1-1-1990

Term Reduction Using Directed Congruence Closure

L Paul Chew
Dartmouth College

TERM REDUCTION USING DIRECTED CONGRUENCE CLOSURE

L. Paul Chew

Technical Report PCS-TR90-149

Term Reduction using Directed Congruence Closure*

L. Paul Chew**
Dartmouth College
Hanover, NH 03755

Abstract

Many problems in computer science can be described in terms of reduction rules that tell how to transform terms. Problems that can be handled in this way include interpreting programs, implementing abstract data types, and proving certain kinds of theorems. A term is said to have a normal form if it can be transformed, using the reduction rules, into a term to which no further reduction rules apply. In this paper, we extend the Congruence Closure Algorithm, an algorithm for finding the consequences of a finite set of equations, to develop Directed Congruence Closure, a technique for finding the normal form of a term provided the reduction rules satisfy the conditions for a regular term rewriting system. This technique is particularly efficient because it inherits, from the Congruence Closure Algorithm, the ability to remember all objects that have already been proved equivalent.

* An early version of this work appeared as "An improved algorithm for computing with equations" in the Proceedings of the 21st Annual Symposium on the Foundations of Computer Science (1980), 108-117. Portions of this work also appear in "Normal forms in term rewriting systems", Ph.D. Thesis, Purdue University (December 1981).

** This research was supported in part by NSF grants MCS-7801812 and MCS-8204821.

1 Introduction

An equational axiom is a pair of terms written in the form (A=B) with the meaning that an occurrence of the term A can be replaced by the term B, or vice versa. If **K** is a set of equational axioms and C and D are terms then we write $C=_{\mathbf{K}}D$ iff C and D are the same except for replacements according to the axioms in **K**. These axioms may be considered equations in the sense that the following property holds: $C=_{\mathbf{K}}D$ iff C=D may be proved as a consequence of the axioms (equations) in **K** using the reflexive, symmetric, transitive, and substitution laws for equality.

Many important problems in computer science can be naturally reduced to the following problem: given a set **K** of equational axioms, a set $\mathbf{\Gamma}$ called the set of simple terms, and some term A, find a term B such that $B\in\mathbf{\Gamma}$ and $A=_{\mathbf{K}}B$. For example:

1. (Hoffmann and O'Donnell [HO82b]) Programming languages can be defined by sets of equational axioms. The process of interpreting a program is equivalent to the process of simplifying the program according to the rules given by the axioms. This is particularly appropriate for descriptive languages such as LISP and Lucid.
2. Equational axioms may be used directly as a programming language. The semantics of such a programming language are particularly simple and intuitive. Hoffmann and O'Donnell [HO82b, O'D85] have implemented an interpreter for an equational language.
3. (Guttag, Horowitz, and Musser [GHM78]) The properties of an abstract data type can be defined by a set of equational axioms. These axioms can be used to automatically produce an implementation of the data type, with the advantage that this implementation is guaranteed to satisfy the axioms.

4. Given a set of axioms, theorems of the form A=B can be proved by showing that both A and B are equivalent to the same simple term. Such theorem provers are discussed by Knuth and Bendix [KB70].

Each of these problems (particularly problems 1, 2, and 3) can fairly naturally use sets of equational axioms in which a direction is assigned to each axiom. In this paper we do not concern ourselves with how these directions are assigned (this problem is discussed by Knuth and Bendix [KB70]). We assume, instead, that we are given a set of axioms in which each axiom has a direction. We write such axioms in the form $A \Longrightarrow B$ with the meaning that an occurence of the term A may be replaced by the term B.

Let **K** be a set of axioms. Suppose $A \Longrightarrow B$ is an axiom in **K** and suppose C is a term that contains an occurrence of A as a subterm. If C' is the result of replacing that occurrence of A by B then we say C **K**-reduces to C' or $C \rightarrow_{\mathbf{K}} C'$. We use $=_{\mathbf{K}}$ as the smallest equivalence relation containing $\rightarrow_{\mathbf{K}}$. A term is said to be in **K** normal form iff it cannot be **K**-reduced. If A reduces to B (in zero or more steps) and B is in normal form then B is a normal form for A.

We represent an infinite set of axioms by using a finite set of axiom schemata (axioms with variables). We create different axiom instances by substituting terms for the variables in an axiom schema. A set of axiom schemata is variously called a term rewriting system [HO80], a term reduction system [O'D85], or a schematic subtree replacement system [Ros73].

An important question for term rewriting systems is whether normal forms are unique (i.e., whether each term has at most one normal form). A method often used to prove this property is to first prove a stronger property; the confluence or Church-Rosser property. A set Δ of axiom

schemata has the confluence property iff whenever $A=_{\Delta}B$, there is a term C such that both A and B reduce (using the rules of Δ) in zero or more steps to C. The confluence property ensures that each term has at most one normal form.

O'Donnell [O'D77] has shown that the following conditions are sufficient to prove the confluence property for a set Δ of axiom schemata.

1. No repeated variables (i.e., no variable is repeated on the left side of an axiom schema).
2. The schemata of Δ must be nonoverlapping (i.e., the axiom schemata do not interfere with each other; this will be fully defined in section 2).
3. The set of schemata must be unequivocal (i.e., if $A \Longrightarrow B$ and $A \Longrightarrow C$ are axiom instances then B and C must be identical terms).

Following Klop [Klo80], we call a term rewriting system that satisfies these restrictions regular.

In this paper, we develop Directed Congruence Closure, a method that can be used to find solutions to the following two problems for regular term rewriting systems.

1. The Normal Form Problem (NF). Given a set Δ of axiom schemata and a term A, find a term B such that B is a Δ normal form for A.
2. The Equivalence of Terms Problem (ET). Given a set Δ of axiom schemata and terms A and B, determine if $A=_{\Delta}B$.

It is easily seen that the halting problem is reducible to the problem of determining whether a term has a normal form (using axiom schemata that simulate a Turing Machine). In fact, a regular term rewriting system using only unary symbols can simulate a type 0 grammar. Since we do not wish to restrict the possible sets of axiom schemata in such a way that simulation of a Turing Machine is impossible, the best an NF-solving

program can do is to produce a normal form for a term if one exists; if a normal form does not exist the program could run forever.

The halting problem is also seen to be reducible to ET; thus any program can solve at best a restricted version of ET. The Congruence Closure Algorithm (to be further discussed) completely solves ET for the special case in which only a finite set of axiom instances is allowed. Another approach is that used by Knuth and Bendix [KB70] and others (e.g., [HO80, Hue80, GB85]) in which the allowable sets of axiom schemata are such that infinite computations are prohibited. Our approach is similar to the approach taken for algorithms solving NF: our algorithm for ET simply will not answer some of the time. If A is equivalent to B then the algorithm will answer "yes", but if A is not equivalent to B the algorithm may answer "no" or it may run forever.

The method we develop, Directed Congruence Closure, is a combination of two algorithms: (1) Straight Reduction, an algorithm that solves NF for regular term rewriting systems, and (2) the Congruence Closure Algorithm. These algorithms are discussed in sections 3 and 4.

2 Preliminaries

Let Σ be a finite alphabet and let $\rho:\Sigma\rightarrow\mathbf{N}$ be a rank function.

Definition 2.1.

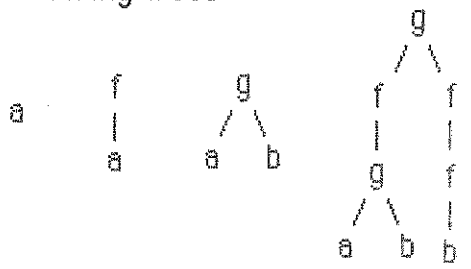
1. If $\rho a=n$ and $A_1,\ldots,A_n$ are Σ terms then $a(A_1,\ldots,A_n)$ is a Σ term.

2. Nothing is a Σ term except as required by rule 1.

Note that if $\rho a=0$ then "a()" is a Σ term. For convenience, we write such a term as "a". When the alphabet Σ is clear from context we will often refer to Σ terms as terms.

Example. Let $\Sigma=\{a, b, f, g\}$ and suppose $\rho a=\rho b=0$, $\rho f=1$ and $\rho g=2$. Then a, f(a), g(a,b) and g(f(g(a,b)),f(f(b))) are all Σ terms.

It is often convenient to consider Σ terms as labeled trees. The examples above correspond to the following trees:



A location within a Σ term can be thought of as a node within the corresponding tree. The subterm at a particular location is the subtree (subterm) rooted at that location. Two locations within a term are said to be independent iff neither lies within the subterm corresponding to the other. If x and y are independent locations then we write $x\perp y$. If location x lies within the subterm at y we write $x<y$.

Definition 2.2. An axiom is an ordered pair of Σ terms written $A\Rightarrow B$.

Our terminology is basically that of [O'D77]. Let $\mathbf{K}$ be a set of axioms. If $A\Rightarrow B$ is a member of $\mathbf{K}$ we write $A\Rightarrow_{\mathbf{K}} B$ and we say A reduces at the root to B. We write $A\rightarrow_{\mathbf{K}} B$ if there is an axiom $A'\Rightarrow B'$ in $\mathbf{K}$ such that A' is a

subterm of A, and B is the same as A, but with a single occurrence of the subterm A' replaced by B'. The reduction is said to occur at the location of the term that is replaced, in this case the location of A'. A location within a term at which a reduction can occur is called a redex. If A' is a proper subterm of A we write $A \langle nr \rangle\!\rightarrow_{\mathbf{K}} B$ (nr stands for nonroot, a term that becomes clear when A is thought of as a tree). If $A \rightarrow_{\mathbf{K}} B$ we say A **K**-reduces in one step to B, and that A can be **K**-reduced.

