ParSets(r', t) \cup ParSets(r', s)) \neq \emptyset$ and
   $[r'.head] \in ParSets(x, r')$ and $\forall\ y \in \mathcal{V}(r)$ either y is a new variable or
   $[r.head] \in ParSets(y, t) \cup ParSets(y, s)$.

Lemma 17 states that each of the conditions in the definition of elementary non-increasing substitutions is sufficient to guarantee that application of such a substitution will not cause the complexity of the terms to increase by the $\prec_C$ ordering.

Lemma 17: If σ is an elementary non-increasing substitution for t and s, then:

$$\langle \sigma t, \sigma s \rangle \preceq_C \langle t, s \rangle.$$

Proof. Let $\nu_1 = \nu(t,s)$, $\tau_1 = \tau(t,s)$, $\nu_2 = \nu(\sigma t, \sigma s)$, and $\tau_2 = \tau(\sigma t, \sigma s)$. Consider each case from the definition of elementary non-increasing substitutions:

1. In this case $\sigma t = \sigma s = s$ and since t was not in ν_1, we know $\nu_2 = \nu_1$ and $\tau_2 = \tau_1$ which implies their cardinalities are also pairwise equal.

2. Since r is a subterm of either t or s, $\nu_2 \subseteq \nu_1$. If $\nu_2 = \nu_1$, then all occurrences of x must have been under operators from a single sub-theory (i.e., $size(ParSets(x, t) \cup ParSets(x, s)) = 1$). Furthermore, r must occur under an operator from this set since r and x have some parent set in common. Therefore, the new occurrences of r in σt or σs will be significant if and only if there was a significant occurrence of r in t or s and, thus, $\tau_2 \subseteq \tau_1$.

3. If y is a new variable, then σ is simply a variable renaming, so $size(\nu_2) = size(\nu_1)$. Otherwise, there is a common parent set of x and y, so $size(\nu_2) \leq size(\nu_1)$. In either case, $size(\tau_2) \leq size(\tau_1)$.

4. Let $y \in \mathcal{V}(r)$. If y is a new variable, then all occurrences of y in σt or σs are in r and, since r is homogeneous, $size(ParSets(y, t) \cup ParSets(y, s)) = 1$. Otherwise, there is a common parent set of y and x, so y will occur under symbols from more than one sub-theory in σt or σs only if either x or y did in t or s. Therefore, $size(\nu_2) \leq size(\nu_1)$. If $size(\nu_2) = size(\nu_1)$, then x must have occurred under only one parent set and this set is $[r.head]$. Therefore, any occurrences of r in σt or σs will not be significant and by homogeneity of r, none of its subterms will be significant, so $size(\tau_2) \leq size(\tau_1)$.

5. t occurs under $\{n\}$ and $[s.head]$, so $size(ParSets(t, t) \cup ParSets(t, s)) > 1$. All other variables in r are either new, and by homogeneity of r have only one parent set, or already occur under $[r.head]$ in t or s. Therefore, $size(\nu_2) < size(\nu_1)$.

6. x occurs in t or s under both $[r'.head]$ and the set of symbols distinct from $[r'.head]$ under which r' occurs to make it significant. Therefore, x occurs under more than one set of symbols in t or s and not at all in σt. All other variables in r occur under multiple parent sets in σt or σs only if they did in t or s by the argument in case (5), so $\nu_2 = \nu_1 - \{x\}$.

□

We extend the definition of non-increasing to general substitutions as well as elementary ones by considering a composition of elementary non-increasing substitutions.

Definition. If $\sigma = \sigma_n \circ \ldots \circ \sigma_1$ and σ_i is elementary non-increasing for $\sigma_{i-1} \ldots \sigma_1 t$ and $\sigma_{i-1} \ldots \sigma_1 s$, $i \leq n$, then σ is said to be *non-increasing* for t and s.

Notice that the definition of non-increasing reflects the way in which substitutions are built in *map-unify*; one unifier is found and applied to the inputs before next unifier is found. Lemma 18 proves the desired property on non-increasing substitutions, i.e., applying a non-increasing substitution to the given pair of terms does not increase their complexity. It extends the property in Lemma 17 to substitutions that are not elementary.

Lemma 18: If σ is a non-increasing substitution for t and s, then:

$$\langle \sigma t, \sigma s \rangle \preceq_{\mathcal{C}} \langle t, s \rangle.$$

Proof. By induction on n using Lemma 17. □

Lemma 19: Let t', s' be subterms of t or s such that $(ParSets(t', t) \cup ParSets(t', s)) \cap (ParSets(s', t) \cup ParSets(s', s)) \neq \emptyset$. If σ is non-increasing for t' and s', then σ is non-increasing for t and s.

Proof. The proof is done by examining the elementary factors of σ to show that each of the conditions that makes it non-increasing for t' and s' corresponds to some conditions that make it non-increasing for t and s.

1. $\{t' \leftarrow s'\}$ corresponds to form (2) for t and s.

2. $\{x \leftarrow r\}$, where r is a significant subterm of t' or s' is still form (2) for t and s since
   $ParSets(x, t') \cup ParSets(x, s') \subseteq ParSets(x, t) \cup ParSets(x, s)$ and
   $ParSets(r, t') \cup ParSets(r, s') \subseteq ParSets(r, t) \cup ParSets(r, s)$.

3. $\{x \leftarrow y\}$ for t' and s' is still form (3) for t and s
   $ParSets(x, t') \cup ParSets(x, s') \subseteq ParSets(x, t) \cup ParSets(x, s)$ and
   $ParSets(y, t') \cup ParSets(y, s') \subseteq ParSets(y, t) \cup ParSets(y, s)$.

4. $\{x \leftarrow r\}$, where r is a homogeneous term is also form (4) for t and s since
   $ParSets(x, t') \cup ParSets(x, s') \subseteq ParSets(x, t) \cup ParSets(x, s)$ and
   $\forall y \in \mathcal{V}(r)$ if $[r.head] \in (ParSets(y, t') \cup ParSets(y, s'))$, then
   $[r.head] \in (ParSets(y, t) \cup ParSets(y, s))$.

5. $\{t \leftarrow r\}$, where r is homogeneous in $[s.head]$ is form (6) for t and s. The term s' acts as r' in form (6) of the definition.

6. $\{x \leftarrow r\}$, where r is homogeneous and there is an r' as described, is still form (6), since all significant subterms of t' and s' (including t' and s', themselves) are significant in t or s.

□

The following technical definition will be used in our proof of termination to show that non-trivial recursive calls are made only to pairs of terms that are significant and appear under a common set of symbols in the input terms with accumulated partial unifiers applied. The property is not obvious in the proof since we are building the corresponding pairs of terms out of substitutions ρ and γ, not directly from t and s.

Definition. Let t and s be two homogeneous terms in F, the constrained symbols for a sub-theory of

E* and let ρ be a substitution that is homogeneous in F. Furthermore, assume either t or s is non-variable. A substitution, σ, is said to be *parent preserving* for homogeneous t, s, and ρ, if and only if:

$$(x \in X \;\&\; x \in \mathfrak{D}(\sigma) \cup \mathfrak{I}(\sigma)) \Rightarrow F \in \mathit{ParSets}(\sigma x, \sigma t) \cup \mathit{ParSets}(\sigma x, \sigma s),$$
$$\text{where } X = \mathfrak{V}(\hat{t}) \cup \mathfrak{V}(\hat{s}) \cup \mathfrak{I}(\rho).$$

Informally, X contains the variables from t or s that were not cut off though homogenization, the new variables from homogenization, and any new variables from ρ. The intuition is that if one of the variables occurs in the range variables of σ, then it must occur somewhere in σt or σs, and, furthermore, one of those occurrences must be under F. In addition, if one of the variables, x, occurs in the domain of σ, then the range element σx must occur in σt or σs under F. It is called the parent preserving property because the variables in X all occur under F in either $\hat{t}$, $\hat{s}$, $\rho\hat{t}$ or $\rho\hat{s}$, and we want this parent relationship to be preserved, even when the unifier, σ, is only partially formed.

3.4.4 The Proof

Theorem V is the termination theorem for the *CR-unify* procedure. The proof is by noetherian induction on the complexity of terms in the calls to *CR-unify*. The induction step is proved using a second induction on the number of calls to *CR-unify* made from a single invocation of *map-unify*.

The proof is quite long and involved. Before presenting the proof in its entirety, we will give a short outline of the proof's structure. This should give the reader a feeling for the purpose of each step in the proof and also act as a check-list for the things that have been proved and the things still left. The proof is by induction on t and s using the $\prec_{\mathcal{C}}$ ordering. The proof uses a stronger induction hypothesis than the property of termination alone. In addition to termination, we prove that returned substitutions are non-increasing for t and s.

Proof Idea.

1. Basis: Cases 1 and 4 of *CR-unify* and cases A and C of *CR-variable-unify*, as invoked in case 2 or 3 of *CR-unify*. We need to prove that each step terminates and that the returned substitution is non-increasing for t and s.

2. Inductive step: Case 5 of *CR-unify* and case B of *CR-variable-unify* (as called by case 2 or 3 of *CR-unify*) are the inductive cases, since these two cases require recursive steps. These cases can be considered together, since both t and $\hat{t}$ for case 5 are v in case B. In this step we will show that calls to *map-unify* terminate. For simplicity, we treat the algorithms as if they always pick some element of a returned set of unifiers rather than exhaustively trying each element of the set. Since these sets of unifiers are finite, this simplification does not affect the soundness of our proof.

