On the Combination of Congruence Closure and Completion

Christelle Scharff[1]* and Leo Bachmair[2]

[1] Department of Computer Science, Pace University, NY, USA, cscharff@pace.edu
[2] Department of Computer Science, SUNY Stony Brook, NY, USA, leo@cs.sunysb.edu

Abstract. We present a graph-based method for constructing a congruence closure of a given set of ground equalities that combines the key ideas of two well-known approaches, completion and abstract congruence closure, in a natural way by relying on a specialized and optimized version of the more general, but less efficient, SOUR graphs. This approach allows for efficient implementations and a visual presentation that better illuminates the basic ideas underlying the construction of congruence closures and clarifies the role of original and extended signatures and the impact of rewrite techniques for ordering equalities.

1 Introduction

Theories presented by finite sets of ground (i.e., variable-free) equalities are known to be decidable. A variety of different methods for solving word problems for ground equational theories have been proposed, including algorithms based on the computation of a congruence closure of a given relation. Efficient congruence closure algorithms have been described in [5, 8, 10, 12]. These algorithms typically depend on sophisticated, graph-based data structures for representing terms and congruence relations.

A different approach to dealing with ground equational theories is represented by term rewriting [1, 4], especially the completion method [7] for transforming a given set of equalities into a convergent set of directed rules that defines unique normal forms for equal terms and hence provides a decision procedure for the word problem of the underlying equational theory. Completion itself is a semi-decision procedure but under certain reasonable assumptions about the strategy used to transform equalities, is guaranteed to terminate if the input is a set of ground equalities. Completion methods are not as efficient as congruence closure, though an efficient ground completion method has been described by [13], who obtains an $O(n\ log(n))$ algorithm that cleverly uses congruence closure to transform a given set of ground equalities into a convergent ground rewrite system. Standard completion is quadratic in the worst case [11].

We combine the two approaches in a novel way, different from [6], and present an efficient graph-based method that combines the key ideas of completion and abstract congruence closure [2, 3]. Our approach employs a specialized version of the SOUR graphs that were developed for general completion [9]. In SOUR graphs the vertices represent terms and the edges carry information about subterm relations between terms (S), rewrite rules (R), unifiability of terms (U) and order relations between terms (O). In the application to congruence closure we consider only ground terms and hence do not need unification edges. Moreover, general term orders are too restrictive as well as too expensive to maintain, and hence we

* This work is supported by the National Science Foundation under grant ITR-0326540.

also dispense with order edges and manipulate rewrite edges in a different way, based on the explicit use of edges (E) representing unordered equalities. Thus, our modifications amount to what might be called "SER graphs." This modified, and simplified, graph structure provides a suitable basis for computing congruence closures. We represent terms and equalities by a directed graph that supports full structure sharing. The vertices of the graph represent terms, or more generally equivalence classes of terms; edges represent the subterm structure of the given set of terms, as well as equalities and rewrite rules.

Some of the graph transformation rules we use are simpler versions of SOUR graph rules and, in logical terms, correspond to combinations of critical pair computations and term simplifications. We also include an explicit "merge rule" that is well-known from congruence closure algorithms, but only implicitly used in SOUR graphs. Exhaustive application of these transformation rules terminates, is sound in that the equational theory represented over Σ-terms does not change, and complete in that the final rewrite system over the extended signature is convergent.

The main difference between our approach and the abstract congruence closure framework of [3] is that the graph-based formalism naturally supports a term representation with full structure sharing, which can not be directly described in an abstract, inference-based framework, though the effect of graph transformation rules can be indirectly simulated by suitable combinations of inference rules.

The efficiency of our method crucially depends on the use of a simple ordering (that needs to be defined only on the set of constants extending the original term signature), rather than a full term ordering. The corresponding disadvantage is that we do not obtain a convergent rewrite system over the original signature, but only over an extended signature. However, we may obtain a convergent rewrite system on the original signature by further transforming the graph in a way reminiscent of the compression and selection rules of [2, 3].

We believe that our approach allows for a visual presentation that better illuminates the basic ideas underlying the construction of congruence closures. In particular, it clarifies the role of original and extended signatures and the impact of rewrite techniques for ordering equalities. It should also be suitable for educational purposes.

The graph-based construction of a congruence closure is described in Section 3. In Section 4 we show how to obtain a convergent rewrite system over the original signature. Section 5 contains examples. In Section 6 we discuss the influence of the ordering on the efficiency of our method and present complexity results. A full version of this article with proofs and more examples is available at: `http://www.csis.pace.edu/~scharff/CC`.