We use $\langle * \rangle\!\rightarrow$ as the reflexive, transitive closure of $\rightarrow$ and if $A \langle * \rangle\!\rightarrow B$ we say A reduces to B. We use $=_{\mathbf{K}}$ as the least equivalence relation containing $\rightarrow_{\mathbf{K}}$; this relation is called the congruence closure of **K**.

A Σ term B is in **K** normal form iff B cannot be **K**-reduced. If $A \langle * \rangle\!\rightarrow_{\mathbf{K}} B$ and B is in **K** normal form then B is a **K** normal form for A. When the set **K** is clear from context it will often be left off.

Let V be a set of symbols called variables such that Σ and V are disjoint and let $\rho X=0$ for all $X \in V$. We use X, Y, and Z to represent variables. A Σ term A is an instance of the $\Sigma \cup V$ term B if we can substitute Σ terms for the variables in B to produce A. If a single variable appears more than once in a $\Sigma \cup V$ term then the same Σ term must be substituted for each occurrence of that variable.

Definition 2.3. An axiom schema is an ordered pair of $\Sigma \cup V$ terms written $C \Longrightarrow D$, subject to the restriction that variables appearing on the right must also appear on the left.

An axiom $A \Longrightarrow B$ is an axiom instance of the axiom schema $C \Longrightarrow D$ if we can substitute Σ terms for the variables in $C \Longrightarrow D$ to produce $A \Longrightarrow B$. Again, the same term must be substituted for each occurrence of a particular variable. By substituting different Σ terms for the variables we can

produce many (usually infinitely many) different axiom instances from a single axiom schema.

If Δ is a set of axiom schemata then $I(\Delta)$ represents the set of all axiom instances of Δ. To avoid cluttering our notation we often use Δ in place of $I(\Delta)$ (e.g., we use $\rightarrow_{\Delta}$ instead of $\rightarrow_{I(\Delta)}$ and $=_{\Delta}$ instead of $=_{I(\Delta)}$).

We need to ensure that normal forms are unique. To do this we restrict the allowable sets of axiom schemata in order to ensure a stronger property, the confluence property.

Definition 2.6.

1. A set **K** of axioms has the Church-Rosser property iff $A =_{\mathbf{K}} B$ imples there exists a term C such that $A \xrightarrow{*}_{\mathbf{K}} C$ and $B \xrightarrow{*}_{\mathbf{K}} C$.
2. A set **K** of axioms has the confluence property iff $A \xrightarrow{*}_{\mathbf{K}} B$ and $A \xrightarrow{*}_{\mathbf{K}} C$ implies there exists a term D such that $B \xrightarrow{*}_{\mathbf{K}} D$ and $C \xrightarrow{*}_{\mathbf{K}} D$.

Since $\xrightarrow{*}_{\mathbf{K}}$ is transitive it follows by a simple induction argument that the Church-Rosser property and the confluence property are equivalent. It is trivial to show that the confluence property implies normal forms are unique.

To ensure the confluence property, three restrictions are placed on the allowable sets of axiom schemata. One restriction says that if a term can be reduced at the root then there should be only one possible result term.

Definition 2.7. (Rosen [Ros73]) A set **K** of axioms is unequivocal iff **K** is a partial function when considered as a map from Σ^t to Σ^t.

In other words, **K** is unequivocal iff whenever $A \Longrightarrow B$ and $A \Longrightarrow C$ are in **K** then B and C are the same term. We say a set of axiom schemata is unequivocal if the set of all axiom instances is unequivocal. In O'Donnell's [O'D77]

terminology an unequivocal set of axiom schemata is said to be consistent.

The other two restrictions prevent axiom instances from interfering with each other. In other words, if a term can be reduced at the root then nonroot reductions should not affect this redex. We list some examples of the sorts of interference we wish to avoid.

Example. Δ: $f(X,X) \Longrightarrow b$

$a \Longrightarrow b$

The first schema could be applied to the term f(a,a), but, if the second schema is applied to the lefthand a, we get f(b,a). The schemata interfere since the first schema no longer matches.

Δ: $f(g(X)) \Longrightarrow a$

$g(X) \Longrightarrow b$

The first schema could be applied to the term f(g(a)), but an application of the second schema produces f(b). The first schema no longer applies.

We need several definitions to more fully describe the necessary restrictions.

Definition 2.4. Let A and B be $\Sigma \cup V$ terms.

1. A and B are unifiable iff there is a Σ term C such that C is an instance of A and also an instance of B. If a variable X occurs in both A and B then the term substituted for X in A is not necessarily the same as the term substituted for X in B.
2. A overlaps B iff there is a proper subterm A' of A such that (1) A' is not a single variable and (2) A' and B are unifiable.
3. A set S of $\Sigma \cup V$ terms is nonoverlapping iff for all $A,B \in S$, A does not overlap B.
4. A set Δ of axiom schemata is nonoverlapping iff the set of left sides is nonoverlapping.

The definition of nonoverlapping given here is equivalent to the definition of nonoverlapping given by O'Donnell [O'D77] and is similar to the definition of nonambiguous given by Huet and Levy [HL79] (the difference is that we require a proper subterm in our definition of overlap).

At this point, we describe the restrictions needed to ensure the confluence property.

Defintion 2.9 (Klop [Klo80]). A set Δ of axiom schemata is regular iff

1. There are no schemata with repeated variables on the left.
2. Δ is unequivocal.
3. Δ is nonoverlapping.

A set of axiom schemata that satisfies the first condition, no repeated variables on the left is often said to be left-linear in the literature.

Theorem 2.10 (O'Donnell [O'D77]). If Δ is regular then $I(\Delta)$ has the confluence property.

Corollary 2.11. If Δ is regular then normal forms are unique.

Using the results of this section, we can state more exact versions of the motivating problems mentioned in the Introduction. Directed Congruence Closure is a method designed to efficiently solve the following two problems:

1. Finding a Normal Form (NF). Given a regular set Δ of axiom schemata and a Σ term A, find a Σ term B such that $A \overset{*}{\longrightarrow}_{\Delta} B$ and B is in Δ normal form. If such a term B does not exist the algorithm may run forever.
2. Deciding Equivalence of Terms (ET). Given a regular set Δ of axiom schemata and two Σ terms A and B, the algorithm answers "yes" if $A =_{\Delta} B$, but if $A \neq_{\Delta} B$ the algorithm may answer "no" or it may run forever.

Note that both of these problems can be solved by exhaustive search; we can generate all possible axiom instances one at a time, saving each

possible reduction result until we reach a normal form (or until we prove $A =_{\Delta} B$). Of course, this method horribly inefficient, although it works with any finite (or even recursively enumerable) set of axiom schemata, while the method we develop requires the set of schemata to be regular.

We develop Directed Congruence Closure (DCC), a combinition of two other algorithms (to be discussed in sections 3 and 4, respectively): (1) Straight Reduction (SR), an algorithm that solves NF, and (2) the Congruence Closure Algorithm (CC), an algorithm that solves ET in the case where $i(\Delta)$ is finite.

3 Straight Reduction

Straight Reduction is the method most people would choose by intuition in an attempt to find the normal form of some expression.

Straight Reduction Algorithm (SR):

input: Δ, a set of axiom schemata;
A, a term to be reduced to normal form;
begin
D := A;
while there is a subterm B of D that can be Δ-reduced **do**
Find C such that B⟹C is an instance of Δ;
Replace B by C in D;
end while;
Print D;
end.

Clearly, if this procedure halts then D is a normal form for A. By restricting Δ to be a regular set of axiom schemata, we ensure that there is no more than one possible normal form for A (Corollary 2.11). It is also important to use a good strategy for choosing reducible subterms of D. If a poor strategy is used then the algorithm may run forever even though A has a normal form.

Example. g(X)⟹b

a⟹f(a)

The term g(a) has normal form b, but by making poor choices we can get g(a)→g(f(a))→g(f(f(a)))→....

O'Donnell [O'D77] has given a strategy for choosing reducible subterms of D that guarantees Straight Reduction halts when a normal form exists. To discuss this strategy we define an outermost redex.

Definition 3.1 (O'Donnell [O'D77]). A redex x of a term A is said to be outermost iff for all other redexes y of A, either x⊥y or y<x.

Intuitively, a redex is outermost iff there are no other redexes between it

and the root.

O'Donnell has shown that for regular sets of axiom schemata, if a term has a normal form then that normal form can be found by using a strategy in which each outermost redex is eventually eliminated.

Definition 3.2 (O'Donnell [O'D77]). Let x be an outermost redex of A_0. A reduction sequence $A_0 \langle * \rangle\!\!\rightarrow A_1 \langle * \rangle\!\!\rightarrow \ldots$ is said to eliminate x iff there exists i such that either

1. A_i is reduced at x in the reduction from A_i to A_{i+1} or
2. x is not an outermost redex in A_i.

In other words, the outermost redex x is eliminated iff it is either used or covered up. In O'Donnell's terminology, a sequence of reductions in which every outermost redex is eliminated is said to be eventually outermost. Eventually outermost sequences of reductions are guaranteed to produce a normal form if one exists.

Theorem 3.3 (O'Donnell [O'D77]). Let $A_0 \langle * \rangle\!\!\rightarrow A_1 \langle * \rangle\!\!\rightarrow \ldots$ be an eventually outermost reduction sequence. If A_0 has a normal form B then there exists k such that for all $i \geq k$, $A_i = B$.

A problem with Straight Reduction is the amount of repeated effort that may be required. For example, if f(b) is a term that reduces in 10 steps to b then the term f(f(f(b))) could require 30 steps to reduce to b. This repeated effort could be avoided if the computer remembered that f(b) reduces to b after going through all the steps only once. In this example, it is easy to see that f(b) will appear several times, but in general, determining whether a term will appear more than once is undecidable. A way to solve this problem is to devise an algorithm that

efficiently saves all reduction information. We develop such an algorithm by combining Straight Reduction with Congruence Closure.

