Normalization via Rewrite Closures

L. Bachmair, C. Ramakrishnan, I. V. Ramakrishnan and A. Tiwari

Department of Computer Science, SUNY at Stony Brook,
Stony Brook, NY 11794, U.S.A
{leo,cram,ram,astiwari}@cs.sunysb.edu

Abstract. We present an abstract completion-based method for finding normal forms of terms with respect to given rewrite systems. The method uses the concept of a rewrite closure, which is a generalization of the idea of a congruence closure. Our results generalize previous results on "non-oblivious" normalization. The presentation simplifies the description and allows a better understanding of known algorithms, apart from extending the results to performing normalization by convergent rewrite systems.

1 Introduction

A term rewriting system R is a finite set of *rewrite* rules, each of the form $l \to r$, where l and r are terms built from *function* symbols in a set Σ and (universally quantified) *variables* in a set $\mathcal{V}$. A rule $l \to r$ can be applied to a term s if a subterm u of s matches l with some substitutions σ for variables appearing in l (i.e., $u = l\sigma$). The rule is applied by replacing the subterm u in s with the corresponding right-hand side $r\sigma$ of the rule, within which the same substitution σ for variables has been made. This process is called *reduction*. A term s can be reduced repeatedly until an irreducible term called its *normal* form is obtained.

Normalization via straight-line reduction are oblivious, *i.e.,* the reductions done are never remembered. For instance if $f(a)$ is reduced to a in n steps then $f(f(a))$ gets reduced to a in $2n$ steps. Observe that since we did not remember the earlier reduction we had to repeat it all over again. In contrast had we remembered it then we could have normalized $f(f(a))$ in $n+1$ steps.

Chew [2] had proposed non-oblivious normalization algorithms. In such algorithms reductions are never repeated. Chew's algorithm though was restricted to the class of orthogonal rewrite systems. His algorithm was based on the congruence closure algorithm developed by Nelson and Oppen [6]. But congruence closure algorithms are used to verify equality of two ground terms with respect to a set of ground equations. So Chew had to extend it to deal with equalities containing variables, represented as a rewrite system. Verma [7] extended Chew's work to deal with priority systems.

Both of the above works can be viewed as an operational approach to doing normalization without repeating reductions. In contrast we present an abstract view of non-oblivious normalization. In particular we describe a completion-based method to do normalization without repeating reductions. Our abstract presentation has several advantages. Firstly it simplifies the description of such normalization algorithms. Secondly the abstract view facilitates a good understanding of the conditions under which one can get completeness of such normalization algorithms. Specifically we show how one can derive Chew's and Verma's algorithms as special cases in our abstract description. Finally using our framework we also obtain non-oblivious normalization algorithms for convergent rewrite systems. Such algorithms were non-existent for this class of rewrite systems. We skip all proofs here. For details, see [1].

1.1 Preliminaries

Let Σ denote a signature consisting of constants and function symbols. We use $\mathcal{T}(\Sigma)$ to denote terms over Σ. The symbols $s, t, u, \cdots$ denote terms; $f, g, \cdots$ denote function symbols; and $x, y, z, \cdots$ denote variables. A subterm of a term t is called *proper* if it is distinct from t. We write $s[t]$ to indicate that a term s contains t as a subterm and (ambiguously) denote by $s[u]$ the result of replacing a particular occurrence of t by u.

A binary relation $\rightarrow$ on terms is *monotonic* (with respect to the term structure) if $s \rightarrow t$ implies $u[s] \rightarrow u[t]$, for all terms s, t and u. It is *stable* (under substitution) if $s \rightarrow t$ implies $s\sigma \rightarrow t\sigma$, for any substitution σ. The symbols $\rightarrow^*$ and $\leftrightarrow$ denote the transitive-reflexive, and symmetric closure of $\rightarrow$, respectively.

An *equation* is a pair (s, t) of terms, written as $s \approx t$. Directed equations are also called *rewrite rules* and are written as $s \rightarrow t$. A *rewrite system* is any set R of rewrite rules. The rewrite relation $\rightarrow_R$ is the smallest stable and monotonic relation that contains R. A term t is in normal form with respect to R (in short, in R-normal form) if there is no term u, such that $t \rightarrow_R u$. We say $s \rightarrow^!_R t$ iff t is a R-normal form of s. Convergent rewriting systems are defined in the usual way.

Let $t[l\sigma] \rightarrow_R t[r\sigma]$ be a (one step) reduction using the rewrite rule $l \rightarrow r \in R$. This reduction will be called a *non-root reduction*, denoted by $\rightarrow^{nr}_R$ if $l\sigma$ is a proper subterm of t; otherwise this is called a *root reduction*, denoted by $\rightarrow^{root}_R$.

Narrowing is the process of looking for an instance of a term that makes a rule applicable (not solely within the substitution part) and then applying that rule. That is, suppose a term $t[s]$ contains a (not necessarily proper) subterm s that can be *unified* with a left-hand side l of a rule $l \rightarrow r \in R$ by the most general substitution σ, i.e., $s\sigma = l\sigma$. Then that substitution is first applied to t - yielding $t[s\sigma]$ - and then the rule is applied - yielding $t[r\sigma]$. If $t[r\sigma]$ is not in R-normal form, it is reduced further, to say t'. The term t is then said to *narrow* to t', denoted by $t_R t'$.

The approach proposed for doing non-oblivious normalization involves the use of the abstract congruence closure, which we describe in section 2. Thereafter, in section 3, we discuss the normalization procedure. This basic procedure is optimized by using a modified congruence closure, called a *rewrite closure*, in section 4. However, this optimization is not general enough and it restricts the applicability of the normalization procedure to finding normal forms only with respect to certain systems. As special cases, we present this for orthogonal systems and convergent systems in section 5.

2 Abstract Congruence Closure

We briefly introduce the concept of an "abstract congruence closure" which is central to all work presented in this paper. Detailed description can be found in [1]. The term "congruence closure" has typically been used to denote certain data structures representing congruence relations induced by set of ground equations. We shall define congruence closures in terms of ground convergent rewrite systems over an extended signature.