We will perform an induction on the variables in the domain of γ and ρ. These variables are divided into two sets denoted V_1 and V_2, where V_2 contains variables representing non-variable subterms of t or s and V_1 contains variables from the original t or s. The induction on these variables is divided into two parts, one for V_1 and one for V_2.

In addition to proving the termination of each step and the non-increasing nature of unifiers, we will show that σ_i is parent preserving for $\hat{t}$, $\hat{s}$, and ρ. The proof that σ_i is parent preserving, will depend only on the induction on i, not on the induction on t and s since in the base cases ρ, $\hat{t}$, and $\hat{s}$ are not defined.

A. This step is for variables in V_1.

 i) Basis: $i = 0$, the first variable in the V_1.

 ii) Inductive step: $i > 0$, the rest of the variables in V_1.

 Show that this step terminates and any substitution formed is both non-increasing for t and s and parent preserving for $\hat{t}$, $\hat{s}$, and ρ.

B. This step is for variables in V_2.

 i) Basis: Since V_1 and V_2 are processed in order, this is simply the last case in V_1.

 ii) Inductive step: For the variables in V_2

 Show that this step terminates and, again, that any substitution formed is both non-increasing for t and s and parent preserving for $\hat{t}$, $\hat{s}$, and ρ. We break this step into five cases depending on the value of s_i, the term formed from ρ, and t_i, the term formed from γ.

Given this rough outline, we now give the termination theorem with complete proof.

Theorem V: For any terms, t and s, *CR-unify*(t, s) terminates.

Proof. By induction on t and s using the $\prec_C$ ordering.

1. Basis: Cases 1 and 4 of *CR-unify* and cases A and C of *CR-variable-unify*, as invoked in case 2 or 3 of *CR-unify*. In each case termination is obvious; We will prove that a returned substitution, σ, is non-increasing for t and s.

 A. Case 1: $\sigma = \{t \leftarrow s\}$ is non-increasing because it is either the identity substitution or of form (1) in the definition of elementary non-increasing.

 B. Case A: $\sigma = \{t \leftarrow s\}$ is of form (1) in the definition of elementary non-increasing.

C. Case C. The empty set of substitutions is returned, so the properties are vacuously true.

D. Case 4: Again, the empty set of substitutions is returned.

2. Inductive step: Case 5 of *CR-unify* and case B of *CR-variable-unify* (as called by case 2 or 3 of *CR-unify*) are the inductive cases. Let E_i^* be the sub-theory of interest and F be the set of function symbols constrained by E_i^*, i.e., F = [*s.head*].

E_i_*CR-unify* terminates by the basic assumption on sub-theory unification procedures.

Map-unify is called with γ, the preserving substitution for t and s, and ρ, an E_i-unifier. Induct over i, the number of iterations in *map-unify*, and for each i use the following notation:

$$\nu_i \in \mathfrak{I}(\gamma) \cup \mathfrak{I}(\rho)$$
$$t_i = \sigma_{i-1} \circ \gamma \nu_i$$
$$s_i = \sigma_{i-1} \circ \rho \nu_i$$
$$\omega_i \in CR\text{-}unify(t_i, s_i)$$
$$\sigma_i = \omega_i \circ \sigma_{i-1}$$

Using this notation, we know σ_i is of the form:

$$\sigma_i = \omega_i \circ \omega_{i-1} \circ \ldots \circ \omega_0.$$

The set of domain variables is divided into V_1 and V_2; we will consider each case separately, first performing induction on i where $\nu_i \in V_1$ and then where $\nu_i \in V_2$. We show that σ_i is parent preserving for $\hat{t}$, $\hat{s}$, and ρ.

A. $\nu_i \in V_1 = \mathfrak{D}(\rho) - \mathfrak{D}(\gamma)$: We know by construction of V_1 that $\gamma \nu_i = \nu_i$. In this case, each unifier is of the form $\omega_i = \{\nu_i \leftarrow \sigma_{i-1} \rho \nu_i\}$.

   i) Basis: If i = 0, then no calls to *CR-unify* have been made and termination is obvious. The only substitution is $\sigma_0 = \iota$, which is trivially non-increasing and parent preserving.

   ii) Inductive step: i>0.

   $\sigma_{i-1} = \omega_{i-1} \circ \ldots \circ \omega_0$ and by the induction hypothesis on i, the ω's are all elementary substitutions of the form $\omega_i = \{\nu_i \leftarrow \sigma_{i-1} \rho \nu_i\}$. Therefore, $\mathfrak{D}(\sigma_{i-1}) \subseteq \{\nu_1, \ldots, \nu_i\}$ and, furthermore, $\nu_i \notin \mathfrak{D}(\sigma_{i-1})$, so $t_i = \sigma_{i-1} \gamma \nu_i = \sigma_{i-1} \nu_i = \nu_i$.

   Also, by construction of V_1 we know $\nu_i \in \mathfrak{D}(\rho)$, which implies $\nu_i \notin \mathfrak{I}(\rho)$ and thus $\nu_i \notin \mathfrak{I}(\sigma_{i-1})$. Therefore, $\nu \notin \mathcal{V}(\sigma_{i-1} \rho \nu_i)$.

   Since t_i does not occur in $s_i = \sigma_i \rho \nu_i$, the unifier is $\omega_i = \{t_i \leftarrow s_i\} = \{\nu_i \leftarrow \sigma_{i-1} \rho \nu_{i-1}\}$, which is case A of *CR-variable-unify*, and obviously a terminating case.

ω_i, is homogeneous in F and elementary non-increasing for $\sigma_{i-1}t$ and $\sigma_{i-1}s$ because it is of form (3), (4), or (5) in the definition of elementary non-increasing. Therefore, σ_i is non-increasing for t and s by Lemma 18.

v_i occurs under F in either t or s, and σ_i is homogeneous in F, so the parent preserving hypothesis holds. (Note: this is true even if v_i is t_i or s_i as in case B of *CR-variable-unify*, since v_i must have another occurrence under F in the non-variable term.)

B. $v_i \in V_2 = \mathfrak{D}(\gamma)$:

i) Basis: The proof for v_i in V_1 proves the base case for V_2.

ii) Induction Step:

Since $v_i \in \mathfrak{D}(\gamma)$ we know γv_i and thus t_i is a non-variable such that the head symbol is not in F. Therefore, t_i is a proper significant subterm of $\sigma_{i-1}t$ or $\sigma_{i-1}s$.

There are five cases to consider for s_i.

a) If s_i is a non-variable such that $s_i.head \neq t_i.head$, then the recursive call terminates with the empty set in case 4 of *CR-unify*.

b) If s_i is a variable such that $s_i \in \mathcal{V}(t_i)$ & $s_i \in \mathfrak{I}(Preserve(\hat{f}_i, t_i))$, then the termination halts with the empty set in case C of *CR-variable-unify*.

c) If s_i is a variable such that $s_i \notin \mathcal{V}(t_i)$ then the single substitution returned by *CR-unify*(t_i, s_i) is $\omega_i = \{s_i \leftarrow t_i\}$.

ω_i is of form (2) for $\sigma_{i-1}t$ and $\sigma_{i-1}s$ in the definition of elementary non-increasing, and since the induction hypothesis on i implies σ_{i-1} is non-increasing for t and s, $\sigma_i = \omega_i \circ \sigma_{i-1}$ is non-increasing for t and s.

Termination is immediate in the recursive call since t_i and s_i fit case A of *CR-variable-unify* as called from case 3 of *CR-unify*.

To show the parent preserving hypothesis on σ_i, we will consider two cases: the variables in the domain of σ_i and the variables in the range of σ_i. Recall the variables of interest, X, are either variables in t or s with occurrences under F, new variables from homogenization, or new variables from E_i-*unify*.

Consider x in $\mathfrak{D}(\sigma_i)$, and show that $\sigma_i x$ will occur under F in either $\sigma_i t$ or $\sigma_{i-1} s$. This is evident not because of any conditions of x, but because γv_i occurred under F in t or s and therefore $\sigma_i x = s_i = \sigma_{i-1}\gamma v_i$ will occur under F in $\sigma_i t$ or $\sigma_i s$.

Let x be in $\mathfrak{I}(\sigma_i)$ and show that x will occur under F in $\sigma_i t$ or $\sigma_i s$. If $x \in \mathfrak{I}(\sigma_i)$ then either $x \in \mathfrak{I}(\sigma_{i-1})$ or $x \in \mathcal{V}(s_i)$. Variables, other than those in $\mathfrak{D}(\omega_i)$, that are in $\mathfrak{I}(\sigma_{i-1})$ will occur under F by the induction hypothesis on i. Variables in $s_i = \sigma_{i-1}\gamma v_i$ are either in t or s or in $\mathfrak{I}(\sigma_{i-1})$. In the first case, if variables occur in t or s and in X, then they must have occurred under F in either t or s. In the second case, variables in σ_{i-1} and X will occur under F in $\sigma_{i-1} t$ or $\sigma_{i-1} s$ by the induction hypothesis and thus under F in $\sigma_i t$ or $\sigma_i s$.

d) If s_i is a non-variable such that $s_i.head = t_i.head$, then:

Since $[t_i.head] \neq$ F we know $[s_i.head] \neq$ F. But $s_i = \sigma_{i-1}\rho v_i$, so ρ maps v_i to itself or to some other variable. In either case there is some variable, $y \in \mathfrak{D}(\sigma_{i-1})$ and $y \in$ X such that $\sigma_{i-1} y = s_i$. Therefore, by the parent preserving property on σ_{i-1}, s_i occurs under F in either $\sigma_{i-1} t$ or $\sigma_{i-1} s$.