4 Congruence Closure

The Congruence Closure Algorithm is an efficient algorithm for discovering the consequences of a finite set of equations (note that these must be equations without variables). There are several different versions of the algorithm, all used to solve the same basic problem (variously called the uniform word problem for finitely generated algebras [Koz77], a variation on the common subexpression problem [DST80], and the decision problem for the quantifier free theory of equality with uninterpreted function symbols [NO80]):

Given a finite relation R on terms and a pair of terms A and B,

determine if $A =_R B$ where $=_R$ is the congruence closure of R.

In other words, the Congruence Closure Algorithm solves ET in the case where the axiom schemata contain no variables.

The Congruence Closure Algorithm is most easily understood as an algorithm on a directed graph G. The graph G consists of the forest representing the terms in the relation R plus the trees representing A and B; each vertex has a label (from the term alphabet Σ) and the successors of each vertex are ordered (so that we can, for example, distinguish a left subtree from a right subtree). The relation R on Σ terms is easily seen to be equivalent to a relation on the vertices of G. Let e be the number of edges in the graph G. We assume there are no isolated vertices; thus, n, the number of vertices, is O(e).

Kozen [Koz77] shows this problem can be solved in time polynomial in e. Nelson and Oppen [NO80] give a simple algorithm that runs in worst-case time $O(e^2)$ and space O(e). Downey, Sethi, and Tarjan [DST80] give a more complicated algorithm that runs in worst-case time $O(e \log^2 e)$ and space O(e) or in worst-case time (e log e) and space $O(e^2)$ (or, using a hash table, in average time O(e log e) and space O(e)).

We present a version of the algorithm similar to that given by Nelson and Oppen [NO77]. The algorithm partitions the vertices into equivalence classes using the procedures FIND and UNION. FIND(u) returns the name of the equivalence class containing vertex u. UNION(u,v) combines the equivalence class of vertex u with the equivalence class of vertex v. These procedures can be implemented using the efficient set union algorithm given by Tarjan [Tar75] which takes time $O(n\ \alpha(n))$ for a series of n UNION and FIND operations where $\alpha()$ is a function which grows very slowly.

The Congruence Closure Algorithm uses the signatures of vertices to determine which equivalence classes should be combined.

Definition 3.4. The signature of a vertex v is an ordered tuple $(\lambda(v),FIND(v_1),...,FIND(v_m))$ where $\lambda(v)$ is the label of vertex v and $v_1,...,v_m$ are the successors (children) of v.

The idea is that if two vertices have the same signature then the terms represented by those vertices have the same root label and all the proper subterms are already known to be equivalent; thus, the two vertices should be placed in the same equivalence class.

Congruence Closure Algorithm (CCA):

input: R, a finite relation on terms;
A and B, terms to be checked for R-equivalence;

begin
Construct the graph G from A, B, and R as explained above;
for each pair (u,v) in R {u and v are vertices of G} **do**
UNION(u,v);
repeat
Use FIND to compute the signature of each vertex in G;
Sort the vertices by signature producing $v_1,...,v_s$;
for each pair (v_i,v_{i+1}) with matching signatures **do**
if FIND(v_i)≠FIND(v_{i+1}) **then** UNION(v_i,v_{i+1});
until no UNIONs are done;
end.

Once the algorithm halts, to check if two terms are equivalent under $=_R$ we check whether the corresponding vertices are in the same equivalence class.

This algorithm runs in worst case time $O(e^2)$ where e is the number of edges in the graph G. To see this, note that there can be at most n-1 calls to UNION, where n is the is the number of vertices in G (after n-1 calls everything would be in a single equivalence class); thus, the repeat loop will be done at most O(n) times. There are at most O(ne) calls to FIND needed to compute the signatures; thus, the total time spent doing UNIONs and FINDs is O(ne) [Tar75]. Using a lexicographic sort as in [AHU74], a single sort of the vertices can be done in O(e) time. The total algorithm thus takes time $O(ne) = O(e^2)$.

The faster Congruence Closure Algorithm of Downey, Sethi, and Tarjan [DST80] keeps track of parent signatures for each class. With this information it is possible to keep track of which signatures change when a UNION is done so that duplicate signatures can be discovered without having to check every signature.

Note that some of the vertices of the graph (G) used in CCA become redundant. The only datum that affects the way a vertex is handled is its signature; once we have two vertices with the same signature in a single class we can dispense with one vertex. Thus CCA can be considered an algorithm that produces a finite set of signatures divided into equivalence classes. We consider the output of CCA to be a set $\sigma_1,...,\sigma_n$ of signatures, partitioned into equivalence classes $c_1,...,c_m$.

As a simple example, the following equivalence classes of signatures occur as a result of applying CCA to the axioms f(g(a),g(b))=a and a=b. Numbers are used as labels for equivalence classes and duplicate signatures have been eliminated.

1: f 2 2 2: g 1

a

b

Note that the resulting classes contain terms that did not appear in the input to CCA. For instance, by following the signatures it seems that class 1 contains f(g(a),g(a)). We say the new term is represented in the class 1. This will be more carefully defined in the next section.

CCA has these important properties:

1. Axioms are treated symmetrically; there is no direction associated with the axioms.
2. There are no syntactic restrictions on the axioms, such as the restrictions necessary to make SR work.
3. Each axiom is applied just once; that axiom is then remembered in the equivalence classes produced by CCA.
4. For any represented term A, all terms R-equivalent to A are represented in the class of A. For instance, if a=f(a) is an axiom of R then the class that represents a also represents f(a), f(f(a)), and

f(f(f(...f(a)...))).

CCA has one important disadvantage: only finitely many axiom instances can be used, a major problem, since we wish to solve NF using an infinite set of axiom instances derived from a finite set of axiom schemata.

5 Directed Congruence Closure

Though CC can process only finitely many axiom instances, it is still potentially useful. In any particular instance of NF, although there are infinitely many possible axiom instances, only finitely many of them are actually used. In other words, if a term D has a normal form then that normal form can be reached in finitely many steps. This observation leads to the following procedure for NF.

input: Δ, a set of axiom schemata;
D, a term to be reduced to (Δ) normal form;
begin
K := Ø;
repeat
Select some instances from Δ and add them to **K**;
Run CC on D and **K**;
until the class of D contains a normal form (w.r.t. Δ);
Print the normal form;
end.

Assuming for the moment that there is a way to implement each of the steps in the program sketched above, it should have some advantages (from CC) over SR. Advantage 1 from CC is lost: to even speak of finding a normal form we must use axioms with a direction; thus the axioms cannot be treated symmetrically. To ensure that each expression has a unique normal form and to ensure that there is an effective strategy for choosing reducible subterms, we must require the set of axiom schemata to be regular as in SR; thus we must have syntactic restrictions and advantage 2 is lost. But, we retain advantage 3: each axiom is applied just once; thus solving the problem of repeated effort that we encountered with SR. We also retain advantage 4: all equivalent terms are represented. This

implies that the program will halt in some cases where SR would run forever.

There are two major difficulties with implementing the procedure sketched above:

1. How can we recognize when a normal form is reached?
2. How do we select axiom instances from Δ?

In the remainder of this section we show how the output of CC can be used to efficiently select appropriate axiom instances and to efficiently recognize a normal form when one is reached.

First recall that the output of CC can be considered as a set of signatures partitioned into equivalence classes. When selecting axioms to add to **K** only terms represented in the output of CC are likely to be of any use. Intuitively, these represented terms are those that can be built by (1) following the links from a signature to its children, then (2) selecting a single signature from each child class, and (3) recursively following the links of each of these signatures. A term that can be built in this manner, starting from a signature σ is said to be represented in σ. Such a term is also said to be represented in c, the equivalence class that contains σ.

In effect the Directed Congruence Closure Algorithm constructs a distinguished term (a represented term) that can be examined to (1) find if it is a normal form, in which case we are done, or (2) discover appropriate axiom instances, appropriate in the sense that instances that reduce this distinguished term are also instances that will lead to a normal form. This distinguished term is constructed as any represented term can be constructed (by recursively following the signature-class links and selecting a signature from each class), but for this distinguished term we select a particular signature (called the unreduced signature) of each class. This intuitive picture is not entirely accurate because (1) not every

class has an unreduced signature and (2) the process of following the signature-class links can get into an infinite loop.

At this point we present an outline of Directed Congruence Closure. Like the Congruence Closure Algorithm, it is based on the use of signatures, but here signatures can be marked either reduced or unreduced. New signatures are always marked unreduced. The procedure that forms the basis of DCC is as follows:

global: Δ, a regular set of axiom schemata;
S, a set of signatures divided into equivalence classes;
{Initially, S corresponds to a single term (e.g., the term for which we wish to find a normal form). S is altered only by CLOSE.}

procedure CLOSE(**J**);
{ **J** is a finite set of axiom instances derived from Δ}
begin
S := S ∪ signatures derived from **J**;
for each instance A⟹B of **J** **do**
UNION the corresponding signatures;
Mark the signature corresponding to A as reduced;
end for;
while there exist signatures σ_1 and σ_2 that match **do**
if FIND(σ_1)≠FIND(σ_2) **then** UNION(σ_1,σ_2);
if either σ_1 or σ_2 is marked reduced
then Mark the other reduced;
Throw away either σ_1 or σ_2 (remove it from S);
end while;
end CLOSE;

Note that procedure CLOSE is just the Congruence Closure Algorithm with a few extra steps added to keep track of whether each signature is reduced or unreduced. What makes the procedure useful is the information that can be derived from S, the set of signatures produced by the procedure. The most important properties are summarized below. The proofs for these properties are given in the next two sections.