A key characteristic of congruence closure algorithms, as pointed out by Kapur [4], is that names are introduced for all given terms and subterms. We may represent such names by constants and specify the correspondence between the new constants and the original terms by rewrite rules.

Definition 1. *Let Σ be a signature and K be a set of constants disjoint to Σ. By a D-*rule *(with respect to Σ and K) we mean a rewrite rule of the form*

$$f(c_1, \ldots, c_k) \to c_0$$

*where $f \in \Sigma$ and $c_0, c_1, \ldots, c_k$ are constants in K. An equation $c \to d$, where c and d are constants in K, is called a C-*rule *(with respect to K). An undirected equation $c \approx d$ is called a C-*equation*.*

For example, let Σ consist of five function symbols, a, b, c, f and g, and let $\mathcal{E}_0$ be a set of three equations $a \approx b$, $gfa \approx fa$ and $gfb \approx c$. To represent each different subterm in $\mathcal{E}_0$ by a new constant, we need a set of D-rules, $D_0 = \{a \to c_0,\ b \to c_1,\ c \to c_2,\ fc_0 \to c_3,\ gc_3 \to c_4,\ fc_1 \to c_5,\ gc_5 \to c_6\}$. Using these D-rules to simplify the original equations in $\mathcal{E}_0$, we obtain a set C_0 of three C-equations, $c_0 \approx c_1$, $c_4 \approx c_3$ and $c_6 \approx c_2$. The C-equations which represent equivalences between terms, can be oriented to give C-rules. These C-rules may also allow one to simplify the presentation of given equations via D-rules and C-rules.

Definition 2. *Let R be a set of D-rules and C-rules (with respect to Σ and K). We say that a constant c in K* represents *a term t in $\mathcal{T}(\Sigma \cup K)$ (via the rewrite system R) if $t \to_R^* c$. A term t is also said to be* represented *by R if it is represented by some constant via R.*

Definition 3. *Let Σ be a signature and K be a set of constants disjoint to Σ. In addition, let $\mathcal{E}$ be a set of ground equations over $\mathcal{T}(\Sigma \cup K)$. A ground rewrite system $R = D \cup C$ is said to be an* (abstract) congruence closure *(with respect to Σ and K)* for *$\mathcal{E}$ if*

(i) D is a set of D-rules, C is a set of C-rules, and if a constant $c \in K$ is in normal form with respect to R, then c represents at least one term $t \in \mathcal{T}(\Sigma)$ via R,

(ii) R is a ground convergent rewrite system over $\mathcal{T}(\Sigma \cup K)$, and

(iii) If s and t are terms over $\mathcal{T}(\Sigma)$, then $s \leftrightarrow_{\mathcal{E}}^* t$ if, and only if, $s \to_R^! \circ \leftarrow_R^! t$.

For example, the constant c_3 represents the term fa via D_0. Furthermore, the rewrite system R_0 above is not a congruence closure for $\mathcal{E}_0$, as it is not a ground convergent rewrite system. But we can transform R_0 into a suitable rewrite system, using a completion-like process described next, to obtain a congruence closure, $R_1 = \{a \to c_1,\ b \to c_1,\ c \to c_6,\ fc_1 \to c_6,\ gc_6 \to c_6\}$, that provides a more compact representation of $\mathcal{E}_0$.

2.1 Construction of Congruence Closures

We next present a general method for construction of congruence closures. Our description is fairly abstract, in terms of transformation (or transition) rules that operate on triples (K, E, R), where K is a set of new constants that have been introduced (the original signature Σ is fixed); E is a set of ground equations (over Σ) yet to be processed; and R is the set of C-rules and D-rules that have been derived so far. Triples represent possible *states* in the process of constructing a closure. The initial state is $(\phi, \mathcal{E}, \phi)$, where $\mathcal{E}$ is the input set of ground equations.

A key transformation rule is the introduction of new constants as names for subterms.

Extension:
$$\frac{(K, E, R)}{(K \cup \{c\}, E, R \cup \{f(c_1, \cdots, c_k) \to c\})}$$

where $f \in \Sigma$, $c_1, \ldots, c_k$ are constants in K, $f(c_1, \ldots, c_k)$ is a term occurring in (some equation in) E, and $c \notin \Sigma \cup K$.[1]

Once a D-rule $f(c_1, \ldots, c_k) \to c$ has been introduced, it can be used to eliminate any occurrence of $f(c_1, \ldots, c_k)$.

Simplification:
$$\frac{(K, E[s], R \cup \{s \to t\})}{(K, E[t], R \cup \{s \to t\})}$$

where s is a subterm of (either the left or the right-hand side of) an equation in E. [2]

It is fairly easy to see that any equation in E can be transformed to a C-equation by suitable extension and simplification steps. The final C-equations are eliminated from E as follows.

Orientation:
$$\frac{(K \cup \{c_1, c_2\}, E \cup \{c_1 \approx c_2\}, R)}{(K \cup \{c_1, c_2\}, E, R \cup \{c_1 \to c_2\})}$$

if $c_1 \succ c_2$.

[1] Some of the transition rules require additional information that is provided by a reduction ordering $\succ$ on terms. (Such an ordering is well-founded and compatible with the given term structure.) We assume that a suitable ordering is supplied initially and is extended appropriately whenever a new constant is introduced. For our purposes, it is sufficient to use a lexicographic path ordering based on a total precedence on symbols in $\Sigma \cup K$, for which $f \succ c$, whenever $f \in \Sigma$ and $c \in K$.

[2] By $E[t]$ we mean the set obtained from E by replacing a single occurrence of s by t in some equation in E.

We need additional transformation rules to obtain a convergent rewrite systems in the third component of a state.

Superposition: $$\frac{(K, E, R \cup \{t \to c, t \to d\})}{(K, E \cup \{c \approx d\}, R \cup \{t \to d\})}$$

if $t \to c$ and $t \to d$ are D-rules.