Therefore, t_i and s_i are both non-variables significant subterms of $\sigma_{i-1} t$ or $\sigma_{i-1} s$ and by Lemma 15, $\langle t_i, s_i \rangle \prec_{\mathcal{C}} \langle \sigma_{i-1} t, \sigma_{i-1} s \rangle$.

By the inductive hypothesis on i, σ_{i-1} is non-increasing for t and s and thus $\langle \sigma_{i-1} t, \sigma_{i-1} s \rangle \preceq_{\mathcal{C}} \langle t, s \rangle$ by Lemma 18.

By transitivity, $\langle t_i, s_i \rangle \prec_{\mathcal{C}} \langle t, s \rangle$. I.e., the recursive call is made to strictly smaller inputs than the original inputs, and we can apply the noetherian induction hypothesis to prove termination.

By the induction hypothesis on t_i and s_i, if ω_i is a returned substitution, then ω_i is non-increasing for t_i and s_i, and by Lemma 19 for t and s since t_i and s_i both occur under F.

The parent preserving property is proved by showing that if $x \in \mathfrak{D}(\sigma_i) \cup \mathfrak{I}(\sigma_i)$ then $\sigma_i x$ occurs under F in $\sigma_i t$ or $\sigma_i s$. Since $\sigma_i = \omega_i \circ \sigma_{i-1}$, x is either in $\mathfrak{D}(\sigma_{i-1}) \cup \mathfrak{I}(\sigma_{i-1})$ or in $\mathfrak{D}(\omega_i) \cup \mathfrak{I}(\omega_i)$. In the first case, the parent preserving property in σ_i follows directly from the induction hypothesis on σ_{i-1}. In the second case, if x is in $\mathfrak{D}(\omega_i)$ then it must be in t_i or s_i, and if x is in $\mathfrak{I}(\omega_i)$

then it must be in t_i, s_i or it must be a new variable and therefore not be in X. We therefore need only consider the variables in t_i and s_i. The variables in t_i are either in $\mathfrak{I}(\gamma)$, and thus in t or s, or they are in $\mathfrak{I}(\sigma_{i-1})$; the variables in s_i are all in $\mathfrak{I}(\sigma_{i-1})$. All variables in $\mathfrak{I}(\sigma_{i-1})$ are covered by the induction hypothesis, and variables in t or s occur in X only if they occur under F, so ω_i and hence σ_i are parent preserving.

e) If s_i is a variable such that $s_i \in \mathfrak{N}(t_i)$ and $s_i \notin \mathfrak{I}(Preserve(\hat{f}_i, t_i)$, then:

Since $s_i = \sigma_{i-1}\rho\nu_i$, is not in t or s but is in t_i, we know $s_i \in \mathfrak{I}(\sigma_{i-1})$.

$t_i \in \mathfrak{N}(s_i) \Rightarrow t_i \in \mathfrak{I}(\sigma_{i-1} \circ \rho)$. Therefore, either $t_i \in \mathfrak{I}(\rho)$ or there exists some $\nu \in \mathfrak{I}(\rho)$ such that $t_i = \sigma_{i-1}\nu$.

In either case, the inductive hypothesis applies to give the parent preserving property on σ_{i-1} and s-(i) occurs under F.

But, s_i also occurs under $[t_i.head]$. Therefore, $size(ParSets(s_i,\ \sigma_{i-1}t) \cup ParSets(s_i,\ \sigma_{i-1}s)) > 1$ and by Lemma 16, $\langle t_i, s_i\rangle \prec_{\mathcal{C}} \langle \sigma_{i-1}t, \sigma_{i-1}s\rangle$.

By the same arguments as in case 4, $\langle t_i, s_i\rangle \prec_{\mathcal{C}} \langle t, s\rangle$ and we can use the inductive hypotheses to show $\sigma_i = \omega_i \circ \sigma_{i-1}$ is parent preserving and non-increasing for t and s.

□

Chapter Four

Conclusions

This chapter summarizes the main contributions of this thesis and suggests areas for future work. We look specifically at possible extensions of *CR-unify* and some issues and related to the complexity and efficiency of our approach.

4.1 Contributions

In this thesis we defined a generalization to the equational unification problem which we called combined unification. The problem is to take a set of equational theories, each of which has a known equational unification algorithm, and automatically produce an algorithm for the union of the theories. The problem solved here, where we restricted ourselves to confined regular theories, was mentioned as an open problem by [Shostak 84] as a generalization to his work in combining decision procedures. It is also described as an open problem in [Siekmann 84], where it is called the *combination of theories* problem.

The combined unification problem was motivated both by a theoretical interest in learning more about equational unification and a pragmatic interest in using equational unification in larger systems such as theorem provers. Many applications of unification involve reasoning about an arbitrary sets of operators with different set of equational properties. For example, in an automatic theorem prover the set of needed equational theories will depend on the theorem to be proved. Moreover, given the difficulty of designing equational unification algorithms, it is not reasonable to assume the algorithm will be designed "on the fly," while proving the theorem. These considerations lead us to conclude two things: systems based on equational unification must have a large set of built-in theories, and, this set must be easily extendible so that new theories can be incorporated as they are deemed interesting.

Our approach was to describe a unification algorithm for equational theories that are the union of theories having known unification algorithms. The main results of this thesis are:

- a careful definition of the combined unification problem,

- characterization of a sub-problem of combined unification for which a solution was possible, and for which extensions outside the sub-problem domain are difficult,

- design of an algorithm for combining equational unification algorithms for confined regular theories,

- proofs of consistency, completeness, and termination for the algorithm,

- a method for a lifting the variable-only case to the case with free symbols and multiple instances using our combining algorithm,

- an implementation of the algorithm.

Our definition of the combined unification problem is based on the existence of what we called a partitioned presentation of the theory. A partitioning on the presentation characterizes the independence of its sub-theories, i.e., the theory presented by each of the partitions. This intuitive notion of independent sub-theories corresponds to having disjoint sets of constrained symbols for each sub-theory. Independence is essential to the ability to automatically combine unification algorithms as evidenced by the fact that combining three theories with decidable unification problems, namely associativity, left distributivity and right distributivity, results in a theory with an undecidable unification problem. (See Figure 1-4 for references.)

The sub-problem of combined unification that we chose to solve was combining confined regular theories. Both of these properties give sufficient conditions for finding pairs of terms that are not unifiable. We defined a confined theory to be one in which there are no equations with a variable equal to a non-variable term. This gave us the invariant that any two equal non-variable terms have head symbols constrained by the same sub-theory. A regular theory is a theory in which the right and left side of each equation contains the same set of variables. Together with the restriction to confined theories, the regularity restriction guarantees that every pair of equal terms has equal sets of significant subterms.

We also showed in Section 2.4 that confinement and regularity were not restrictions of convenience, but that both our algorithm and a number of simple extensions to it were not correct for unconfined or non-regular theories. Taken collectively, the examples in Section 2.4 point out a conflict between the goals of completeness and termination of E-unification procedures. In many cases it was possible to either prove completeness assuming termination or to prove termination of an incomplete procedure.

The examples in Section 2.4 also serve as counter-examples to the correctness of some existing E-unification algorithms when more than one instance or free symbols are allowed. The abelian group unification algorithm and the algorithm for AC unification with either idempotence or unit are examples of algorithms that are not correct in the more general cases. The ACI and ACU theories have variable-only case algorithms defined in [Livesey 76]. [Fages 83b] shows termination for the general cases of ACI and ACU but neither his algorithm nor the algorithms in [Livesey 76] are complete in these cases. In particular, they do not find a unifier when unifying a variable with a non-variable term containing the variable. For the same reason, the algorithm for abelian group unification [Lankford 84] is incomplete in the general cases.

Our algorithm is a combining algorithm for the confined regular theories. It automatically builds an algorithm for the combined unification problem by dividing the combined problem into a number of variable-only problems, invoking algorithms for the sub-theories on those problems, and recombining the answers. A homogenizing function is used for the problem division and unification of substitutions used for the recombining. The technique requires no redesign of the sub-theory algorithms, and no theory-specific computation in the generalized algorithm. The restrictions of regularity and confinement are implicitly used for correctness, but are never explicitly tested by the algorithm.

The practical significance of our results depend upon the existence of confined regular theories for which the combined unification problem was previously unsolved, and for which the theories describe "interesting" properties. The permutative theories are examples of such theories. The insert operation on the *set* data type, for example, has the property of right commutativity, a permutative property that also has known unification algorithm.

An algorithm for unifying permutative axioms is useful in term rewriting system completion procedures and in resolution, because both of these procedures make use of classical unification and are limited by the necessity of maintaining termination. The permutative axioms are a class that will always lead to termination problems, since they can be applied repeatedly. The non-permutative axioms can often be handled directly by the application, such as resolution or term rewriting, without loosing termination. It is often the case that equations leading to termination problems in an application can be handled by combined unification, while the equations leading to problems in combined unification can be handled by the application. Referring again to the *set* example, we note that most axioms for this data type can be handled in a term rewriting system by placing them in the rules. An exception is the right commutativity of insert, which would violate properties necessary for termination. Right commutativity can be handled by our *CR-unify* algorithm, and is allowed in the unification algorithm in an equational term rewriting system. We therefore have a method for proving theorems about sets of terms or sets of sets by incorporating the combining unification algorithm with right commutative unification into an equational term rewriting system.