1. Each class c has at most one representative signature called the unreduced signature of c.
2. These unreduced signatures can be used to construct a directed graph, called the term graph. The vertices of the term graph correspond to the equivalence classes of S. For class c with unreduced signature $\sigma=(\lambda,c_1,\ldots,c_k)$, the corresponding vertex c in the the term graph has label λ and successors $c_1,\ldots,c_k$. A class without an unreduced signature corresponds to an unlabeled vertex with no successors.
3. Each vertex (class) in the term graph corresponds to a (possibly infinite) term that can be constructed by following edges in the term graph. The term corresponding to vertex c is called the primary term of c. The primary term of c is finite iff the portion of the term graph that can be reached from c contains no cycles. It is never necessary to explicitly construct this term since it is implicit in the structure of the term graph.
4. Assume that D is a represented term that has a Δ normal form. Axiom instances that reduce primary terms are axiom instances that lead to the normal form of D.
5. If T, the primary term of class c, is finite and has no unlabeled vertices and if no axiom schema of Δ can reduce T then T is the normal form for class c. In other words, if the primary term for c has these properties then, for any term A represented in c, T is the normal form of A.

These properties allow us to design an efficient algorithm for finding normal forms. Such an algorithm is outlined below. A more efficient program is given in Section 7.

```
Program NFs (NF-simple):
    input:  Δ, a regular set of axiom schemata;
            D, a term to be reduced to Δ normal form;
    begin
      S := signatures derived from D;
      loop
        T := the primary term of the class of D;
        J := all axiom instances of Δ that apply to T or a subterm of T;
      exit loop if J = Ø;
        CLOSE(J);
      end loop;
      if T has no cycles and no unlabeled vertices
        then T is the Δ normal form of D;
        else D has no normal form;
    end.
```

6 Why it Works

At any time during the execution of Directed Congruence Closure the state of DCC is given by S, the set of signatures divided into equivalence classes.

Proposition 6.1. S, the current state of DCC, is entirely determined by the initial term (e.g., the term to be reduced to normal form) and the set **K** of axiom instances that have been used to this point. In particular, S is unaffected by changing the order in which axiom instances are used.

In other words, S depends on just the initial term (we usually use D as the initial term) and $\mathbf{K} = \mathbf{J}_1 \cup \ldots \cup \mathbf{J}_k$ where we assume CLOSE has been called k times with sets of axiom instances $\mathbf{J}_1$ through $\mathbf{J}_k$. The state S also includes a mark for each signature indicating whether the signature is reduced or unreduced. This information is also independent of the order in which axiom instances are used. We use S(D,**K**) to represent the current state of DCC.

Notation. S(D,**K**) is the state of DCC after starting with initial term D and applying the axiom instances of **K**. The state includes:

1. the set of signatures divided into equivalence classes, and
2. a mark for each signature indicating whether the signature is reduced or unreduced.

6.1 c* and σ*

The initial term (call it D) and all the terms that appear in **K** can be reconstructed by choosing the appropriate class, choosing the appropriate signature within that class, and following the signature links to other classes, etc. Terms that can be built in this manner are said to be represented in S(D,**K**). Congruence Closure has the property that not only

are these original terms represented in S(D,**K**), but any term equivalent ($=_{\mathbf{K}}$) to one of these terms is also represented.

Definition 6.1.1. Let $\sigma=(f,c_1,...,c_n)$ be a signature of S(D,**K**) and let c be a class of S(D,**K**).

$\sigma^* = \{ f(A_1,...,A_n) \mid A_i \in c_i^* \text{ for each } i\}$

$c^* = \{ A \mid A \in \tau^* \text{ for some signature } \tau \in c\}$

Intutively, σ^* is the set of terms that can be built by starting at signature σ; c^* is the set of terms that can be built by starting from some signature of c. The following propositions follow from the correctness of the Congruence Closure Algorithm and give some of the important properties of these sets of terms.

Proposition 6.1.2. Let A and B be terms where A is represented in S(D,**K**), say $A \in c^*$ where c is an equivalence class of signatures.

$$A =_{\mathbf{K}} B \text{ iff } B \in c^*.$$

Proposition 6.1.3. Let σ and τ be signatures of S(D,**K**) and let A be a term.

$$A \in \sigma^* \text{ and } A \in \tau^* \text{ implies } \sigma=\tau.$$

In other words, each term appears in at most one signature.

Proposition 6.1.4. For every signature σ of S(D,**K**), σ^* is nonempty.

Proof. Each signature is created to represent some term and that term can always be reconstructed. □

6.2 The Closure of a Set of Axiom Instances

We need a way to relate $=_{\mathbf{K}}$, derived using axioms both forward and backward, to **K** normal forms, a concept based on the use of axioms only in the forward direction. This relation is derived using $\bar{\mathbf{K}}$, the closure of **K**. $\bar{\mathbf{K}}$ is an imaginary closed set of axiom instances, often infinite, with

useful properties needed in our proofs. $\bar{\mathbf{K}}$ can be used because in many cases $\bar{\mathbf{K}}$ produces the same results as **K**: for instance, we show $A =_{\mathbf{K}} B$ iff $A =_{\bar{\mathbf{K}}} B$. $\bar{\mathbf{K}}$ is also used in the proof that each class produced by DCC contains an unreduced signature. First, we define what is meant by a closed set of axiom instances.

Definition 6.2.1. Let **K** be a set of axiom instances from Δ, a regular set of axiom schemata. **K** is said to be <u>closed</u> with respect to Δ iff for each set of terms A, B, and A' such that $A \Longrightarrow_{\mathbf{K}} B$ and $A \langle nr \rangle\!\!\rightarrow_{\mathbf{K}} A'$, there exists a term B' such that $A' \Longrightarrow_{\mathbf{K}} B'$.

(Note: Because Δ is regular it follows that $A \Longrightarrow B$ and $A' \Longrightarrow B'$ are instances of the same axiom schema and that $B \langle * \rangle\!\!\rightarrow_{\mathbf{K}} B'$.) The definition given here is a less general version of the definition of closed given by Rosen [Ros73] for sets of axioms not related to schemata. In his definition a set **K** of axioms is closed iff $A \Longrightarrow_{\mathbf{K}} B$ and $A \langle nr \rangle\!\!\rightarrow_{\mathbf{K}} A'$ implies there exists a term B' such that $A' \Longrightarrow_{\mathbf{K}} B'$ and $B \langle * \rangle\!\!\rightarrow_{\mathbf{K}} B'$, where the reduction from B to B' occurs at copies of the subterm reduced in $A \langle nr \rangle\!\!\rightarrow_{\mathbf{K}} A'$. A set of axiom instances closed with respect to our definition is also closed with respect to Rosen's definition.

Theorem 6.2.2 (Rosen [Ros73]). If **K** is a closed, unequivocal set of axioms then **K** has the confluence property.

For our definition of closed, more restrictive than Rosen's, the phrase "closed, unequivocal" is redundant. For us, a closed set of axiom instances is automatically unequivocal by definition.

Definition 6.2.3. Let Δ be a regular set of axiom schemata. The <u>closure of **K** with respect to Δ</u> (written $\bar{\mathbf{K}}$) is the minimal closed (with respect

to Δ) set of axiom instances containing **K**.

Proposition 6.2.4. $\bar{\mathbf{K}}$ can be defined inductively as follows:

$$\mathbf{K}_0 = \mathbf{K}$$

$$\mathbf{K}_{i+1} = \{A' \Rightarrow B' \in i(\Delta) \mid \exists\, A \Rightarrow B \in \mathbf{K}_i \text{ and } A \langle nr \rangle\!\rightarrow_{\mathbf{K}_i} A'\}$$

$$\bar{\mathbf{K}} = \cup\, \mathbf{K}_i,\ 0 \le i < \infty.$$

Using the inductive definition of $\bar{\mathbf{K}}$ it is easy to show:

Proposition 6.2.5. Let **K** be a set of axiom instances from Δ, a regular set of axiom schemata. For all terms A and B, $A =_{\mathbf{K}} B$ iff $A =_{\bar{\mathbf{K}}} B$.

Using Theorem 6.2.2 we get:

Proposition 6.2.6. Let **K** be a set of axiom instances from Δ, a regular set of axiom schemata. $\bar{\mathbf{K}}$ has the confluence property.

These results on the relationship between represented terms, **K** and $\bar{\mathbf{K}}$ are summarized in the following theorem and corollary.

Theorem 6.2.7. Let **K** be a set of axiom instances from Δ, a regular set of axiom schemata and let A and B be terms. Assume A is represented in S(D,**K**), say in class c (i.e., $A \in c^*$). The following conditions are equivalent:

1. $B \in c^*$.
2. There exists a term C such that $A \langle * \rangle\!\rightarrow_{\bar{\mathbf{K}}} C$ and $B \langle * \rangle\!\rightarrow_{\bar{\mathbf{K}}} C$.
3. $A =_{\mathbf{K}} B$.
4. $A =_{\bar{\mathbf{K}}} B$.

Corollary 6.2.8. Let **K** be a set of axiom instances from Δ, a regular set of axiom schemata and let A and B be terms. Assume A is represented in S(D,**K**), say in signature σ (i.e., $A \in \sigma^*$). The following conditions are equivalent:

1. $B \in \sigma^*$.

2. There exists a term C such that $A \langle nr^* \rangle \rightarrow_{\bar{K}} C$ and $B \langle nr^* \rangle \rightarrow_{\bar{K}} C$.