Additionally, we need (i) a deletion rule to delete trivial equations from the set E; (ii) a collapse rule, to simplify left-hand sides of D rules; and, (iii) a composition rule, to simplify right-hand sides of D rules.

Example 1. Consider the set of equations $\mathcal{E}_0 = \{a \approx b,\ gfa \approx fa,\ gfb \approx c\}^3$. The initial state tuple, therefore, is $\xi_0 = (K_0, E_0, R_0) = (\emptyset, \mathcal{E}_0, \emptyset)$. We show below the intermediate states of one of the possible derivations. When the third component is fully-reduced, we don't show the C-rules.

$(\emptyset,$	$\{a \approx b, gfa \approx fa, gfb \approx c\},$	$\emptyset)$
$(\{c_0, c_1\},$	$\{gfa \approx fa, gfb \approx c\},$	$\{a \to c_0,\ b \to c_1,\ c_0 \to c_1\})$
$(\{c_1\},$	$\{gfa \approx fa, gfb \approx c\},$	$R_1 = \{a \to c_1,\ b \to c_1\})$
$(\{c_1, c_2, c_3\},$	$\{gfb \approx c\},$	$R_1 \cup \{fc_1 \to c_2,\ gc_2 \to c_3,\ c_2 \to c_3\})$
$(\{c_1, c_3\},$	$\{gfb \approx c\},$	$R_2 = R_1 \cup \{fc_1 \to c_3,\ gc_3 \to c_3\})$
$(\{c_1, c_3\},$	$\{\},$	$R_3 = R_2 \cup \{c \to c_3\})$

The set R_3 is the required congruence closure.

Definition 4. *We use the symbol $\vdash_{CC}$ to denote the one-step transformation relation on states induced by the above transformation rules. A* derivation *is a sequence of states $\xi_0 \vdash_{CC} \xi_1 \vdash_{CC} \cdots$. A finite derivation $\xi_0 \vdash_{CC} \xi_1 \vdash_{CC} \cdots \vdash_{CC} \xi_n$ is said to be* maximal *if no further transformation rules can be applied to ξ_n.*

Proposition 1. *(Correctness) Let (K, ϕ, R) be the last state of a maximal derivation beginning from the state $(\phi, \mathcal{E}, \phi)$. If $s, t \in \mathcal{T}(\Sigma)$, then $s \leftrightarrow^*_{\mathcal{E}} t$ iff $s \to^!_R \circ \leftarrow^!_R t$.*

This shows that R is a congruence closure for $\mathcal{E}$. In order to get a maximal derivation, we just need a strategy that is *fair*, i.e., if any transition rule is continuously enabled, then it is eventually applied. The above correctness result is easy to establish, but we skip the proof here [1].

3 Normalization Using Congruence Closure

The problem we address is that of doing efficient non-oblivious normalization: Given some term rewriting system $\mathcal{E}$ (containing variables) and a term t, find the $\mathcal{E}$-normal form of t without repeating a reduction step. We use the abstract congruence closure to avoid processing the same rule instance again and again.

[3] In all examples, new constants will be ordered as follows : $c_i \succ c_j$ if $i < j$.

Example 2. For exposition, consider the simple case when $\mathcal{E}$ is a *ground* rewriting system. Suppose we want to normalize gfb with respect to the directed equations $\mathcal{E}_0 = \{a \to b, gfa \to fa, gfb \to c\}$, see example 1.

Let R_0 denote the (abstract) congruence closure for $\mathcal{E}_0$ computed in that example. To obtain the normal form of gfb from R_0, we first find the R_0-normal form of gfb. This is c_3. Now we need to find a $\mathcal{E}_0$-irreducible term represented by c_3. We note that, $gfb \to^*_{R_0} c_3 \leftarrow^*_{R_0} c$, and also $gfb \to^*_{R_0} c_3 \leftarrow^*_{R_0} fa$.

Thus, c and fa are two terms represented by c_3 which are also $\mathcal{E}_0$-irreducible. We could return either one as the normal form of gfb.[4]

We generalize the above procedure to deal with systems $\mathcal{E}$ containing variables too. Introducing variables causes the complication of picking all the needed (ground) instances from $\mathcal{E}$, for obtaining a normal form for t. We use narrowing for this.

3.1 Narrowing

If the set of directed rules $\mathcal{E}$ contains rules with variables, then it is not immediately clear what instances to process and include in the congruence closure. Usually, given the term t to normalize, one finds an appropriate substitution σ such that for some left-hand side term l_i (of the rules in $\mathcal{E}$), it is the case that $l_i\sigma$ is identical to t or a subterm of t.

In our case, we don't know the intermediate terms explicitly. Since every constant represents some term in the original signature, therefore, we need to just find a substitution σ such that for some left-hand side l_i, $l_i\sigma \to^*_R c$, for some constant c. In other words, we need to check if l_i can be *narrowed* to a constant c via R.

We describe a simplified narrowing procedure, that will be used later, using transition rules. The transition rules are going to be defined on states of the form $(K, R, \mathcal{S})$, where K and R are *inputs*, and $\mathcal{S}$ is the output of the narrowing procedure[5]. The component $\mathcal{S}$ will denote a forest of narrowing trees[6] - one tree for each left-hand side of a rule in $\mathcal{E}$. If $l_i \to r_i$, for $i = 1, \ldots, n$ is the set of all rules in $\mathcal{E}$, $\mathcal{S}$ would be a sequence of n sets, each set containing pairs (l'_i, σ^e_i), where σ^e_i is a substitution[7] and l'_i would be such that $l_i\sigma^e_i \to^*_R l'_i$. Note that we are interested in finding if any l_i can be narrowed to a constant in K, i.e. if there is a σ_i such that $l_i\sigma_i \to^*_R c$, for some $c \in K$. Formally,

$$\textbf{Narrow:} \quad \frac{(K, R, \langle \ldots, s_j \cup \{(C[t], \sigma^e)\}, \ldots\rangle)}{(K, R, \langle \ldots, s_j \cup s'_j, \ldots\rangle)}$$

[4] Correctness issues will be discussed later.