Another class of theories we have discussed are those theories presented by only ground equations. These theories have an automatically generated unification algorithm through narrowing [Hullot 80], and are also confined and regular. While the case with free symbols and multiple instance case are solved by the narrowing process, our algorithm provides a method for combining these narrowing unification algorithms with arbitrary other algorithms for confined regular theories.

A proof of total correctness for the *CR-unify* algorithm was given in Chapter 3. The proof was presented in three separate theorems: Theorem III shows consistency, Theorem IV completeness, and Theorem V termination. The consistency and termination properties are proved for all theories, while completeness is proved for the regular confined theories. The consistency and completeness proofs assume termination, and therefore constitute a partial completeness argument. Combined with the termination proof this shows total correctness. The termination proof was based on Fages's proof of termination for AC-unification. However, the extension was non-trivial, particularly because case B of *CR-unify* is a failure case in the AC theory, whereas case B is a recursive case in *CR-unify* to handle theories presented by ground equations.

A consequence of our method and the proofs is that any variable-only case algorithm for an equational theory E can be automatically lifted to both the multiple instance case and case with free symbols. The importance of this is demonstrated by considering an example. A unification algorithm for the AC theory was originally described by Stickel in 1975, and although the procedure was purported to solve the general cases for AC, termination in the general case was unproven until 1983, when it was solved by Fages. The AC theory is one example of a confined and regular theory; Fages's generalization of Stickel's algorithm is a special case of our *CR-unify* just as his termination proof is a special case of our termination proof for *CR-unify*. The unification problem with free symbols is also immediately solved by our algorithm for any theory with a variable-only case algorithm, because the unconstrained symbols are handled by the empty theory unification algorithm.

The ability to go directly from a variable-only case solution to the more general solutions also give us some bounds on the number of unifiers in certain theories. As a corollary to our proof of total correctness, termination in the variable-only case of a confined and regular theory implies termination in both the multiple instance case and the case with free symbols of this theory. This in turn implies the existence of a finite complete set of unifiers in these more general cases. Conversely, if either the case with free symbols or the multiple instance case is known to be infinite for a given theory, then the variable-only case must also be infinite.

The *CR-unify* algorithm has been implemented as part a general effort to extend the REVE term rewriting system generator [Lescanne 83, Forgaard 84b] to equational term rewriting systems

[Kirchner 84b]. The implementation supports the generalized unification algorithm and allows for simple modular extension to new sub-theories as their unification algorithms are implemented. In the current version, the unification algorithms for the AC and empty theories have been implemented. The implementation of the REVE system, including our unification algorithm, was done in CLU [Liskov 81].

For efficiency reasons, the implementation differs from the description given in this thesis. Each E-unification procedure is implemented to perform unification in a single equational theory, making no assumptions about the properties of the operators in the subterms, but recursively calling the top-level general unification procedure to unify subterms rather than returning homogeneous substitutions and combining them through unification of substitutions. This eliminates the overhead of forming the homogeneous terms. The sets of substitutions that appear in both *CR-unify* and *map-unify* are replaced in the implementation with CLU iterators. This simplifies the code and may allow for combining non-terminating unification procedures, such as that for the associative theory [Plotkin 72]. Finally, as in Fages's implementation of AC-unification, the order of recursion imposed in *map-unify* by forming V_1 and V_2 is not preserved. I.e., variable elements in $\mathcal{R}(\gamma)$ are not necessarily unified first. The assumption on the order of recursion is made only to simplify the termination proof.

4.2 Future Work

The work in this thesis has suggest some areas for further research. We will discuss, on a pragmatic level, some techniques for improving the efficiency of our algorithm. We also consider the problem of weakening the restrictions on the equational theories allowed in the combining algorithm, and present directions for further work in this area.

4.2.1 Efficiency Issues

The feasibility of using E-unification in applications will depend in part on the ability to find reasonably efficient algorithms for performing the unification. Although the combined unification problems are inherently hard for many interesting theories, there are a number of optimizations that will improve the running time in practice.

One of these optimizations comes from the difference between *U-Homog* and *Homog*, defined in this thesis. *CR-unify* is still correct if *U-Homog* or other homogenizing functions are used in place of *Homog* in the procedure. There is a trade-off between the efficiency of the homogenizing function and the number of unifiers of the homogenized terms, since terms with multiple occurrences of variables have fewer unifiers than terms in which each variable is unique. The best homogenizing function may be one that performs differently depending on the theory of the head symbols.

The order of recursive calls is also very important to the running time of combined unification. In practice it is best to perform the simplest recursive calls first, especially when they will lead to failure. Ordering of recursive calls can be done partially on the basis of the kind of terms to be unified (i.e., variable versus non-variable) and partially on the basis of the relative difficulty of performing a unification in the different theories [Fages 85].

Perhaps the most interesting class of optimizations would involve weakening our strict boundaries between sub-theory unification algorithms. There are sufficient conditions on terms for non-unifiability that can be checked very quickly, e.g., clash of head symbols. Sometimes, a sub-theory unification algorithm can make use of this information during its processing and thereby never produce unifiers that would require two obviously non-unifiable terms to be unified. A clash between symbols from different sub-theories could be detected in the current structure of the algorithm, but more general kinds of checks for non-unifiability are specific to a theory and would therefore requires sharing information between unification algorithms. Our current implementation does not make use of this kind of information because the emphasis was on correctness and modularity of the program rather than efficiency.

A measure of complexity that exists for unification problems, and is also related to the efficiency in practice, is minimality of a solution. An algorithm that is totally correct but produces many non-minimal unifiers will be too inefficient for some applications. Minimality is probably too expensive to require of unification algorithms, since in some theories this would require an exponential filtering process. However, non-minimal unifiers affect the execution time of both the algorithm and its applications and thus one measure of a good algorithm should be that it produces few non-minimal unifiers. A related open problem is to find a minimal combining unification algorithm that avoids exponential filtering.

Efforts to gain significant improvements in efficiency through parallel processing are limited by some lower bounds in that area. First, note that E-unification problems for which the best algorithm is exponential will have at best exponential parallel algorithms, since we have only a polynomial number of processors. For theories in which polynomial unification algorithms exist, improvements through parallel processing may be possible. However, the fact that empty theory unification is inherently sequential [Dwork 84] is not promising.

4.2.2 Removing Restrictions on the Theories

A challenging area of research that is not addressed in this thesis is the problem of combining theories in which constrained symbols are not disjoint. Some of the negative results on the decidability of unification problems indicate the difficulty of this problem. The undecidability results in this area have been based on the undecidability of Hilbert's 10th problem, solving Diophantine equations over the integers, which was shown undecidable by Matiyasevič [Davis 73]. [Arnborg 85] and [Szabo 78] investigate combinations of theories with non-disjoint function symbols by studying the lattice of sub-theories that are consistent with Peano arithmetic. [Szabo 78] shows the undecidability of the associative theory with two sided distributivity, AD, while [Arnborg 85] shows the undecidability of the associative theory with one-sided distributivity and a right and left identity element, AD_lU or AD_rU. In both cases it was also shown that any theory consistent with Peano arithmetic and containing the AD or AD_1U theory, respectively, also has an undecidable unification problem.

The *CR-unify* described in this thesis cannot handle non-terminating procedures for enumerating unifiers. For the sake of notational convenience, the *CR-unify* invokes a sub-theory algorithm which returns a complete set of unifiers, although processing of unifiers could be done one at a time. It may be possible to get a combining algorithm for non-terminating unification procedures by processing on sub-theory unifier before the others are generated. The interesting problem here is to show that the resulting procedure is a complete generating procedure for the combined theory.

An obvious problem that is left open in this thesis is combining unification algorithms when the theories may be either unconfined or non-regular. The problems that arise in trying to extend *CR-unify* were characterized by the examples in Section 2.4. We found, in general, that it was not hard to guarantee consistency, but that the properties of termination and completeness seem to conflict. It was possible to get a provably complete procedure if one assumed termination, while in actuality the procedure would loop in a trivial manner before any unifiers were generated. Alternatively, a terminating algorithm could be achieved, but it was found that the algorithm was incomplete on some non-trivial examples.

Before our algorithm can be extended to unconfined or non-regular theories, further theoretical work must be done. The experience gained in this work shows that the problem of generalizing unification procedures is not trivial, that seemingly obvious approaches are not always correct, and, therefore, that algorithms in this field require detailed descriptions and careful proofs of consistency, completeness, and termination.

Appendix A

Protection of Variables in *CR-unify*

A technical issue that we have avoided discussing in detail until now is the generation and protection of new variables in *CR-unify*. A unification algorithm is often used in a larger system, and that system may have variables of its own, and it is important that any new variables generated by *CR-unify* do not coincide with those existing externally. This problem could be handled by simply renaming variables after performing unification, but the more general problem comes up within the *CR-unify* algorithm because of the recursion. It is important to all three correctness properties, i.e., consistency, completeness and termination, that new variables generated on recursive calls do not coincide with those existing in subterms not involved in the recursion. For example, if two new variables appear together and both use the same name, the resulting unifier may be less general than intended substitution. In particular, the property of idempotence of unifiers, used in the proofs, depends on the disjointness of domain and range variables. This disjointness can only be guaranteed if the variables in the domain of one factor of a substitution can be protected from appearing in the range of another factor. The parent preserving property of the termination proof also depends on the protectiveness of recursive calls to *CR-unify*.