6.3 Unreduced Signatures

As mentioned in Section 5, unreduced signatures are used to build the term graph. In turn, the term graph is used to select appropriate axiom instances and to determine when a normal form has been reached. We first define what is meant by a reduced signature.

Defintion 6.3.1. For signatures of S(D,**K**):

A signature σ is said to be reduced if it satisfies any of the three conditions of the following lemma.

Lemma 6.3.2. Let **K** be a set of axiom instances from Δ, a regular set of axiom schemata. For any signature σ of S(D,**K**), the following conditions are equivalent:

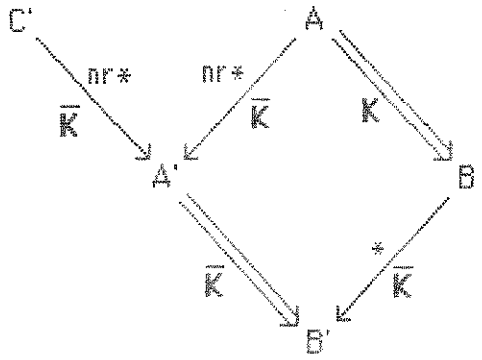
1. There exist terms A and B such that $A \in \sigma^*$ and $A \Rightarrow_K B$.

2. There exist terms A and B such that $A \in \sigma^*$ and $A \Rightarrow_{\bar{K}} B$.

3. For all $C \in \sigma^*$ there are terms A and B such that $C \langle nr^* \rangle \rightarrow_{\bar{K}} A \Rightarrow_{\bar{K}} B$.

Proof.

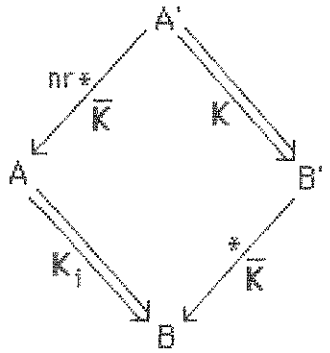
(1⇒3). By hypothesis there exist terms A and B such that $A \in \sigma^*$ and $A \Rightarrow_K B$. Let C' be any member of σ^*. The proof is exhibited in the following picture:



Since C' and A are both in σ^*, the term A' exists by Corollary 6.2.8. The term B' exists because $\overline{\mathbf{K}}$ is closed.

(3⟹2). Consequence of σ^* being nonempty (Proposition 6.1.4).

(2⟹1). By hypothesis there exists a term $A \in \sigma^*$ such that $A \Longrightarrow_{\overline{\mathbf{K}}} B$. By the inductive definition of $\overline{\mathbf{K}}$ (Proposition 6.2.4) there exists an i such that $A \Longrightarrow B$ is a member of $\mathbf{K}_i$. At this point we need the following result, proved by induction on i: If $A \Longrightarrow B$ is a member of $\mathbf{K}_i$ then there exist terms A' and B' such that



Using Corollary 6.2.8, we show $A' \in \sigma^*$; thus completing the proof. □

Definition 6.3.3. A signature is said to be <u>unreduced</u> iff it is not reduced.

Lemma 6.3.4. Let **K** be a set of axiom instances from Δ, a regular set of axiom schemata. Each equivalence class of S(D,**K**) contains at most one unreduced signature.

Proof. Assume class c contains unreduced signatures σ and τ. Choose $A \in \sigma^*$ and $B \in \tau^*$. Since A and B are both in c^*, it follows from Theorem

6.2.7 that there exists a term C such that A $\langle * \rangle\!\!\rightarrow_{\overline{K}}$ C and B $\langle * \rangle\!\!\rightarrow_{\overline{K}}$ C. If any of the reductions on the way from A to C is a root reduction then σ is, by definition, reduced, a contradiction; thus, A $\langle nr^* \rangle\!\!\rightarrow_{\overline{K}}$ C and, similarly, B $\langle nr^* \rangle\!\!\rightarrow_{\overline{K}}$ C. By Corollary 6.2.8, terms A, B, and C are all represented by the same signature; therefore, since each term is represented by at most one signature (Proposition 6.1.3), we have $\sigma=\tau$. □

6.4 The Term Graph, Primary Terms, and the Represenation of Axiom Instances

The term graph is a directed graph built using the unreduced signature of each class. Intuitively, the term graph represents the current version of the initial term. Each equivalence class, c, corresponds to a vertex in the term graph. If c contains only reduced signatures then the vertex is unlabeled and has no successors. If c has an unreduced signature, say $\sigma=(\lambda,c_1,\ldots,c_k)$, then the vertex is labeled λ and has as successors the vertices corresponding to classes $c_1,\ldots,c_k$. Note that the successors of each vertex are ordered so that, for instance, the edge corresponding to the first successor can be distinguished from the edge corresponding to the second successor.

Each class that has an unreduced signature corresponds to a primary term in the term graph. A primary term can, for our purposes, usually be treated as a term, although a primary term is not necessarily a term since it may be infinite. Let c be a class with an unreduced signature. The primary term of c is the portion of the term graph that can be reached by following directed edges starting from vertex c. Let T be the primary term of c. If T has no cycles then T obviously corresponds to a term over

the alphabet $\Sigma \cup$ {blank}, while if T has one or more cycles, T corresponds to an infinite term. Of course, even when T corresponds to an infinite term it has a simple finite representation (i.e., the term graph).

An axiom schema can be matched to a primary term in the same manner that such a scema can be matched to a term. A simple procedure first checks if the labels at the root match, then recursively checks if the children match.

Once an axiom schema is matched we need to create the corresponding axiom instance by substituting for the variables. Although terms and primary terms can use the same procedure for matching, there is some difference in the way variables are assigned. When a term is matched a variable corresponds to a subterm, when a primary term is matched a variable corresponds to a class. In practice, it is not necessary to actually create a complete axiom instance after a match is found (although it is simple to do so). Since we work with just the signatures, all we need to do is ensure that the appropriate signatures are created. Instead of substituting a term for a variable, we need only know the appropriate class. Knowing the class associated with a variable is enough to build the proper signatures for the righthand side of an axiom instance.

Example. Let Δ={ f(X)$\Rightarrow$f(f(X)), h(a,X)$\Rightarrow$f(X) }. We start with the initial term g(f(a),h(a,b)). Before any axiom instances are introduced, we have the following signatures:

1: g 2 3	2: f 4	3: h 4 5
4: a	5: b	

Numbers are used to represent equivalence classes. At this point, since no axiom instances have been used, each class has just one signature and all signatures are unreduced. We use the instance h(a,b)$\Rightarrow$f(b) and apply DCC to get:

1: g 2 3 2: f 4 3: h 4 5 *
f 5

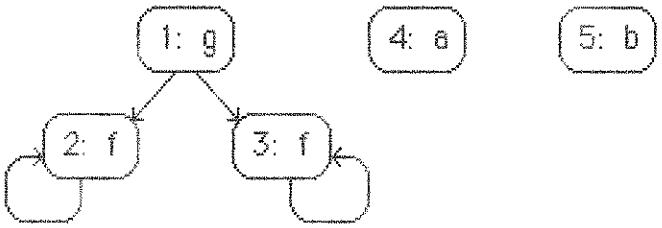
4: a 5: b

Note that class 3 now contains 2 signatures. Signature (h,4,5) has been marked with * to show that it has been reduced. The primary term of class 1 is now g(f(a),f(b)); the primary term of class 3 is f(b). DCC is applied after introducing instances f(a)⇒f(f(a)) and f(b)⇒f(f(b)) giving:

1: g 2 3 2: f 4 * 3: h 4 5 *
f 2 f 5 *
f 3

4: a 5: b

The primary term of class 3 is now f(f(f(...))). The term graph is as follows:



At this point, class 2 and class 3 both match the axiom schema f(X)⇒f(f(X)). Signature (f,2) ("f 2" in the list of signatures above) matches the left side with X equal to class 2. Signature (f,3) matches the left side with X equal to class 3. In effect the axiom instances we use are f(2)⇒f(f(2)) and f(3)⇒f(f(3)). (Alternately, X can be said to match an arbitrary represented term of the corresponding class. For example, the axiom schema matches class 2 with X=f(a) giving the instance f(f(a))⇒f(f(f(a))). In section 6.5 it is shown that any term represented in class 2 will produce the same signatures and cause the same classes to be combined.) After the above instances are introduced and after applying DCC we get:

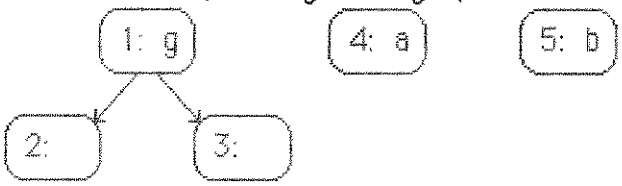
1: g 2 3 2: f 4 * 3: h 4 5 *

f 2 * f 5 *

f 3 *

4: a 5: b

No new signatures have been created, but all signatures in classes 2 and 3 have been reduced. The corresponding term graph is now:



Note that classes 2 and 3 no longer have labels or successors. At this point no further reduction is possible.

6.5 Choosing Axiom Instances

In this section we prove a theorem and a corollary that show why it is possible to restrict our attention to primary terms. Intuitively, these results say that useful axiom instances are exactly those that reduce primary terms.

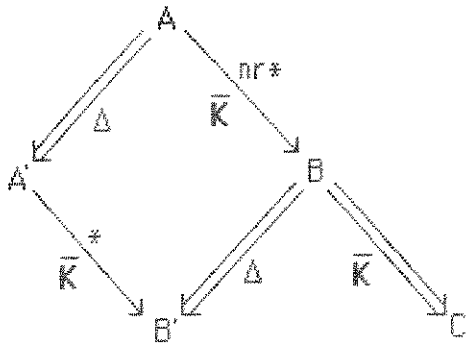
Theorem 6.5.1. Let **K** be a set of axiom instances from Δ, a regular set of axiom schemata. Let A be a term represented in S(D,**K**), say A∈σ*, and suppose there exists a term A' and an axiom schema s∈Δ such that A⟹A' is an axiom instance of s. The following conditions are equivalent.