[5] K would denote the set of constants and R would be the current congruence closure.

[6] A narrowing tree for l is obtained by keeping l on the root, and having as children all possible narrowing steps on l using R (as described in the rule above). The j-th component s_j then simply is the set of current leaves in the narrowing tree with root l_j.

[7] Substitutions here are homomorphic extensions of a mapping from $\mathcal{V}$ to $\mathcal{T}(\Sigma \cup K)$. Since the range is the extended signature, we use a superscript e.

where s'_j is the set of all pairs $(C[c]\sigma_1^e, \sigma^e\sigma_1^e)$ such that there is a rule $l \to c$ in R and $t\sigma_1^e \equiv l$. If there is no such rule, then $s'_j = \phi$.

By $\vdash_N$ we denote the one-step transformation relation on states induced by the narrowing transformation rule.

3.2 Normalization

Now we have all the gadgets required to describe the normalization procedure, namely the abstract congruence closure computation relation $\vdash_{CC}$ and the narrowing transformation relation $\vdash_N$. The transformation rules for normalization procedure will use these two relations. We would work on states of the form $(\mathcal{E}; t; (K, E, R); (K', R', \mathcal{S}))$, where we apply the relation $\vdash_{CC}$ to the third component, and the relation $\vdash_N$ to the fourth. The first and the second components are the inputs to the normalization.

The normalization procedure is started by representing the term t to be normalized using D-rules. Since a congruence closure would automatically represent all new terms, we introduce a dummy rule $c \to t$ and add it to an appropriate component.

Initialization: $$\frac{(\mathcal{E}; t; (\phi, \phi, \phi); (\phi, \phi, \Lambda))}{(\mathcal{E}; t; (\{c\}, \{c \to t\}, \phi); (\phi, \phi, \Lambda))}$$

where c is a new constant, and Λ is a sequence of empty sets.

Once we have identified a non-empty set E of (ground) equations to add to the congruence closure, we can perform a step of the congruence closure computation.

Cong. Closure: $$\frac{(\mathcal{E}; t; (K, E, R); (\phi, \phi, \Lambda))}{(\mathcal{E}; t; (K', E', R'); (\phi, \phi, \Lambda))}$$

if $(K, E, R) \vdash_{CC} (K', E', R')$.

Once we have processed all ground equations in E, to continue we need to select new ground instances. The process of selection is done by narrowing, as described before. The following rule initiates this.

Init-Selection: $$\frac{(\mathcal{E}; t; (K, \phi, R); (\phi, \phi, \Lambda))}{(\mathcal{E}; t; (K, \phi, R); (K, R, \langle\{(l_i, \epsilon)\} : i = 1, \ldots, n\rangle))}$$

if $\mathcal{E} = \{l_i \to r_i : i = 1, \ldots, n\}$ and R is fully reduced. The symbol ϵ denotes the empty substitution.

We use narrowing to find out if any instance of a left-hand side of a rule is represented by a constant.

Narrow: $$\frac{(\mathcal{E}; t; (K, \phi, R); (K, R, \mathcal{S}))}{(\mathcal{E}; t; (K, \phi, R); (K, R, \mathcal{S}'))}$$

if $(K, R, \mathcal{S}) \vdash_N (K, R, \mathcal{S}')$.

The result of narrowing can be used to identify new instances to process.

$$\textbf{Selection:} \quad \frac{(\mathcal{E}; t; (K, \phi, R); (K, R, \langle s_1, \ldots, s_j \cup \{(c, \sigma^e)\}, \ldots, s_n\rangle))}{(\mathcal{E}; t; (K, \{l_j\sigma^e \to r_j\sigma^e\}, R); (\phi, \phi, \Lambda))}$$

if $c \in K$. The rule $l_j\sigma^e \to r_j\sigma^e$ which is moved to the set E in this rule, will be called a *selected*, or, *processed* rule[8].

Next we need rules for termination. If all the narrowing trees have been fully explored, and none ends in a constant, then selection rule cannot be applied. This means that there are no ground instances to process and we can terminate[9].

$$\textbf{Terminate1:} \quad \frac{(\mathcal{E}; t; (K, \phi, R); (K, R, \Lambda))}{\Omega}$$

where $\Omega = t^*$ if there is a $t^* \in \mathcal{T}(\Sigma)$ such that $t \to^!_R c \leftarrow^*_R t^*$, and t^* is $\mathcal{E}$ irreducible; otherwise, $\Omega = \bot$.

We can have another termination condition too, in which we terminate once we have identified a normal form of t.

$$\textbf{Terminate2:} \quad \frac{(\mathcal{E}; t; (K, \phi, R); (K, R, \langle s_1, \ldots, s_n\rangle))}{t^*}$$

if (i) there is a $t^* \in \mathcal{T}(\Sigma)$ such that $t \to^!_R c \leftarrow^*_R t^*$, and (ii) t^* is not further reducible by $\mathcal{E}$.[10]

Terminating in a state $\bot$ means that we output "no normal form of t exists", and terminating in a state t^* means that we output t^* as the normal form of t.

Example 3. Consider the problem of normalizing the term fa with respect to the rewrite system $\mathcal{E} = \{a \to b, fX \to gfX, gfb \to c\}$. The initial state obtained by the initialization rule is $\xi_{-1} = (\mathcal{E}; fa; (\{c_{-1}\}, \{c_{-1} \to fa\}, \phi); (\phi, \phi, \Lambda))$. Therefore, as a result of computing the congruence closure, we would have

$$\xi_0 = (\mathcal{E}; fa; (\{c_{-1}, c_0, c_2\}, \phi, \{a \to c_0, fc_0 \to c_2, c_{-1} \to c_2\}); (\phi, \phi, \Lambda)).$$

In order to choose the next instance to process, we apply the "init-selection" rule. This means we apply narrowing transition rules on the state $(\{c_{-1}, c_0, c_2\}, \{a \to$

[8] Any rule which can potentially be selected will be called *selectible*.