In the implementation the protection problem corresponds to the problem of generating globally unique identifiers from within any local procedure environment. The solution in the implementation is to pass an object for generating unique identifiers to each unification procedure and to guarantee *a priori* that all variables in the input are disjoint from any variables that may be generated. We do this by picking a special prefix for generated variables and concatenating a unique integer whenever a new variable is needed. In the formal context, it is more convenient to pass the set of variables to be protected than to pass a function for generating identifiers, although the two approaches are effectively the same.

We begin by imposing a total order on the universe of variables, **V**. If V is any set of variables, *Next*(V) denotes the smallest variable in V, as defined by the imposed ordering. The homogenizing function, *Homog*, is then modified to incorporate the protected set of variables. The following modification to the definition of *Homog* will legitimize calling *Homog* a function, since it will now be mathematically well-defined. The definition given here replaces the earlier one.

Definition. Let F be a set of function symbols, W be a finite set of variables, and t be a term such that $\mathcal{V}(t) \subseteq$ W. *Homog*(t, F, W) is defined as follows:

1. If t is a variable, then *Homog*(t, F,) = t.

2. If $t = f(t_1,\dots,t_n)$ and $f \in$ F, then
   Homog(t. F. W) = f(*Homog*(t_1. F, W$_1$),...,*Homog*(t_n, F, W$_n$)), where W$_1$ = W and for i > 1,
   W$_i$ = W$_{i-1}$ $\cup$ $\mathcal{V}$(*Homog*(t_{i-1}. F. W$_{i-1}$)).

3. If $t = f(t_1,\dots,t_n)$ and $f \notin$ F, then *Homog*(t, F, W) = *Next*(V−W).

Figures 4-1 and 4-2 show the *CR-unify* and *map-unify* procedures, respectively. For the sake of consistency between Figure 2-3 and 4-1, we will abuse our notation slightly. The shorthand form of the homogenizing function, $\hat{t}$, was used in the less formal description of *CR-unify* in Figure 2-3. Because it does not allow for specification of the the set of protected variables, we will use the longhand form, *Homog*, and use $\hat{t}$ and $\hat{s}$ as identifiers in the code; they represent the same values as in Figure 2-3.

For simplicity we assume the set of protected variables, W, contains all variables in the other inputs arguments. I.e., $\mathcal{V}(t) \cup \mathcal{V}(s) \subseteq$ W in calls to the *CR-unify* procedure, $\mathcal{V}(s) \cup \{v\} \subseteq$ W in *CR-variable-unify*, and $\mathcal{I}(\varphi_1) \cup \mathfrak{I}(\varphi_1) \cup \mathcal{I}(\varphi_2) \cup \mathfrak{I}(\varphi_2) \subseteq$W in *map-unify*.

```
CR-unify = proc ($t$, $s$: term, W:var_set) returns (subst_set)
    case
        is_variable($t$) and is_variable($s$) $\Rightarrow$              % case 1
            return($\{\{t \leftarrow s\}\}$)
        is_variable($t$) and ~is_variable($s$) $\Rightarrow$             % case 2
            return(CR-variable-unify($t$, $s$, W))
        is_variable($s$) and ~is_variable($t$) $\Rightarrow$             % case 3
            return(CR-variable-unify($s$, $t$, W))
        $t.head \underset{\pi}{\neq} s.head \Rightarrow$                         % case 4
            return ($\emptyset$)
        $t.head \underset{\pi}{=} s.head \Rightarrow$                         % case 5
            $\hat{t} :=$ Homog($t$, $t.head$, W)
            $\hat{s} :=$ Homog($s$, $t.head$, $W \cup \mathcal{V}(\hat{t})$).
            $\gamma :=$ Preserve($t$, $\hat{t}$) $\cup$ Preserve($s$, $\hat{s}$)
            $P := E_{[t.head]}$-unify($\hat{t}$, $\hat{s}$, $W \cup \mathcal{V}(\hat{t}) \cup \mathcal{V}(\hat{s})$)
            $\Sigma := \bigcup_{\rho \in P}$ map-unify($\rho$, $\gamma$, $W \cup \mathcal{V}(\hat{t}) \cup \mathcal{V}(\hat{s}) \cup \mathcal{I}(\rho)$)
            return($\Sigma \restriction_{\mathcal{V}(t) \cup \mathcal{V}(s)}$)
            end
        end
    end CR-unify
CR-variable-unify = proc ($v$: variable, $s$: term, W:var_set) returns (subst_set)
    $\hat{s} :=$ Homog($s$, $s.head$, W)
    $\gamma :=$ Preserve($s$, $\hat{s}$)
    case
        $v \notin \mathcal{V}(s) \Rightarrow$                                % case A
            return ($\{\{v \leftarrow s\}\}$)
        $v \in \mathcal{V}(s)$ & $v \notin \mathcal{I}(\gamma) \Rightarrow$                      % case B
            $P := E_{[s.head]}$-unify ($v$, $\hat{s}$, $W \cup \mathcal{V}(\hat{s})$)
            $\Sigma := \bigcup_{\rho \in P}$ map-unify($\rho$, $\gamma$, $W \cup \mathcal{V}(\hat{s}) \cup \mathcal{I}(\rho)$)
            return($\Sigma \restriction_{\{v\} \cup \mathcal{V}(s)}$)
        $v \in \mathcal{V}(s)$ & $v \in \mathcal{I}(\gamma) \Rightarrow$                      % case C
            return({})
        end
    end CR-variable-unify
```

Figure 4-1: A Careful Description of the *CR-unify* Procedure

```
map-unify = proc ($\varphi_1$, $\varphi_2$:subst, W:var_set) returns(subst_set)
    $V_1 := \mathfrak{D}(\varphi_1) - \mathfrak{D}(\varphi_2)$
    $V_2 := \mathfrak{D}(\varphi_2)$
    $\Sigma_0 := \{\iota\}$
    $i := 0$
    $W_0 := W$
    for j = 1 to 2 do
        for $\nu$ in $V_j$ do
            $i := i + 1$
            $\Sigma_i := \{\omega_i \circ \sigma_{i-1} \mid \sigma_{i-1} \in \Sigma_{i-1}$ & $W_i = W \cup \mathfrak{I}(\sigma_{i-1})$
                & $\omega_i \in$ CR-unify($\sigma_{i-1}\varphi_1\nu$, $\sigma_{i-1}\varphi_2\nu$, $W_i$)$\}$
        end
    return($\Sigma_i$)
    end map-unify
```

Figure 4-2: A Careful Description of the *map-unify* Procedure

We will use an inductive argument to show the protectiveness of *CR-unify*. Recall that the conditions comprising protectiveness of a set Σ of unifiers of t and s are:

$$\forall \sigma \in \Sigma \;\; \mathfrak{D}(\sigma) \subseteq V \;\&\; W - V \cap \mathfrak{I}(\sigma) = \emptyset$$
$$\& \; \mathfrak{D}(\sigma) \cap \mathfrak{I}(\sigma) = \emptyset$$

where $V = \mathcal{V}(t) \cup \mathcal{V}(s)$. The protectiveness of each step depends on W containing all variables in the input arguments, so it is important that recursive calls maintain this convention.

The applications of *Homog* to t and s in both *CR-unify* and *CR-variable-unify* of Figure 4-1 meet this requirement, since $\mathcal{V}(t) \cup \mathcal{V}(s) \subseteq W$. Calls to the sub-theory unification algorithm case 5 and case B explicitly add any new variables from $\hat{t}$ and $\hat{s}$ to W. The only non-trivial case is the invocation of *map-unify*, and here we know by construction of γ that all variables in $\mathfrak{D}(\gamma) \cup \mathfrak{I}(\gamma)$ are in either $\mathcal{V}(t)$, $\mathcal{V}(s)$, $\mathcal{V}(\hat{t})$, or $\mathcal{V}(\hat{s})$. Furthermore, by protectiveness of the sub-theory unification algorithm, $\mathfrak{D}\rho \subseteq \mathcal{V}(\hat{t}) \cup \mathcal{V}(\hat{s})$. By explicitly adding $\mathfrak{I}(\rho)$ as well as $\mathcal{V}(\hat{t})$ and $\mathcal{V}(\hat{s})$ to W, we are guaranteed to cover all variables in both γ and ρ. Note that the protected variables in calls to one invocation of *map-unify* do not contain variables generated from a previous invocation of *map-unify*.

Map-unify also maintains the convention of having all variables from the arguments in the protected set, since it explicitly places any new variables from one invocation into the protected set for the next invocation.

Given this assumption on W, we can see that *CR-unify* produces a protective set of unifiers. Cases 1, 2, 3, 4, A, and C are obvious. In cases 5 and B, if σ is a returned unifier, then $\mathcal{D}(\sigma) \subseteq \mathcal{V}(t) \cup \mathcal{V}(s)$ by the explicit restriction of Σ. The two properties on the range of σ, $(W-V) \cap \mathcal{I}(\sigma) = \emptyset$ and $\mathcal{D}(\sigma) \cap \mathcal{I}(\sigma) = \emptyset$ follow because σ is the composition of substitutions formed by recursive calls to *CR-unify*, and in each case W is a subset of the protected set in the recursive call.