1. S(D,**K**) ≠ S(D, **K**∪(A⟹A')).
2. Signature σ is unreduced.
3. Axiom schema s matches the primary term of σ.

Proof.

(1⟹2). We show that if σ is reduced then S(D,**K**) = S(D, **K**∪(A⟹A')). Since σ is reduced and A∈σ*, there exist terms B and C such that

$A \langle nr^* \rangle\!\!\rightarrow_{\bar{\mathbf{K}}} B \Rightarrow_{\bar{\mathbf{K}}} C$ by Lemma 6.3.2. The set of all axiom instances from Δ is closed so we get



Since Δ is unequivocal, it follows that B' and C are the same term. Thus, A' can be reached from A by following a trail of $\bar{\mathbf{K}}$ reductions (using both forward and backward reductions) and by definition $A =_{\bar{\mathbf{K}}} A'$. It follows (using Theorem 6.2.7) that $A =_{\mathbf{K}} A'$. We conclude that $S(D,\mathbf{K}) = S(D,\mathbf{K} \cup (A \Rightarrow A'))$ because (1) A' was already represented (Proposition 6.1.3), thus, no new signatures or classes are created by adding axiom instance $A \Rightarrow A'$, (2) we already have $A =_{\mathbf{K}} A'$, thus $A \Rightarrow A'$ causes no classes to be combined, and (3) σ, the signature associated with term A, is already reduced, thus $A \Rightarrow A'$ causes no change in reduced/unreduced information.

(2⇒3). Let T be the primary term of σ. If A and the T are the same then the result follows immediately, so assume A and T are different. Recall that locations within A correspond to signatures of $S(D,\mathbf{K})$ and that T is built using vertices that are either unlabeled (for classes in which all signatures are reduced) or that correspond to unreduced signatures. If A and T are different then there must be some location where either A and T use different signatures or T has an unlabeled vertex. Let x be such a location chosen to be maximal (i.e., A and T agree along the path from the root to x, but they use different signatures at x). Note that x cannot be the root because we have assumed σ is unreduced.

Let A_0 be the subterm of A at x and let τ be the signature corresponding to A_0 (i.e., $A_0 \in \tau^*$). Note that if A and T have different signatures at x then τ must be reduced since T uses the only unreduced signature. We must also conclude τ is reduced when A and T differ due to the use of an unlabeled vertex at x in T; τ must be reduced because, otherwise, T would use τ instead of the unlabeled vertex. In either case τ is reduced and by the definition of reduced, there exist terms B_0 and C_0 such that $A_0 \langle nr^* \rangle\!\!\rightarrow_{\overline{K}} B_0 \Longrightarrow_{\overline{K}} C_0$.

Consider what happens to term A as A_0 is reduced to B_0 and then to C_0. We get terms B and C such that $A \langle nr^* \rangle\!\!\rightarrow_{\overline{K}} B \langle nr \rangle\!\!\rightarrow_{\overline{K}} C$ where the reduction from B to C takes place at location x and all other reductions take place below (within the subterm at) location x. We know schema s can be applied at the root of term A to produce $A \Longrightarrow A'$. Because the set Δ of axiom schemata is nonoverlapping, s must apply (at the root) to B and also to C. Since the reduction from B to C takes place at location x, we know a change at x cannot affect whether or not schema s matches B. We also know a change at x cannot affect whether or not schema s matces A. Since x was an arbitrary maximal location where A and T differed, we conclude that any differences between A and T do not affect whether axiom schema s matches. Thus, schema s must match T, the primary term of σ.

(3⇒1). For σ to have a primary term it must be the case that σ is unreduced. If DCC is applied to the axiom instance $A \Longrightarrow A'$ then, since $A \in \sigma^*$, σ will be marked reduced. Obviously, if the status of some signature changes, it must be the case that $S(D,\mathbf{K}) \neq S(D, \mathbf{K} \cup (A \Longrightarrow A'))$. □

Theorem 6.5.1 shows that only axiom instances that affect an

unreduced signature can cause progress toward a normal form. The next result shows that if there is a choice of axiom instances to reduce a signature then any of the instances will have the same result. First we need some notation.

Notation. Inst(s,A) represents the axiom instance that results from applying axiom schema s to term A. Inst(s,σ) represents the axiom instance that results from applying axiom schema s to the primary term of signature σ. Note that s must match for either notation to be valid.

Corollary 6.5.2. Let **K** be a set of axiom instances from Δ, a regular set of axiom schemata. Let A be a term and let σ be an unreduced signature of S(D,**K**) such that A$\in\sigma^*$ and such that A matches some axiom schema s$\in\Delta$. Then S(D,**K**$\cup$Inst(s,A)) = S(D,**K**$\cup$Inst(s,σ)).

Proof. First note that s must match the primary term of σ by Theorem 6.5.1, so Inst(s,σ) is well defined. There are 3 ways in which an axiom instance affects the state of DCC: (1) new signatures are created, (2) a signature may be marked reduced, and (3) classes are combined. We show that Inst(s,A) and Inst(s,σ) both affect the state in the same way.

First, consider the new signatures that are created. No new signatures are created for the left side of either instance because both instances act on represented terms. The proof of Theorem 6.5.1 showed that A and the primary term of σ use exactly the same signatures on the nonvariable portion of the lefthand side of s; thus, variables correspond to the same classes in both Inst(s,A) and Inst(s,σ). The only possible new signatures are those that correspond to the nonvariable potion of the righthand side of s. Since variables correspond to the same classes for both Inst(s,A) and Inst(s,σ), it follows that the same signatures are created for the

remainder of the righthand sides. Thus, both instances create the same new signatures.

For either instance, signature σ is marked reduced. Thus, both instances have the same effect on the reduced/unreduced information.

Both Inst(s,A) and Inst(s,σ) cause the same classes to be combined. If the righthand side of s is a single variable then, as shown above, that variable corresponds to the same class for both instances. If the righthand side is not just a single variable then, since both of the instances use the same signatures, it follows that the same class appears for the right side of each instance. □

7 Normal Forms

7.1 Detecting Normal Forms

We need a way to detect a normal form. Our previous results tell us that axiom instances that reduce the primary term are instances that help lead to a normal form. We show in this section that the primary term can also be used to detect when a normal form has been reached.

Lemma 7.1.1. Let $\mathbf{K}$ be a set of axiom instances from Δ, a regular set of axiom schemata. Let A be a represented term of S(D,$\mathbf{K}$), say $A \in \sigma^*$. A is in $\bar{\mathbf{K}}$ normal form iff A is the same as the primary term of σ.

Proof.

($\Rightarrow$). Since A is in $\bar{\mathbf{K}}$ normal form, no subterm of A can be $\bar{\mathbf{K}}$-reduced. Thus, by the defintion of reduced signatures (Lemma 6.3.2), all subterms of A correspond to unreduced signatures. Since the primary term of σ is defined to be the term built with unreduced signatures, it follows that A and the primary term of σ are the same.

($\Leftarrow$). Assume A is not in $\bar{\mathbf{K}}$ normal form. By definition, A can be $\bar{\mathbf{K}}$-reduced. Thus, there exist terms A' and B' such that A' is a subterm of A and $A' \Rightarrow_{\bar{\mathbf{K}}} B'$. As a subterm of a represented term A' is represented and, by the defintion of a reduced signature, the signature corresponding to A' is reduced. Since part of A corresponds to a reduced signature, A cannot be the same as a primary term. □

This result gives us a simple way to detect Δ normal forms. First note that a $\bar{\mathbf{K}}$ normal form is easy to detect - it is a primary term that is finite (no loops) and that has no unlabeled vertices. A term is in Δ normal form

iff it is in $\bar{K}$ normal form and no axiom schema of Δ applies.

7.2 A Program for Finding Normal Forms

We now have the tools to prove correctness for a normal form finding program. A more efficient program is given in section 7.3.

Program NFp (NF - preliminary):

```
    input:  Δ, a regular set of axiom schemata;
            D, a term to be reduced to Δ normal form;
    begin
      S := signatures derived from D;
      loop
        T := the primary term of the class of D;
        H := all axiom instances of Δ that apply to T or a subterm of T;
        J := all axiom instances of Δ that apply to some primary term;
      exit loop if H = Ø;
        CLOSE(J);
      end loop;
      if T has no cycles and no unlabeled vertices
        then T is the Δ normal form of D;
        else D has no normal form;
    end.
```

Note that two different sets of instances are used, **H** and **J**. **H** is used to detect when a normal form has been reached. **J** is used to create reductions. It is easy to see that **H** is a subset of **J**, and that, in practice, **H** could easily be computed as part of a procedure that finds **J**.

Theorem 7.2.1. Let Δ be a regular set of axiom schemata and let D be a term to be reduced to Δ normal form. Program NFp will find the Δ normal form of D if one exists.

Proof. Consider a reduction $A \rightarrow_{\Delta} B$. If A is represented at some point in Program NFp then B is represented on the next cycle through the main loop of Program NFp. To see this, note that, by Theorem 6.5.1, either A

corresponds to a reduced signature and we already know $A=_{\Delta}B$ or A corresponds to an unreduced signature and, by Corollary 6.5.2, a primary term will be used to create an instance that will prove $A=_{\Delta}B$. In either case, by Proposition 6.1.2, B is represented on the next cycle.