[9] The way this has been described here, except for the trivial case when the initial term t is not reducible by any rule in $\mathcal{E}$, this inference rule will never be applicable. But, if we add an additional rule that deletes from $\mathcal{S}$ any pair that has already been processed, or selected, then this rule will be more useful. This would however mean that we somewhere store with each rule in $\mathcal{E}$, the set of substitutions with which it has been instantiated and applied. We, however, do not need this rule for correctness of the procedure, though it definitely improves the termination characteristics of the normalization procedure.

[10] This inference rule can be effectively applied. We simply non-deterministically guess for each constant $d \in K$, a D rule $f(\cdots) \to d$. Then the required t^* is one which reduces to c using these guessed rules, and which satisfies (ii) above.

$c_0, fc_0 \to c_2, c_{-1} \to c_2\}, \langle\{(a,\epsilon)\}, \{(fX,\epsilon)\}, \{(gfb,\epsilon)\}\rangle)$. Note that a narrows to c_0 with the empty substitution. Therefore, the rule $a \to b$ is selectible. The new congruence closure obtained by adding this rule is $R_1 = \{b \to c_1, a \to c_1, fc_1 \to c_2, c_0 \to c_1, c_{-1} \to c_2\}$. Now note that fX can be narrowed to a constant, with the substitution $X \mapsto c_1$. Thus we process the instance $fc_1 \to gfc_1$ next. We get a congruence closure R_2 of example 1 (ignoring C rules). Finally we can choose to process $gfb \to c$ and get R_3 as the new closure. Now we would identify the normal form and terminate, using the rule terminate2, as $fa \to^!_R c_3$ and $c \to^*_R c_3$ and c is not further reducible by $\mathcal{E}$.

3.3 Soundness and Completeness

By $i(\mathcal{E})^e$ we shall denote the set of all ground instances of the rules in $\mathcal{E}$ in the extended signature $\Sigma \cup K$. Let $F \subset i(\mathcal{E})^e$ be the set of processed rule instances of $\mathcal{E}$. When we compute the congruence closure R of a set of rules F, we in essence actually compute the closure of the set F^ρ_R in the original signature, which is defined below.

Definition 5. *Let R be a set of D-rules and C-rules (with respect to Σ and K). If F is a set of equations over $\mathcal{T}(\Sigma \cup K)$, then the* R-extension *of F is defined to be the set F^ρ_R of all equations $s\rho \to t\rho$, where $s \to t$ is an equation in F and ρ is a mapping from K to $\mathcal{T}(\Sigma)$, such that $c \leftrightarrow^*_R c\rho$, for all constants c in K.*

Theorem 1. *(Soundness) Assume that the rewrite system $\mathcal{E}$ is confluent. If a derivation starting from $(\mathcal{E}; t; (\phi,\phi,\phi); (\phi,\phi,\Lambda))$ terminates with normal form t^*, then t^* is the $\mathcal{E}$-normal form of t.*

We next establish completeness of the procedure. First we need to make sure that normal form of t would be eventually represented. For this, we require fairness conditions.

Definition 6. *A derivation is said to be* fair *if either it is finite (i.e. some termination rule has been successfully applied), or, (i) it is always the case that eventually we get to states of the form $(\mathcal{E}; t; (K,\phi,R); (K,R,\langle s_1,\ldots,s_n\rangle))$ where each s_i contains only those pairs whose first element is a constant in K, and, (ii) the rule* terminate2 *is applied whenever possible, and, (iii) any rule instance which can be chosen by the* selection *rule from such states and which has not already been chosen, is eventually selected.*

Essentially fairness says that after some finite number of steps, we always construct the *full* narrowing forest, and make sure every rule instance gets processed, unless of course, we terminate. A fair reduction strategy ensures that enough rule instances are processed so as to guarantee the representation of normal form term.

Theorem 2. *(Completeness) If $t \in \mathcal{T}(\Sigma)$ has a $\mathcal{E}$-normal form then a fair derivation starting from state $(\mathcal{E}; t; (\phi,\phi,\phi); (\phi,\phi,\Lambda))$ terminates in state t^*, where t^* is in $\mathcal{E}$-normal form.*

The procedure described above is a rather straight forward modification of a oblivious normalization algorithm where the derivation tree with root t is constructed in a breadth first manner, in order to avoid getting caught in an infinite branch. The difference is that we work on equivalence classes of terms, rather than individual terms. This however restricts our applicability to confluent systems only.

The procedure may or may not terminate when the term t does not have a normal form. Termination when t does not have a normal form is given by the *terminate1* inference rule. To apply it usefully, we need to store the set of processed instances, and delete from the sets in $\mathcal{S}$ any substitutions that have been processed. This can be formulated as another inference rule, but we do not do so here.

One major drawback of the inference rules for normalization is that the termination checks essentially involve a non-deterministic step. To make this more efficient, we can store some information about which rules to use to find normal form terms in the congruence closure itself. The modified congruence closure will be called a *rewrite closure*.

4 Abstract Rewrite Closure

In order to make the termination checks more efficient, the basic congruence closure procedure requires additional refinements, so that one can determine whether a given represented term is in normal form or not. We will achieve this by *marking* certain rules, or, in other words, we will partition the set D into two sets: marked rules X, and unmarked rules N. The idea would be that terms represented by the left-hand sides of N rules will be F_D^ρ-irreducible. Hence while searching for normal forms in the termination rules, instead of guessing, we will use the N-rules directly.

Definition 7. *An abstract congruence closure $R = D \cup C$ for F is called a* rewrite closure *if, the set D can be partitioned into $N \cup X$ such that for all terms t in $\mathcal{T}(\Sigma)$ represented by D, t is in normal form with respect to F_R^ρ if, and only if, it is represented by N.*

Equivalently, we can also say that a congruence closure $D = N \cup X$ is a rewrite closure for F if for every $t \in \mathcal{T}(\Sigma)$, t is F_R^ρ-irreducible, iff, any reduction sequence starting with t contains only N steps, and no X steps. For example, let $F = \{ffa \to fffa,\ fffa \to fa\}$. If we let N_0 be the set of two rules, $a \to c_0$ and $fc_0 \to c_1$; and X_0 be a set of one rule, $fc_1 \to c_1$, then $N_0 \cup X_0$ is a rewrite closure for F. Rewrite closures need not always exist, though. Consider the set of equations $F' = \{fffa \to ffa, fffa \to fa\}$. We cannot get a rewrite closure from the abstract congruence closure $D_1 = \{a \to c_0,\ fc_0 \to c_1,\ fc_1 \to c_1\}$ for F'. Since a, fa and ffa are all in normal forms, we are forced to have all the D-rules in R_1 in the set N_0.