Appendix B

Glossary of Terms

$\mathcal{A}$-assignment
a mapping from terms to elements of algebra $\mathcal{A}$.

algebra
A set of elements and a set of function on the elements.

carrier
The set of elements in an algebra.

clash
In classical unification, this is the problem that occurs when trying to unify terms with different head symbols.

congruence relation
An equivalence relation closed under the equality rule, $t_1 \sim s_1, \ldots, t_n \sim s_n \Rightarrow f(t_1,\ldots,t_n) \sim f(s_1,\ldots,s_n)$ for all $f \in F$ of arity n.

complete set of unifiers
A generating set for the set of all unifiers.

completeness
The property on unification algorithms that guarantees a complete set of unifiers is always found.

confined
A set of equations is confined if is contains no equations with a variable equal to a non-variable term.

consistency
The property on unification algorithms that guarantees all returned substitutions are unifiers.

constants
Function symbols of arity 0, denoted a, b, c, d, 0, or 1.

corresponding pair
The pair of terms with which *CR-unify* is invoked from within *map-unify*. The pair is formed by picking a variable in the domain of the two substitutions, applying each substitution to the variable, and then applying any previously accumulated substitution.

cycle
The problem that occurs in classical unification when unifying a variable with a term containing that variable. More generally in equational unification, this happens whenever the unification is an infinite term.

domain
The domain of a substitution, denoted $\mathfrak{D}(\sigma)$ is the set of all variables mapped to something other than themselves, i.e., $\mathfrak{D}(\sigma) = \{v \mid \sigma v \neq v\}$.

elementary substitution
Substitution with a domain of size 1 or 0.

elementary non-increasing

A fairly technical definition giving sufficient conditions on elementary substitutions such that the complexity ($\prec_{\mathcal{C}}$) of two terms is not increased by applying the substitution.

ground terms — The set of terms formable from only function symbols, i.e., no variables.

head — The leftmost symbol of a term.

homogeneous — A term is homogeneous with respect to a set of function symbols is the term contains no function symbols outside that set. A substitution is homogeneous if all terms in its range are homogeneous with respect some set of function symbols. (See *Homog*, *U-Homog*, and *U-HomogMap*.)

Homog — The homogenizing operation on terms, denoted (t), that replaces subterms with new variables.

instance — A term t is an instance of s is $t = \sigma s$ for some substitution σ. Similarly, a substitution, φ_1 is an instance of φ_2 if and only if there exists φ_3 such that $\varphi_1 = \varphi_3 \circ \varphi_2$. φ_2 is said to be more general than φ_2 in this case, and the partial order on substitutions is denoted $\varphi_1 \leq \varphi_1$.

instantiation — A rule of inference used in equational logic From $t \underset{E}{=} s$ deduce $\sigma t \underset{E}{=} \sigma s$.

map-unify — A procedure for finding unifiers of substitutions.

match — A substitution mapping a term to an instance of itself.

minimal complete set of unifiers

An set of substitutions that generates all unifiers and contains no redundant substitutions.

minimal partition — The smallest partition on a presentation of a theory that preserves disjointness of operators.

minimality — A property on unification algorithms that guarantees no returned substitution is an instance of another returned substitution.

more general modulo E

A partial order on substitutions, denoted $\overset{V}{\underset{E}{\leq}}$, that is similar to $\leq$ except $\underset{E}{=}$ is used in place of term equality.

most general unifier

In classical unification this is the unique unifier of which all other unifiers are instances. In E-unification there may be a set of most general unifiers.

non-confining — An equation is non-confining if it is of the form $v = t$ or $t = v$, where t a non-variable term.

non-increasing — A substitution formed of only elementary non-increasing factors.

occurrence
A of integers, o, denoting node within a term. For example, $f(t_1,\ldots,t_n)/i.o = t_i/o$, where t/o denotes the term at occurrence o in t.

parent
A parent of t in s is an operator in s having t as an argument. The set of all parents of t in s is written $Parents(t,s)$.

parent set
The equivalence of a parent. The set of all parents sets of t in s is written $ParSets(t,s)$.

partitioned presentation
A set of sets of axioms with pairwise disjoint operators.

presentation
A set of axioms for an equational theory.

preserving substitution
The substitution mapping $\hat{t}$ to t, which is just the match of $\hat{t}$ by t.

proper occurrence
An occurrence other than the empty occurrence, ε.

protection
A property on unification algorithms limiting the variables that can appear in an answer, i.e., all answers must be protective unifiers.

protective unifier
A unifier, σ, of t and s is protective if is domain and range contain disjoint sets of variables, the domain is a subset of the variables in t and s, and any variables in the range of σ than do not occur in t or s are new variables.

quotient algebra
The algebra formed from another algebra by taking equivalence classes of elements of the second as elements of the first. For example, $\mathcal{T}/\underset{E}{=}$ denotes the term algebra modulo an equational theory E*.

range
The range of a substitution is the set of terms to which some element of the domain is mapped.

regular
An equation is regular if the right and left sides contain identical sets of variables. A set of equations is regular if every element is regular.

relevant function symbols
The set of function symbols constrained by a sub-theory. In most cases this is the set of function symbols in a presentation of the theory given in π; in the case of the empty theory, this is the set of symbols that do not appear in any axioms.

significant occurrence
An occurrence, o is significant if the occurrence just above it does not have the same operator as o, where same means $\underset{\pi}{=}$. All variable occurrences as well as the empty occurrence are significant.

significant subterm
A subterm is significant if it occurs at a significant occurrence.

strict occurrence
A non-variable occurrence.

strictly consistent A theory is strictly consistent if and only if $x =_E y$ is not in the theory. A theory that is not strictly consistent contains all equations.

strict theory A theory is strict if for any set of unificands having a unifier in common, the transitive closure of $\prec_{\mathfrak{K}}$ is a strict ordering on the set.

strongly complete theory
A theory is strongly complete if for any pair of terms, a variable, x, and non-variable, t, if x and t are unifiable then there is complete set of unifiers such that every substitution in the set has a domain of $\{x\}$.

sub-theory If E_i is an element of π, then E_i^* is a sub-theory of E^*.

substitution A mapping from variables to terms extended to a mapping from terms to terms.

subterm ordering Denoted $t \preceq s$, this partial ordering on terms holds when t is a subterm of s.

theory A set of equations that is closed under rules of inference.

U-Homog A homogenizing function on terms that replaces subterms with elements of **U** rather than new variables as in *Homog*. $\bar{t}$ is used when the value of F is clear from context.

U-HomogMap A homogenizing function on substitutions, denoted $\bar{\sigma}$, that homogenizes each term in the range and maps some elements of **U** to others.

unifier In the classical case, a substitution, σ, is a unifier of two terms t and s if and only if $\sigma t = \sigma s$. In the more general equational unification, term equality if replace by equality in an equational theory.

CR-unify An algorithm for unifying in confined regular theories.

uninterpreted Function symbols that do not appear in a presentation of a theory and uninterpreted.

universal E-preserving substitution
The substitution, μ, that maps the homogeneous form of a term under *U-Homog* to a term within E of the original.

variables The universe of variables is denoted **V** while individual variables are denoted by either u, v, w, x, y, or z.

Appendix C

Special Symbols

$\mathcal{A}$ An algebra.

$\prec_{\mathfrak{C}}$ The noetherian ordering on inputs to *CR-unify* used to show termination. $\prec_{\mathfrak{C}}$ is the lexicographic extension of the cardinalities of ν and τ.

$\mathfrak{D}$ The domain of a substitution, i.e., $\mathfrak{D}(\sigma) = \{v \mid \sigma v \neq v\}$.

$\leq$ Used to denote the ordering on substitutions, $\varphi_1 \leq \varphi_2$ if and only if there exists φ_3 such that $\varphi_3 \circ \varphi_1 = \varphi_2$.

$\underset{E}{=}$ The congruence relation on terms defined by an equational theory of E. Also used to denote the congruence relation extended to substitutions.

E* The equational theory presented by the set of axioms, E. I.e., E denotes any set of equations whereas E* denotes a closed set.

Eq $\tilde{M}$ The set of equations valid in all models of the class $\tilde{M}$.

F The universe of function symbols, i.e., the signature of the entire theory E*.

$\mathfrak{F}(t)$ The set of function symbols in *t*

γ A substitution, usually used to denote the preserving substitution of a term and its homogeneous form or the combined preserving substitution for a pair of terms.

G The set of all ground terms formable from F.

ι The empty substitution.

t.head The function symbol at the head (or root) of *t*.

ε The empty occurrence, i.e., the empty string. For any term $t/\varepsilon = t$.

$\mathfrak{I}$ The set of all variables in the range of a substitution, i.e., $\mathfrak{I}(\sigma) = \{v \mid v \in \mathfrak{V}(t)$ for $t \in \mathfrak{R}(\sigma)\}$.

$\prec_{\mathfrak{K}}$ An ordering on pairs of terms defined in [Kirchner 85] that is used in defining a strict equational theory. One pair is less than another if a term in the first pair is a variable, and that variable occurs in a non-variable term in the second pair.

$\mathcal{M}(E)$ The set of all models of E.