In particular, if D has Δ normal form D_n then there is a reduction sequence $D\rightarrow_{\Delta}D_1\rightarrow_{\Delta} \ldots \rightarrow_{\Delta}D_n$, and by the observation above, for each i, D_i is represented after i cycles through the main loop. Lemma 7.1.1 shows that once the normal form is represented, it will be recognized; thus the Δ normal form for D will be found. □

Corollary 7.2.2. For a regular set Δ of axiom schemata, if $D\rightarrow_{\Delta}D_1\rightarrow_{\Delta} \ldots \rightarrow_{\Delta}D_n$ is the shortest reduction sequence leading from D to the normal from D_n then Program NFp will reach the normal form within n cycles through the main loop.

Proof. Follows from the proof of Theorem 7.2.1. □

Note that Program NFp may do several reductions on each cycle through the main loop, so Corollary 7.2.2 does not provide a bound on the number of reductions that may be done in the process of finding a normal form. In fact, there may be exponentially many extra reductions performed.

Example. Δ: $a_1 \Longrightarrow a_2 \Longrightarrow \ldots \Longrightarrow a_n \Longrightarrow a$

$h(a,X) \Longrightarrow b$

$g(X) \Longrightarrow f(g(s(X)),g(t(X)))$

Suppose we wish to find the normal form of $h(a_1,g(b))$. The shortest reduction sequence takes n+1 steps: n steps to get to h(a,g(b)) and 1 more step to get to b, the normal form. Program NFp would require n+1 cycles through the main loop, but, due to the third axiom schema, the number of g

redexes doubles each cycle; thus, Program NFp does $2^{n+1}-1$ extra reductions. Note that SR (Straight Reduction) would also work on these extra reductions.

Exponential growth in the number of reductions, as in the example above, is the worst that can occur. To see this, note that there is at most one reduction per class and that the maximum number of new classes per reduction is O(m) where m is the maximum size of the $\Sigma \cup V$ terms in Δ. Thus, if Program NFp starts with p classes then at the end of n cycles through the main loop there are at most $p+pm+\ldots+pm^n$ classes and $O(m^n)$ reductions have been done. Thus, given a term in which the shortest possible reduction sequence leading to a normal form has length n, Program NFp could take time $O(m^n)$, while an imaginary version of SR that made all the correct choices would take time O(n).

7.3 Improvement using Outermost Reductions

Program NFp does too much work: it checks every class for possible matches, even those classes that are unrelated to the current version of the term to be reduced to normal form.

Example. Δ: $h(X,Y) \Longrightarrow X$

$a \Longrightarrow a_1 \Longrightarrow a_2 \Longrightarrow \ldots \Longrightarrow a_n$

$g(X) \Longrightarrow g(f(X))$

Suppose we attempt to find the normal form of h(a,g(b)). After the reduction $h(a,g(b)) \Longrightarrow a$, the term g(b) is clearly unimportant. However, Program NFp will do n steps on the reduction sequence $g(b) \rightarrow g(f(b)) \rightarrow g(f(f(b))) \rightarrow \ldots$ since each reduction create a new class. It would be better if some way could be found for the program to realize that further reductions of g(f(...f(b)...)) are not helpful.

Using O'Donnell's result [O'D77] that an eventually outermost reduction sequence will always find a normal form if one exists, we can improve Program NFp. The improved version, Program NFo, is less likely to spend time on unnecessary reductions. The major difference between Program NFp and Program NFo is in the choice of reductions (i.e., the set of axiom instances sent to procedure CLOSE).

Program NFo (NF - outermost):

input: Δ, a regular set of axiom schemata;
D, a term to be reduced to Δ normal form;

begin
S := signatures derived from D;
loop
T := the primary term of the class of D;
J := all axiom instances of Δ that correspond to an outermost redex of T;
exit loop if J = **Ø**;
CLOSE(**J**);
end loop;
if T has no cycles and no unlabeled vertices
then T is the Δ normal form of D;
else D has no normal form;
end.

Note that even though T is not technically a term (it may be infinite), the concept of an outermost redex is still well-defined. It is a redex with no other redexes between it and the root.

Theorem 7.3.1. Program NFo will find a normal form if one exists.

Proof. Let **K** be the set of axiom instances that have been used at some intermediate point in the algorithm and let D be the term that is to be reduced to normal form.

Claim: If $A =_{\mathbf{K}} D$ and A has outermost redex x then on the next cycle through the main loop of Program NFo there exists a represented term $C =_{\mathbf{K}} D$ and a Δ

reduction sequence from A to C such that the sequence eliminates the redex at x.

To show this we examine two cases: (1) every subterm of A that contains x corresponds to an unreduced signature, and (2) there is a subterm A' of A such that A' contains location x and A' corresponds to a reduced signature.

For case (1), we note that the redex at x is, by definition, outermost in T, the primary term of the class of D. It is straightforward to complete the proof of the claim.

For case (2), by hypothesis, A' is at some location y>x. By the definition of reduced (Lemma 6.3.2), there are terms B' and C' such that $A' \langle nr^* \rangle\!\!\rightarrow_{\overline{K}} B' \Rightarrow_{\overline{K}} C'$. By substituting back in the term A we get $A \langle nr^* \rangle\!\!\rightarrow_{\overline{K}} B \rightarrow_{\overline{K}} C$ where the reduction from B to C occurs at y>x. The term C is already represented by the class of D because $A =_K D$ and $A =_K C$. The reduction sequence from A to C is, thus, the desired reduction sequence that eliminates redex x.

From this result we can conclude that there exists an eventually outermost sequence $A_0 \langle * \rangle\!\!\rightarrow A_1 \langle * \rangle\!\!\rightarrow \ldots$ where $A_0 = A$ such that A_i is represented by the class of A on the ith time through the main loop. By Theorem 3.3 such a sequence will find a normal form if there is one; thus, Program NFo will work correctly. □

Corollary 7.3.2. Program NFs (presented in Section 5) will find a normal form if there is one.

Proof. Program NFs finds all instances that reduce the primary term of the class of D where D is the term for which we wish to find a normal form. The outermost redexes are included; therefore, the proof of Theorem 7.3.1 applies. □

7.4 Implementation and Running Time for Finding Normal Forms

An implementation of procedure CLOSE, the heart of Directed Congruence Closure, can be based on the straightforward Congruence Closure Algorithm (CCA) given in Section 4, or on the faster CCA of Downey, Sethi, and Tarjan [DST80]. The basic difference between the two CCAs is in how duplicate signatures are detected. The analysis of the straightforward CCA presented in Section 4 can be used to show that the total time spent in CLOSE is $O(e^2)$ where e is the total number of edges among all signatures used. If CLOSE is implemented using the faster CCA this can be brought down to O(e log e).

If Program NFo (or NFs or NFp) halts then the time it takes is at most $O(e^2)$ where e is the total number of edges among all the signatures used by the program. For this analysis we assume that CLOSE is implemented using the simple CCA outlined in Section 4, and that naive matching is used to detect reductions (i.e., try each axiom schema at each of the appropriate primary terms). The total time spent in CLOSE is $O(e^2)$. To this we must add the time spent looking for matches. A single test for a match takes time O(m) where m is the maximum size of the $\Sigma \cup V$ terms in Δ. To test all axiom schemata takes time O(mk) where k is the number of axiom schemata in Δ. The total number of signatures is bounded by e, so to find all matching schemata for a single term graph takes time O(mke). Since at least one signature is marked reduced after each term graph inspection except the last, it follows that there are O(e) term graph inspections. Thus, the total time spent matching is $O(mke^2)$. For any particular set Δ of axiom schemata, m and k can be considered constants; thus, the total time is $O(e^2)$.

This time bound can probably be improved by using a faster algorithm

for congruence closure and by using a more sophisticated matching technique. Hoffmann and O'Donnell [HO82a] discuss patterm matching in trees and present several different algorithms that are faster than the naive method suggested here. However, because the term graph is possibly cyclic, not all the algorithms they present are applicable.

It is not possible to bound the running time of Program NFs, Program NFp, or Prgram NFo in terms of the size of the inputs. For any total recursive time bound T we can find a regular term rewriting system and a term D with a normal form such that the time it takes for one of these programs to find the normal form for D is greater than T(n) where n is the size of D plus the size of the term rewriting system. This is because we can write axiom schemata that simulate a Turing Machine.

We can, however, compare Program NFo with an implementation of SR (Straight Reduction). We assume SR reduces every outermost redex on each cycle through its main loop. Intuitively, Program NFo should not require much more time than SR. They both go through the same basic steps:

0. Let D be the term to be reduced to normal form.
1. Examine the current version of D. For SR this is a single term. For Program NFo this is the primary term corresponding to D.
2. Find all axiom instances that reduce outermost redexes of the current version of D. If there are none then halt.
3. Apply the instances from step 2 and go to step 1.

Intuitively, steps 1 and 2 are almost the same for SR and Program NFo, so we will ignore the time spent on those steps. The major difference between the two programs is in step 3.

By implementing SR so that pointers are used to indicate the subtrees substituted for variables in the axiom schemata, each reduction can be

done in constant time; thus, the total time spent on reductions is O(r) where r is the number of reductions done.

Using the straightforward implementation of the Congruence Closure Algorithm given in Section 4, the total time spent doing reductions by Program NFo is $O(e^2)$ where e is the number of edges in the final term graph. Since each reduction produces a bounded number of new edges (the bound depends on Δ), Program NFo spends $O(r^2)$ time doing reductions where r is the number of reductions done.

This could be improved to time O(r log r) by using the faster congruence closure algorithm of Downey, Sethi, and Tarjan [DST80], but, even with the faster algorithm, it seems that Program NFo takes longer. However, Program NFo has two advantages over SR:

1. Since DCC remembers each axiom instance used, Program NFb may not need to do as many reductions as SR.
2. Program NFb can recognize certain cases where no normal form exists.