Note that there are several ways in which the set of D-rules can be partitioned. If $s \to t$ is a rewrite rule in F, then its left-hand side s is called an

F-*redex* (or simply a *redex*). One method is to put all D-rules, whose left-hand sides represent F-redexes, into the set X. Rules in X are therefore also called *redex rules*. We write $s \xrightarrow{n} t$ to indicate that $s \rightarrow t$ is a rule in N, and $s \xrightarrow{x} t$ to indicate that $s \rightarrow t$ is a rule in X. If $f(c_1, \ldots, c_k) \rightarrow c_0$ is a rule in X, then the term $f(c_1, \ldots, c_k)$ is also called a *redex template*. However, using this scheme for marking rules we may not still get a rewrite closure. We need the additional property of *persistence*.

Definition 8. *Let R be a abstract congruence closure for a set of ground rules F over $\mathcal{T}(\Sigma \cup K)$. The set F is said to have the* persistence *property with respect to R if whenever, there exist terms $f(t_1, \cdots, t_n), f(t'_1, \cdots, t'_n) \in \mathcal{T}(\Sigma)$ such that, $f(t_1, \cdots, t_n)$ is F_R^ρ-reducible at the top (root) position, and $f(t_1, \cdots, t_n) \leftrightarrow^{*,nr}_{F_R^\rho} f(t'_1, \cdots, t'_n)$, it is always the case that, $f(t'_1, \cdots, t'_n)$ is F_R^ρ-reducible.*

The idea behind the persistence property is simple. Since we put every redex-template in the set X, this simply means that we assume that all the terms represented by that template are reducible. The persistence property is true whenever this is actually the case.

Lemma 1. *Let F be a finite set of equations over $\mathcal{T}(\Sigma \cup K)$. A congruence closure R of F can be extended to a rewrite closure if F has the* persistence *property with respect to R.*

The converse of this theorem is however false, as the set $F = \{a \rightarrow b,\ fa \rightarrow c,\ c \rightarrow fb\}$ is *not* persistent (with respect to its abstract congruence closure), but the congruence closure can be extended to a rewrite closure.

4.1 Construction of Rewrite Closures

We give a set of transition rules (similar to the ones for congruence closure), that would compute the rewrite closure for a given F,[11] assuming that the persistence property holds.

The *extension* inference rule, which introduces new constants as names for subterms, is the same as before except that now it creates N-rules. We have to be a little careful in simplification rules, as we cannot simplify at the top of the left hand side.

$$\textbf{Simplification:} \quad \frac{(K, F[s], R \cup \{s \rightarrow t\})}{(K, F[t], R \cup \{s \rightarrow t\})}$$

where s is either a subterm of a right-hand side of a rule in F, or else a *proper* subterm of a left-hand side.Note that only proper subterms of left-hand sides of rules in F can be replaced.

[11] It should be mentioned here that the set F which contains ground equations over $\mathcal{T}(\Sigma \cup K)$ is dynamically generated in our application to normalization. That is the reason why some constants from K that are introduced by the congruence closure procedure appear in the set F.

It is fairly easy to see that any rewrite rule in E can be transformed to a D-rule by suitable extension and simplification steps. The final D-rules are eliminated from F as follows.

$$\textbf{Orientation:} \qquad \frac{(K, F \cup \{f(c_1, \cdots, c_k) \to c\}, R)}{(K, F, R \cup \{f(c_1, \cdots, c_k) \xrightarrow{x} c\})}$$

if $f(c_1, \ldots, c_k) \to c$ is a D-rule with respect to Σ and K. Note that orientation generates a *redex* rule.

In the superposition rule too, we have to be careful with the markings.

$$\textbf{Superposition:} \qquad \frac{(K, F, R \cup \{t \xrightarrow{\alpha_1} c, t \xrightarrow{\alpha_2} d\})}{(K, F, R \cup \{t \xrightarrow{\alpha_3} d, c \to d\})}$$

if (i) $t \to c$ and $t \to d$ are D-rules, (ii) $c \succ d$, and (iii) α_3 is n only in the case when both α_1 and α_2 are n; in all other cases, α_3 is x.[12]

The other rules like deletion and collapse can be similarly formulated. Let us illustrate the computation of a rewrite closure with a simple example.

Example 4. Consider the set of rules $\mathcal{E}_0 = \{a \to b, fb \to gfb, gfb \to c\}$. Beginning with $\xi_0 = (\emptyset, \mathcal{E}_0, \emptyset)$ as initial state, we use the same strategy as we used in example 1. The only difference is that now we also keep track of the markings.

$(\emptyset,$	$\{a \to b, fb \to gfb, gfb \to c\},$	$\emptyset)$
$(\{c_0\},$	$\{fb \to gfb, gfb \to c\},$	$\{a \xrightarrow{x} c_0,\ b \xrightarrow{n} c_0\})$
$(\{c_0, c_1, c_2\},$	$\{gfb \to c\},$	$R_1 \cup \{fc_0 \xrightarrow{n} c_1,\ gc_1 \xrightarrow{n} c_2,\ fc_0 \xrightarrow{x} c_2\})$
$(\{c_0, c_2\},$	$\{gfb \approx c\},$	$R_1 \cup \{fc_0 \xrightarrow{x} c_2,\ gc_2 \xrightarrow{n} c_2\})$
$(\{c_0, c_2, c_3\},$	$\{\},$	$R_2 \cup \{c \xrightarrow{n} c_3,\ gc_2 \xrightarrow{x} c_3\})$

If we fully reduce the last component, we get $\{a \xrightarrow{x} c_0,\ b \xrightarrow{n} c_0,\ fc_0 \xrightarrow{x} c_3,\ gc_3 \xrightarrow{x} c_3,\ c \xrightarrow{n} c_3\}$, which is a rewrite closure of $\mathcal{E}_0$.