μ	The universal E-preserving substitution.
ν	An $\mathcal{A}$-assignment, or interpretation, mapping terms to objects in an algebra.
n	The null operator, used to denote the parent of the a term in itself.
$\nu(t, s)$	The set of variables in t and s that occur under more than one equivalence class of parent operators. The cardinality of ν is denoted by $\boldsymbol{\nu}$. (ν is in no way related to the use of ν as an $\mathcal{A}$-assignment.)
U	The set of special variables used to denoted congruence classes of terms.
$\mathcal{O}(t)$	The set of occurrences in t
ω	A substitution, used in this thesis to denote factors of a unifier as it is being built in *CR-unify*.
Parents(t, s)	The set of parent operators of t in s.
ParSets(t, s)	The equivalence classes of parent operator of t in s.
π	The partitioned presentation, usually assumed to present the theory E*.
$\underset{\pi}{=}$	The equivalence relation on function symbols defined by the partitioned presentation, π.
φ	A substitution.
ρ	A substitution, used in this thesis to denote a sub-theory unifier of two homogeneous terms.
σ	A substitution, used the *CR-unify* to denote a unifier or, when subscripted, a partially formed unifier.
$\mathfrak{R}$	The terms in the range of a substitution, i.e., $\mathfrak{R}(\sigma) = \{\sigma v \mid v \in \mathfrak{D}(\sigma)\}$.
S	The set of all possible substitutions.
$\tau(t, s)$	The set of significant subterms of t and s. The cardinality of τ is denoted by $\boldsymbol{\tau}$.
T = T(F, V)	The set of all terms formable from **F** and **V**.
$\mathcal{T}/\underset{E}{=}$	The quotient algebra of the term algebra modulo and equation theory congruence relation on terms.
V	The universe of variables.
$\mathcal{V}(t)$	The set of variables in t.
$\restriction_V$	Restricts the domain of a substitution: $\sigma\restriction_V = \{v \leftarrow \sigma v \mid v \in V\}$. Also used to restrict sets of substitutions: $\Sigma\restriction_V = \{\sigma\restriction_V \mid \sigma \in \Sigma\}$.

$\varphi_1 \circ \varphi_2$ Functional composition. For substitutions φ_1 and φ_2, $\varphi_1 \circ \varphi_2 t = \varphi_1(\varphi_2 t)$ for any term t.

$t.head$ The function symbols at the head of a term t.

$\models$ Validity.

U_E Set of all E-unifiers of two terms.

CSU_E Complete set of unifiers.

μCSU_E Minimal and complete set of unifiers

$[f]$ The equivalence class of function symbols (defined by π) that contains f.

$\hat{t}$ The homogeneous form of t. *Homog*(t, [$t.head$]). Homogenization is done with respect to the set of relevant function symbols for some sub-theory of E* such that the head of t is in the set. I.e., it is the maximum homogeneous term at the top of t where new variables take the place of subterms outside the homogeneous part.

$\bar{t}$ The homogeneous form of t which is similar to $\hat{t}$ except subterms are replaced with elements of the special set **U** rather than new variables.

$\preceq$ The subterm ordering on terms. I.e., $t \preceq s$ if and only if t is a subterm of s. $t \prec s$ may be used if t is a proper subterm of s.

$\preceq_{\mathcal{S}}$ The significant subterm ordering, i.e., $t \preceq_{\mathcal{S}} s$ if and only if t is a subterm of s and t is significant in s. If t is also know to be proper in s, $t \prec_{\mathcal{S}} s$ may be used.

References

[Arnborg 85] A. Arnborg and E. Tidén, "Unification Problems with One-Sided Distributivity," *Proc. of the Conference on Rewriting Techniques and Applications, Dijon, France, 1985*, Springer-Verlag, 1985. to appear

[Baxter 73] L. D. Baxter, "An Efficient Unification Algorithm," Technical Report CS-73-23, Dept. of Applied Analysis and Computer Science, Univ. of Waterloo, Waterloo, Ontario, 1973.

[Benanav 85] D. Benanav, D. Kapur, and P. Narendran, "Complexity of Matching Problems," *Proc. of the Conference on Rewriting Techniques and Applications, Dijon, France, 1985*, Springer-Verlag, 1985. to appear

[Birkhoff 35] G. Birkhoff, "On the Structure of Abstract Algebras," *Proc. Cambridge Phil. Soc., Vol. 31*, 1935, pp. 433-454.

[Chandra 84] A. Chandra and P. Kanellakis, private communication, 1984.

[Choppy 85] C. Choppy and C. Johnen, "Retrireve: Proving Petri Net Properties with Rewriting Systems," *Proc. of the Conference on Rewriting Techniques and Applications, Dijon, France, 1985*, Springer-Verlag, 1985. to appear

[Clocksin 81] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, Springer-Verlag Berlin Heidelberg New York, 1981.

[Cohn 65] P. M. Cohn, *Universal Algebra*, Harper & Row, 1965.

[Corbin 83] J. Corbin and M. Bidoit, "A Rehabilitation of Robinson's Unification Algorithm," R. E. A. Mason (Ed.), *Proc. 9th World Computer Congress*, IFIP '83, North-Holland, September 1983, pp. 909-914.

[Cosmadakis 85] S. Cosmadakis and P. Kanellakis, "Two Applications of Equational Theories to Data Base Theory," *Proc. of the Conference on Rewriting Techniques and Applications, Dijon, France, 1985*, Springer-Verlag, 1985. to appear

[Davis 73] M. Davis, "Hilbert's Tenth Problem is Unsolvable," *American Mathematical Monthly* 80(3):233-269, 1973.

[Dershowitz 83a] N. Dershowitz, "Computing with Rewrite Systems," Technical Report ATR-83(8478)-1, Aerospace Corp., El Segundo, CA, January 1983.

[Dershowitz 83b] N. Dershowitz, N. A. Josephson, J. Hsiang, and D. Plaisted, "Associative-Commutative Rewriting," *8th IJCAI, Karlsruhe, West Germany*, 1983.

[Dwork 84] C. Dwork, P. C. Kanellakis, and J. C. Mitchell, "On the Sequential Nature of Unification," *Journal of Logic Programming* 1, pp. 35-50, June 1984.

[Fages 83a] F. Fages and G. Huet, "Complete Sets of Unifiers and Matchers in Equational Theories," *Trees in Algebra and Programming*, CAAP '83, Proceedings of the 8th Colloquium, L'Aquila, Italy, Lecture Notes in Computer Science, Springer-Verlag, March 1983, pp. 205-220.

[Fages 83b] F. Fages, "Formes Canoniques dans les Algebres Booleennes, et Application a la Demonstration Automatique en Logique de Premier Ordre," Ph.D. Thesis, L'Universite Pierre et Marie Curie, Paris VI, June 1983.

[Fages 84] F. Fages, "Associative-Commutative Unification," *Proc. 7th CADE, Napa Valley*, Springer-Verlag, 1984, pp. 194-208.

[Fages 85] F. Fages, private communication, 1985.

[Fay 79] M. Fay, "First-order Unification in an Equational Theory," *Proc. 4th Workshop on Automated Deduction*, Austin, TX, February 1979, pp. 161-167.

[Filgueiras 82] M. Filgueiras, "A Prolog Interpreter Working with Infinite Terms," Technical Report FCT/UNL - 20/82, Faculdade de Ciencias e Tecnologia, November 1982. Quinta da Torre, 2825 Monte da Caparica, Portugal

[Forgaard 84a] R. Forgaard and J. V. Guttag, "REVE: A Term Rewriting System Generator with Failure-Resistant Knuth-Bendix," *Proc. of an NSF Workshop on the Rewrite Rule Laboratory, Sept. 6-9, 1983*, General Electric Corporate Research and Development Report No. 84GEN008, Schenectady, NY, April 1984, pp. 5-31.

[Forgaard 84b] R. Forgaard, "A Program for Generating and Analyzing Term Rewriting Systems," Master's Thesis, MIT Lab. for Computer Science, 1984.

[Fribourg 84] L. Fribourg, "Oriented Equational Clauses as a Programming Language," *ICALP*, 1984.

[Garey 79] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., San Francisco, 1979.

[Goguen 79] J. A. Goguen and J. J. Tardo, "An Introduction to OBJ: A Language for Writing and Testing Formal Algebraic Program Specifications," *Proc. Specification of Reliable Software*, Institute of Electrical and Electronics Engineers, April 1979, pp. 170-189.

[Goguen 80] J. A. Goguen, "How to Prove Algebraic Inductive Hypotheses Without Induction, With Applications to the Correctness of Data Type Implementation," *Lecture Notes in Computer Science, Vol. 87: Proc. 5th Conf. on Automated Deduction*, Les Arcs, France, Springer-Verlag, New York, July 1980, pp. 356-373.

[Grätzer 78] G. Grätzer, *Universal Algebra*, Springer-Verlag, 1969, 1978.

[Guttag 83] J. V. Guttag and J. J. Horning, "Preliminary Report on The Larch Shared Language," Technical Report TR-307, MIT Lab. for Computer Science, October 1983.

[Hsiang 82] J. Hsiang, "Topics in Automated Theorem Proving and Program Generation," Ph.D. Thesis, Univ. of Illinois, Urbana-Champaign, November 1982.

[Huet 80a] G. Huet and D. C. Oppen, "Equations and Rewrite Rules: A Survey," in R. Book (Ed.), *Formal Language Theory: Perspectives and Open Problems*, Academic Press, New York, 1980, pp. 349-405.

[Huet 80b] G. Huet, "Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems," *Journal of the ACM* **27**(4):797-821, October 1980.

[Huet 82] G. Huet and J. M. Hullot, "Proofs by Induction in Equational Theories with Constructors," *Journal of the ACM* **25**, pp. 239-266, 1982.

[Hullot 80] J. M. Hullot, "Canonical Forms and Unification," *Lecture Notes in Computer Science, Vol. 87: Proc. 5th Conf. on Automated Deduction*, Les Arcs, France, Springer-Verlag, New York, July 1980, pp. 318-334.

[Jeanrond 80] J. Jeanrond, "Deciding Unique Termination of Permutative Rewrite Systems: Choose your Term Algebra Carefully," W. Bibel and R. Kowalski (Eds.), *Proc. 5th CADE, Napa Valley*, Springer-Verlag, Lecture Notes in Computer Science, Volume 87, 1980, pp. 194-208.