Unfortunately, there are examples for which it is a disadvantage to remember each instance that is used.

Example. Δ: 1: $a_1 \Rightarrow a_2 \Rightarrow \ldots \Rightarrow a_n \Rightarrow a$

2: $b_1 \Rightarrow b_2 \Rightarrow \ldots \Rightarrow b_n \Rightarrow b$

3: $g_1(X) \Rightarrow g_2(X) \Rightarrow \ldots \Rightarrow g_n(X) \Rightarrow g(X)$

4: $h(a,X) \Rightarrow h(b_1, g_1(b))$

5: $h(b,X) \Rightarrow b$

6: $g(X) \Rightarrow f(g(s(X)), g(t(X)))$

The last schema here is the trouble maker. Each time it is used, it produces four new classes and two new places where it can be applied; thus, it causes an exponential growth in the number of classes. Using SR

with an outermost strategy on $h(a_1,g_1(b))$, both a's and g's would be reduced, producing h(a,g(b)) after n cycles through the main loop (2n reductions). At this point schema 4 can be applied, producing $h(b_1,g_1(b))$, and the process begins again using b's instead of a's. This finally produces h(b,g(b)) which, using schema 5, reduces in one step to b, for a total of 2n+2 cycles through the main loop and 4n+2 reductions.

If Program NFo is used, the reduction sequence $g_1(b) \Rightarrow g_2(b) \Rightarrow \ldots \Rightarrow g_n(b) \Rightarrow g(b)$ is remembered; thus, instead of repeating that sequence after reaching $h(b_1,g_1(b))$, the program knows that it can instead use $h(b_1,g(b))$. This turns out to be a disadvantage because schema 6, the schema that causes exponential growth, can now be used. Though the number of cycles through the main loop is just 2n+2, the number of reductions done is $3n+1+2^n$.

We believe that pathological examples, such as this one, do not often occur in practice. Some experimentation is necessary to determine if the advantages of Directed Congruence Closure outweigh the slowdown that may occur for certain pathological sets of axiom schemata.

8 Proving Equivalence of Terms

Unfortunately, outermost reduction sequences cannot be used to prove equivalence of terms because the correct axiom instances might never be found.

Example. Δ: $f(X) \Rightarrow f(f(X))$

$a \Rightarrow b$

Starting with f(a) and f(b), outermost reductions only increase the number of f's without ever using the instance $a \Rightarrow b$. The reduction $a \Rightarrow b$ is a necessary step to prove f(a) equivalent to f(b).

The following program (similar to Program NFp) will work, but, like Program NFp, may do some unnecessary reductions.

Program ET:

```
input:  Δ, a regular set of axiom schemata;
        A and B, terms to be tested for Δ equivalence;
begin
   S := signatures derived from A and B;
   loop
      T := the primary term of the class of A;
      U := the primary term of the class of B;
      H := all axiom instances of Δ that apply to T or U;
      J := all axiom instances of Δ that apply to some primary term;
   exit loop if (class of A = class of B)
                or (J = Ø)
                or ((H = Ø) and (T is in Δ normal form))
                or ((H = Ø) and (U is in Δ normal form));
      CLOSE(J);
   end loop;
   if class of A = class of B
      then A and B are Δ equivalent;
      else A and B are not Δ equivalent;
end.
```

Recall that a primary term is in Δ normal form if it contains no unlabeled

vertices and no cycles and if no axiom schemata apply. Thus, the exit test for the loop is not difficult, although a number of details need to be considered.

Taken in order the exit tests are, intuitively, (1) quit if we have already discovered the terms are equivalent, (2) quit if no useful work can be done, and (3, 4) quit if one term has a normal form and the other term cannot be further reduced toward a normal form. Note that test 3 includes the case where normal forms have been found for both terms.

Theorem 8.1. Let Δ be a regular set of axiom schemata. If $A=_{\Delta}B$ then Program ET will find a proof. If both A and B have Δ normal forms and if $A\neq_{\Delta}B$ then the Program ET will show $A\neq_{\Delta}B$.

Proof. Similar to the proof of Theorem 7.2.1. □

Corollary 8.2. Let Δ be a regular set of axiom schemata. If $A\rightarrow_{\Delta}C$ in m steps and $B\rightarrow_{\Delta}C$ in n steps then Program ET will prove $A=_{\Delta}B$ within max(m,n) cycles through the main loop.

Proof. See the proof of Theorem 7.2.1. □

Program ET is particularly useful for showing $A=_{\Delta}B$ when A and B do not have normal forms. Other methods get into problems of the following sort. A_i, the current version of A, is different from B_i, the current version of B, but is the same as some older version of B that has been thrown away. The equivalence of A and B may not be detected, since the A's may never catch up with the B's. Directed Congruence Closure avoids this problem by representing all intermediate reduction steps in a compact and easily handled form.

An analysis similar to that presented for Program NFo shows that if

Program ET halts then the time it takes is at most $O(e^2)$ where e is the total number of edges among all the signatures used by the program. This should be contrasted with the time it takes to prove $A =_{\Delta} B$ using CCA (the Congruence Closure Algorithm). If the input to CCA is the same as the set of axiom instances chosen by Program ET then the time it takes to prove $A =_{\Delta} B$ is either $O(e^2)$ using the simple implementation given in Section 4 or O(e log e) using an implementation as in [DST80]. Of course, Program ET is able to choose appropriate axiom instances while CCA must be given the axiom instances as input.

9 Conclusions

Directed Congruence Closure (DCC) has several advantages over other algorithms for working with regular term rewriting systems (systems that are unequivocal, nonoverlapping, and have no repeated variables on the left). In particular, the data structure it uses allows DCC to remember each reduction that is done; thus, no reduction is ever repeated. For finding normal forms, DCC has the advantage that it will often recognize that no normal form exists in cases where other algorithms continue working to find a nonexistent normal form.

For proving the equivalence of terms, DCC retains a complete history of the versions of each term. Other algorithms can retain only a current version of each term; thus, there is the possibility of terms that are equivalent, but the versions that agree are never in memory at the same time. DCC allows all versions of each term to be retained in a compact data structure in which equivalent terms can be easily detected.

A particularly useful concept used in this work is that of the closure of a set of axiom instances. If Δ is a regular set of axiom schemata and if $\mathbf{K}$ is a set of axiom instances of Δ then $\bar{\mathbf{K}}$, the closure of $\mathbf{K}$ with respect to Δ, is the smallest closed set of axiom instances containing $\mathbf{K}$. The properties of $\bar{\mathbf{K}}$ that are most useful are:

1. For any terms A and B, $A =_{\mathbf{K}} B$ iff $A =_{\bar{\mathbf{K}}} B$.
2. $\bar{\mathbf{K}}$ has the confluence property.

This concept is a major tool used both here and in [Ch81] where it is used to determine sufficient conditions for normal forms in term rewriting systems with repeated variables.

References

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass. (1974).

[Che80] L. P. Chew, An improved algorithm for computing with equations, 21st Annual Symposium on the Foundations of Computer Science (1980), 108-117.

[Che81] L. P. Chew, Unique normal forms in term rewriting systems with repeated variables, 13th Annual ACM Symposium on Theory of Computing (1981), 7-18.

[DST80] P. J. Downey, R. Sethi, and R. E. Tarjan, Variations on the common subexpression problem, JACM 27:4 (Oct. 1980), 758-771.

[GB85] J. H. Gallier and R. V. Book, Reductions in tree replacement systems, Theoretical Computer Science 37 (1985), 123-150.

[GHM78] J. V. Guttag, E. Horowitz, and D. R. Musser, Abstract data types and software validation, CACM 21:12 (Dec. 1978), 1048-1064.

[HO82a] C. M. Hoffmann and M. J. O'Donnell, Pattern matching in trees, JACM (1982), 68-95.

[HO82b] C. M. Hoffmann and M. J. O'Donnell, Programming with equations, ACM Transactions on Programming Languages and Systems 4:1 (Jan. 1982), 83-112.

[Hue80] G. Huet, Confluent reductions: abstract properties and applications to term rewriting systems, JACM 27:4 (Oct. 1980), 797-821.

[HL79] G. Huet and J-J. Levy, Call by need computations in nonambiguous linear term rewriting systems, Rapport de Recherche No 359, IRIA/LABORIA (1979).

[HO80] G. Huet and D. C. Oppen, Equations and rewrite rules: a survey, Formal Languages: Perspectives and Open Problems, ed. R. Book, Academic Press (1980), 349-405.

[Klo80] J. W. Klop, Combinatory Reduction Systems, Mathematical Centre Tracts 127, Mathematisch Centrum, Amsterdam (1980).

[KB70] D. E. Knuth and P. B. Bendix, Simple word problems in universal algebra, Computational Problems in Abstract Algebra, ed. J. Leech, Pergammon Press, Oxford (1970), 263-297.

[Koz77] D. Kozen, Complexity of finitely presented algebras, 9th Annual ACM Symposium on Theory of Computing (1977), 164-177.

[NO80] G. Nelson and D. C. Oppen, Fast decision procedures based on congruence closure, JACM 27:2 (April 1980), 356-364.

[O'D77] M. J. O'Donnell, Computing in Systems Described by Equations, Lecture Notes in Computer Science 58, Springer-Verlag (1977).

[O'D85] M. J. O'Donnell, Equational Logic as a Programming Language, MIT Press (1985).

[Ros73] B. K. Rosen, Tree-manipulating systems and Church-Rosser theorems, JACM 20:1 (Jan. 1973), 160-187.

[Tar75] R. E. Tarjan, Efficiency of a good but not linear set union algorithm, JACM 22:2 (April 1975), 215-225.