The correctness proofs are exactly similar to those for the congruence closure. Using the results from the last section we also know that this procedure constructs a rewrite closure whenever F satisfies the persistence property. We use the symbol $\vdash_{RC}$ to denote the one-step transformation relation on states induced by the above transformation rules.

4.2 Normalization Using Rewrite Closure

The normalization procedure described earlier, can now be optimized by replacing the inference rule *congruence-closure* by a step *rewrite-closure*. The additional marking information in the rewrite closure can be used to optimize the *terminate1* and the *terminate2* inference rules: Essentially all we are saying is

[12] In other words, $t \to d$ is a redex rule in the new state if, and only if, at least one of the two superposed rules is a redex rule.

that in order to find a normal form in the equivalence class c, we need to check for only those term t' such that $t' \rightarrow_N^* c$. The only additional condition needed for correctness is that the processed rules satisfy the persistence property. The important point to take note of is that in order to do *selection*, we still have to construct the narrowing forest using *all* D rules, and not just N rules, which would have been desirable. This is because the completeness proof otherwise doesn't go through.

We just mention those inference rules of the normalization procedure which now look different. The *congruence-closure* inference rule is replaced by the following rewrite-closure rule.

Rewrite-Closure: $$\frac{(\mathcal{E}; t; (K, E, R); (\phi, \phi, \Lambda))}{(\mathcal{E}; t; (K', E', R'); (\phi, \phi, \Lambda))}$$

if $(K, E, R) \vdash_{RC} (K', E', R')$.

The *init-selection*, *narrow* and *selection* rules are the same as in section 3. Next we need rules for termination.

Terminate1: $$\frac{(\mathcal{E}; t; (K, \phi, R); (K, R, \Lambda))}{\Omega}$$

where $\Omega = t^*$ if there is a $t^* \in \mathcal{T}(\Sigma)$ such that $t \rightarrow_R^! c \leftarrow_N^* t^*$; otherwise, $\Omega = \perp$.

Terminate2: $$\frac{(\mathcal{E}; t; (K, \phi, R); (K, R, \langle s_1, \ldots, s_n \rangle))}{t^*}$$

if (i) there is a $t^* \in \mathcal{T}(\Sigma)$ such that $t \rightarrow_R^! c \leftarrow_N^* t^*$, and (ii) t^* is in $\mathcal{E}$-normal form.

The soundness theorem 1 holds under these changes. In order to prove completeness of the method, we need to be able to get a rewrite closure for the set F of *processed* instances of $\mathcal{E}$. To make sure that normal form of t would be eventually represented, we require fairness conditions exactly like before.

Theorem 3. *(Completeness) If $t \in \mathcal{T}(\Sigma)$ has a $\mathcal{E}$-normal form t^* then a fair derivation (starting from state $(\mathcal{E}; t; (\phi, \phi, \phi); (\phi, \phi, \Lambda))$) in which the processed rule instances F is always (eventually) persistent, terminates in state t^*.*

The conditions of *fairness* and *persistence* are complementary. In order to satisfy *persistence*, we should process fewer and only particular rules. On the other hand, to satisfy fairness we are required to process as many rules as possible. For example, informally, an innermost strategy in choosing instances to process shall always process sets of instances that are persistent. But, unfortunately, such a strategy may violate fairness. In the next section, we consider two special cases of rewrite systems $\mathcal{E}$ where we can effectively satisfy both conditions together and use the normalization transition rules to find normal forms.

5 Further Optimizations: Special Cases

Next we introduce one further optimization, and show that normalization in orthogonal systems and in convergent systems can be done using this new set of inference rules.

It appears wasteful that we use all of the D-rules in the narrowing process. For computing normal forms, typically we just choose to process rule instances that reduce current *irreducible* terms. So, one would conjecture that we just need to use the N-rules during narrowing, as N-rules represent irreducible terms. In fact the soundness result still holds even in the presence of this restriction. The problem is in the proof of completeness where we want to claim that t^* is eventually represented (and that we are not caught in an infinite branch). But in special cases, we can carry out this proof, and hence use only the N-rules to construct the narrowing forest.

$$\textbf{Init-Selection:} \qquad \frac{(\mathcal{E}; t; (K, \phi, N \cup X \cup C); (\phi, \phi, \Lambda))}{(\mathcal{E}; t; (K, \phi, N \cup X \cup C); (K, N, \langle \{(l_i, \epsilon)\} : i = 1, \ldots, n \rangle))}$$

if $\mathcal{E} = \{l_i \to r_i : i = 1, \ldots, n\}$ and R is fully reduced.

Essentially this means that we only use the N rules for performing narrowing. If this is the case, then the second termination check can be further optimized as follows.

$$\textbf{Terminate2:} \qquad \frac{(\mathcal{E}; t; (K, \phi, R); (K, R, \langle s_1, \ldots, s_n \rangle))}{t^*}$$

if (i) no further Narrow transitions can be applied, and (ii) there is a $t^* \in \mathcal{T}(\Sigma)$ such that $t \to_R^! c \leftarrow_N^* t^*$, and (iii) none of the constants that appear in the derivation $t^* \to_N c$ occur in (as the first component in) any element of $s_1, \ldots, s_n$.

As mentioned before, for soundness and completeness results to hold under these modifications, all we need is (i) the confluence of $\mathcal{E}$, (ii) persistence of the processed set of rules, and (iii) a strategy that ensures that the normal form term is eventually represented. We shall see that all these can be ensured for two special cases below.

5.1 Normalization in Orthogonal Systems

In this subsection, we consider the problem of performing normalization of terms with respect to a special kind of rewriting systems, called orthogonal systems. The correctness proof is simple and clear in our formulation. For details, see [1]. A term rewriting system $\mathcal{E}$ is *orthogonal* if the reduction rules of $\mathcal{E}$ are *left-linear* and there are *no critical overlaps.* We use the following well-known result.