[Jouannaud 83] J. P. Jouannaud, C. Kirchner, and H. Kirchner, "Incremental Construction of Unification Algorithms in Equational Theories," *Proc. 10th EATCS Intl. Colloq. on Automata, Languages, and Programming*, Barcelona, 1983, pp. 361-373.

[Jouannaud 84] J. P. Jouannaud and H. Kirchner. "Completion of a Set of Rules Modulo a Set of Equations." Technical Note. SRI Intl. Computer Science Laboratory, Menlo Park, CA, April 1984.

[Kandri-Rody 85] A. Kandri-Rody. D. Kapur. and P. Narendran. "An Ideal-Theoretic Approach to Word Problems and Unification Problems over Finitely Presented Commutative Algebras," *Proc. of the Conference on Rewriting Techniques and Applications, Dijon, France, 1985*, Springer-Verlag, 1985. to appear

[Kapur 84] D. Kapur and D. R. Musser, "Proof by Consistency," *Proc. of an NSF Workshop on the Rewrite Rule Laboratory. Sept. 6-9, 1983*, General Electric Corporate Research and Development Report No. 84GEN008. Schenectady, NY, April 1984, pp. 245-267.

[Kapur 85] D. Kapur, private communication, 1985.

[Kirchner 81] C. Kirchner and H. Kirchner, "Solving Equations in the Signed Trees Theory," Technical Report 81-R-056. Centre de Recherche en Informatique de Nancy, UER de Mathematiques. Universite de Nancy I, 54037 Nancy Cedex, 1981.

[Kirchner 84a] C. Kirchner, "A New Equational Unification Method: A Generalisation of Martelli-Montanari's Algorithm," *CADE*, 1984.

[Kirchner 84b] C. Kirchner and H. Kirchner, private communication, 1984.

[Kirchner 85] C. Kirchner and H. Kirchner, "Methodes et Outils de Conception Systmatique D'Algorithmes D'Unification dans les Theories Equationalles," Ph.D. Thesis, Centre de Recherche en Informatique de Nancy, UER de Mathematiques, Universite de Nancy I, 54037 Nancy Cedex, June 1985.

[Knuth 70] D. E. Knuth and P. B. Bendix, "Simple Word Problems in Universal Algebras," in J. Leech (Ed.), *Computational Problems in Abstract Algebra*, Pergamon, Oxford, 1970, pp. 263-297.

[Kowalski 74] R. A. Kowalski, "Predicate Logic as a Programming Language," *Proc. IFIP-74 Congress*, North-Holland, 1974, pp. 569-574.

[Kownacki 84] R. W. Kownacki, "Semantic Checking of Formal Specifications," Master's Thesis, MIT Lab. for Computer Science, June 1984.

[Lankford 84] D. Lankford, G. Butler, and B. Brady, "Abelian Group Unification Algorithms for Elementary Terms," *Proc. of an NSF Workshop on the Rewrite Rule Laboratory, Sept. 6-9, 1983*, General Electric Corporate Research and Development Report No. 84GEN008, Schenectady, NY, April 1984, pp. 301-318.

[Lescanne 83] P. Lescanne, "Computer Experiments with the REVE Term Rewriting System Generator," *Proc. 10th ACM Symp. on Principles of Programming Languages*, Austin, TX, January 1983, pp. 99-108.

[Liskov 81] B. Liskov, R. Atkinson, T. Bloom, E. Moss, J. C. Schaffert, R. Scheifler, A. Snyder, *Lecture Notes in Computer Science, Vol. 114: CLU Reference Manual*, Springer-Verlag, New York, 1981.

[Livesey 76] M. Livesey and J. Siekmann, "Unification of A + C Terms (Bags) and A + C + I Terms (Sets)," Technical Report Interner Bericht Nr. 3/76, Institut für Informatik I, Universität Karlsruhe, 1976.

[Makanin 77] G. S. Makanin, "The Problem of Solvability of Equations in a Free Semigroup," *Soviet Akad., Nauk SSSR, Tom 233*, 1977.

[Martelli 82] A. Martelli and U. Montanari, "An Efficient Unification Algorithm," *ACM Transactions on Programming Languages and Systems* 4(2):258-282, April 1982.

[Milner 78] R. Milner, "A Theory of Polymorphism in Programming," *Journal of Computer and System Sciences* 17, pp. 348-375, 1978.

[Mitchell 84] J. Mitchell, "Coercion and Type Inference (Summary)," *Principal of Programming Languages*, 1984.

[Nelson 79] G. Nelson and D. Oppen, "Simplification by Cooperating Decision Procedures," *ACM Transactions on Programming Languages and Systems* 1(2):245-257, Oct 1979.

[Paterson 78] M. S. Paterson and M. N. Wegman, "Linear Unification," *Journal of Computer and System Sciences* 16, pp. 158-167, 1978.

[Peterson 81] G. E. Peterson and M. E. Stickel, "Complete Sets of Reductions for Some Equational Theories," *Journal of the ACM* 28(2):233-264, April 1981.

[Plotkin 72] G. D. Plotkin, "Building-in Equational Theories," in *Machine Intelligence, Vol. 7*, Halsted Press, 1972, pp. 73-90.

[Raulefs 78] P. Raulefs and J. Siekmann, "Unification of Idempotent Functions," Technical Report D-7500 Karlsruhe 1, Institut für Informatik I, Universität Karlsruhe, 1978.

[Rety 85] P. Réty, C. Kirchner, H. Kirchner, and P. Lescanne, "Narrower, a New Algorithm for Unification and its Application to Logic Programming," *Proc. of the Conference on Rewriting Techniques and Applications, Dijon, France, 1985*, Springer-Verlag, 1985. to appear

[Robinson 65] J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," *Journal of the ACM* 12(1):23-41, January 1965.

[Robinson 71] J. A. Robinson, "Computational Logic: The Unification Computation," in B. Meltzer and D. Michie (Eds.), *Machine Intelligence, Vol. 6*, Edinburgh Univ. Press, Edinburgh, Scotland, 1971, pp. 63-72.

[Shostak 84] R. E. Shostak, "Deciding Combinations of Theories," *Journal of the ACM* 31(1):1-12, January 1984.

[Siekmann 79] J. Siekmann, "Unification of Commutative Terms," *Proceedings of the Conference on Symbolic and Algebraic Manipulation*, 1979.

[Siekmann 84] J. Siekmann, "Universal Unification," *Proc. 7th CADE, NAPA Valley, CA*, Springer-Verlag, 1984, pp. 1-42.

[Slagle 74] J. R. Slagle, "Automated Theorem-Proving for Theories with Simplifiers, Commmutativity and Associativity," *JACM* 21(4):622-642, October 1974.

[Stickel 81] M. E. Stickel, "A Unification Algorithm for Associative-Commutative Theories," *Journal of the ACM* 28(3):423-434, July 1981. Preliminary version in *Proc. 4th Intl. Joint Conf. on Artificial Intelligence*, Tbilissi, 1975.

[Szabo 78] P. Szabò "The Undecidability of the D+A-Unification Problem," Technical Report, Universität Karlsruhe, Institut für Informatik I, 1978.

[Tiden 85] E. Tidén, private communication, 1985.

[Vogel 78] E. Vogel, "Morphismenunifikation," Technical Report, Universität Karlsruhe, 1978. Diplomarbeit

[Yelick 85] K. Yelick, "Combining Unification Algorithms for Confined Regular Equational Theories," *Proc. of the Conference on Rewriting Techniques and Applications, Dijon, France, 1985*, Springer-Verlag, 1985. to appear

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER MIT/LCS/TR-344 / 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A Generalized Approach to Equational Unification	5. TYPE OF REPORT & PERIOD COVERED B.S. and M.S. Thesis August 1985
	6. PERFORMING ORG. REPORT NUMBER MIT/LCS/TR-344
7. AUTHOR*(s)* Katherine Anne Yelick	8. CONTRACT OR GRANT NUMBER*(s)* DARPA/DOD N00014-83-K-0125
9. PERFORMING ORGANIZATION NAME AND ADDRESS MIT Laboratory for Computer Science 545 Technology Square Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS DARPA/DOD 1400 Wilson Blvd. Arlington, VA 22209	12. REPORT DATE August 1985 13. NUMBER OF PAGES 102
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* ONR/Department of the Navy Information Systems Program Arlington, VA 22217	15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for Public Release, distribution is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

Unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Unification, equational unification, equational theories, confined regular theory, unification algorithm, automatic theorem proving, term rewriting, resolution.

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

Given a set of equational axioms and two terms containing function symbols and variables, the equational unification problem is to find a uniform replacement of terms for the variables that makes the terms provably equal from the axioms. In the variable-only case, the two terms contain only variables and function symbols from the axioms. In the general case, the terms may contain symbols not appearing in the axioms, there may be more than one instance of a set of axioms, and there may be more than one set of axioms.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

This thesis presents a method for combining equational unification algorithms to handle the terms with "mixed" sets of function symbols. For example, given one algorithm for unifying associative-commutative operators, and another for unifying commutative operators, our algorithm provides a method for unifying terms containing both kinds of operators. It is based on a general strategy for decomposing terms and combining unifiers. We restrict our attention to sets of axioms whose function symbols are pairwise disjoint.

A simplifying assumption is that we are working only with confined regular equational theories, a class of theories defined in this thesis. We present a unification algorithm that solves the general case unification problem for any combination of these theories, given variable-only case algorithms for the theories. The algorithm is proven totally correct. The termination proof is a generalization of Fages' proof of termination for associative-commutative unification.

Our algorithm has been implemented as part of a larger system for generating and reasoning about equational term rewriting systems.