Lemma 2. *[5] Every orthogonal term rewriting system is confluent.*

Next we note that irrespective of what subset of instances are processed, the persistence property is always satisfied.

Lemma 3. *Let $\mathcal{E}$ be an orthogonal system. Let t be root reducible (by an instance of the rule $l \to r \in \mathcal{E}$) and also reducible to t' at a non-root position by some rule in $\mathcal{E}$. Then t' is root-reducible by an instance of the rule $l \to r$.*

Using the previous two lemmas, it can established that irrespective of the strategy chosen to select the next instance to process, the processed instances is confluent, and persistent. Hence, we can conclude the following.

Theorem 4 ((Correctness)). *If $\mathcal{E}$ is an orthogonal rewriting system, then given any term and a fair strategy, the inference system outlined above finds its normal form with respect to $\mathcal{E}$, if one exists.*

5.2 Normalization in Convergent Systems

In order to perform normalization of terms with respect to convergent systems, we need a strategy such that we can satisfy the persistence property and ensure that the normal form term gets represented. Intuitively, for convergent systems, the normal form term will always eventually get represented under any strategy for choosing the next instance, as convergent systems are confluent and *terminating*. Using an innermost strategy allows us to satisfy persistence.

Definition 9. *Let $l\sigma^e \to r\sigma^e$ and $l'\sigma'^e \to r'\sigma'^e$ be two different (extended) instances of rules in $\mathcal{E}$ that are selectible. Say $l\sigma^e \to_N^* c$ and $l'\sigma'^e \to_N^* c'$. An* innermost strategy *is one that makes sure that if there exists a term $t[c]$ containing c such that $t[c] \to_N^* c'$, then the rule $l\sigma^e \to r\sigma^e$ is chosen first.*

Lemma 4. *Suppose that in a derivation, we choose the next instance to process using an innermost strategy. If we choose to process the instance $l\sigma^e \to r\sigma^e$ at some point, then for every proper subterm t of $l\sigma^e$, there exists a constant c such that $t \to^!_{N_l \cup C_l} c$ always eventually. (assuming constants introduced earlier are smaller than constants introduced later).*

When $\mathcal{E}$ is convergent, by performing an induction on the number of applications of the *selection* rule, we can show that an (i) innermost strategy is unambiguous; (ii) the processed set of rules satisfy the persistence property whenever rules are chosen using an innermost strategy; (iii) the R-extension of the processed rules is convergent; and (iv) each new constant in normal form represents exactly one term in $\mathcal{T}(\Sigma)$ via N-rules. Once this result is proven, it is straightforward to establish the correctness result.

Theorem 5 ((Correctness)). *Let $\mathcal{E}$ be a convergent rewriting system. Then given any term, the inference system outlined above finds its normal form.*

6 Conclusion

The problem of normalization using a given set of rewrite rules is fundamental to efficient implementations of symbolic simplifiers. Oblivious strategies for performing normalization were studied first by Hoffmann and O'Donnell [3]. The simple straight line reduction methods were replaced by efficient variants. Chew [2] proposed combining the earlier known straight line reduction strategy with congruence closure algorithm (in order to avoid repeating reduction steps). The advantages of using a slightly modified congruence closure to do normalization were also pointed out there. But Chew's procedure worked only for orthogonal systems. His work was further refined and generalized in Verma and Ramakrishnan [8] and Verma [7].

The idea of abstract congruence closure presented here helps us to see the various different congruence closure algorithms in a generalized framework [13]. The extension to the idea of a rewrite closure is fairly natural and simple. The simplified presentation allows to give straightforward proofs for the results of Chew and Verma. In contrast, it is said in Verma [7] that the proof of completeness of the normalization method is *long and intricate*. We dispense away with the concept of strong closure which is so forcefully used in proofs there.

Our results are strictly more general than the ones known before. The best known result, that appears in Verma [7], gives six postulates that a rewrite relation $\mathcal{E}$ should satisfy in order to prove completeness of a rewrite-closure based procedure for normalization. The three postulates relevant for standard rewrite relations (the other three are for priority rewrite systems which are also considered in that work, but are not dealt with here) *imply* confluence of $\mathcal{E}$ and *non-overlapping* of rules in $\mathcal{E}$. This means our results are more general. Furthermore, Verma uses his results to show that the rewrite-closure based procedure could be used to find normal forms with respect to *consistent* convergent systems. This he has to do by translating such a system to a priority rewrite system. In contrast, our approach yields a direct and more general method for normalization in convergent systems.

References

[1] L. Bachmair, C. Ramakrishnan, I.V. Ramakrishnan, and A. Tiwari. *Normalization via Rewrite Closure (Full Version)*. `http://www.cs.sunysb.edu/ astiwari/-rta-full.ps`, 1998.

[2] L. P. Chew. *Normal forms in term rewriting systems*. PhD thesis, Purdue University, 1981.

[3] C. M. Hoffmann and M. J. O'Donnell. Programming with equations. *Transactions on Programming Languages and Systems*, 4(1):83–112, 1982.

[4] D. Kapur. Shostak's congruence closure as completion. In H. Comon, editor, *Proc. 8th Intl. RTA*, pages 23–37, 1997. LNCS 1232, Springer, Berlin.

[5] J. W. Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, chapter 6, pages 2–116. Oxford University Press, Oxford, 1992.

[6] G. Nelson and D. Oppen. Fast decision procedures based on congruence closure. *JACM*, 27(2):356–364, 1980.

[7] R. M. Verma. A theory of using history for equational systems with applications. *JACM*, 42:984–1020, 1995.

[13] This aspect is however not discussed in this paper.

[8] R. M. Verma and I. V. Ramakrishnan. Nonoblivious normalization algorithms for nonlinear systems. In *Proc. of the Int. Colloquium on Automata, Languages and Programming*, New York, 1990. Springer-Verlag.