2 Preliminaries

We assume the reader is familiar with standard terminology of equational logic and rewriting. Key definitions are included below, more details can be found in [1, 4]. In the following, let Σ and $\mathcal{K}$ be disjoint sets of function symbols. We call Σ the (basic) *signature*, and $\Sigma \cup \mathcal{K}$ the *extended signature*. The elements of $\mathcal{K}$ are assumed to be constants, and are denoted by subscripted letters c_i ($i \geq 0$).

Flat ground terms. The height H of a term is recursively defined by: if t is a variable, then $H(t) = 0$, otherwise $H(t) = H(f(t_1, \ldots, t_n)) = 1 + max\{H(t_1), \ldots, H(t_n)\}$. A term is said to be *flat* if its height is 2 at most. We will consider flat ground terms, i.e., variable-free terms t with $H(t) \leq 2$.

D-rules, C-rules, C-equalities. A *D-rule* on $\Sigma \cup \mathcal{K}$ is a rewrite rule $f(c_1, \ldots, c_n) \to c_0$, where $f \in \Sigma$ is a function symbol of arity n and $c_0, c_1, \ldots, c_n$ are constants of $\mathcal{K}$. A *C-rule* on $\Sigma \cup \mathcal{K}$ (respectively, a *C-equality*) is a rule $c_0 \to c_1$ (respectively, an equality $c_0 \approx c_1$), where c_0 and c_1 are constants of $\mathcal{K}$.

The constants in $\mathcal{K}$ will essentially serve as names for equivalence classes of terms. Thus, an equation $c_i \approx c_j$ indicates that c_i and c_j are two names for the same equivalence class. A constant c_i is said to *represent* a term $t \in \mathcal{T}(\Sigma \cup \mathcal{K})$ via a rewrite system R, if $t \leftrightarrow_R^* c_i$.

Abstract congruence closure: A ground rewrite system $R = D \cup C$ of D and C-rules on $\Sigma \cup \mathcal{K}$ is called an *abstract congruence closure* if: (i) each constant $c_0 \in \mathcal{K}$ represents some term $t \in \mathcal{T}(\Sigma)$ via R and (ii) R is convergent. If E is a set of ground equalities over $\mathcal{T}(\Sigma \cup \mathcal{K})$ and R an abstract congruence closure such that for all terms s and t in $\mathcal{T}(\Sigma)$, $s \leftrightarrow_E^* t$ if, and only if, there exists a term u with $s \to_R^* u \leftarrow_R^* t$, then R is called an *abstract congruence closure of* E. That is, the word problem for an equational theory E can be decided by rewriting to normal form using the rules of an abstract congruence closure R of E.

3 Graph-Based Congruence Closure

We describe a graph-based method for computing an abstract congruence closure of a given set of ground equalities E over a signature Σ. First a directed acyclic graph (DAG) is constructed that represents the set E, as well as terms occurring in E. In addition, each vertex of this initial graph is labeled by a distinct constant c_i of $\mathcal{K}$. Next various graph transformation rules are applied that represent equational inferences with the given equalities. Specifically, there are four *mandatory* rules (*Orient*, *SR*, *RRout*, and *Merge*) and one optional rule (*RRin*). Exhaustive application of these rules, or *saturation*, will under certain reasonable assumptions yield an abstract congruence closure. The vertices of each (initial and transformed) graph represent equivalence classes of terms. Transformed graphs need not be acyclic, but will conform to full structure sharing in that different vertices represent different terms (or equivalence classes). The transformation rules crucially depend on an ordering $\succ$ on $\mathcal{K}$.[3]

3.1 Initial Graphs

We consider directed graphs where each vertex v is labeled by (i) a function symbol of Σ, denoted by $Symbol(v)$, and (ii) a constant of $\mathcal{K}$, denoted by $Constant(v)$. In addition, edges are classified as *equality*, *rewrite*, or *subterm* edges. We write $u -_E v$ and $u \to_R v$ to denote equality and rewrite edges (between vertices u and v), respectively. Subterm edges are also labeled by an index, and we write $u \to_S^i v$. Informally, this subterm edge indicates that v represents the i-th subterm of the term represented by u.

An *initial* graph $DAG(E)$ represents a set of equalities E as well as the subterm structure of terms in E. It is characterized by the following conditions: (i) If $Symbol(v)$ is a constant, then v has no outgoing subterm edges; and (ii) if $Symbol(v)$ is a function symbol of arity n, then there is exactly one edge of the form $v \to_S^i v_i$, for each i with $1 \leq i \leq n$. (That is, the number of outgoing vertices from v reflects the arity of $Symbol(v)$.)

[3] An ordering is an irreflexive and transitive relation on terms. An ordering $\succ$ is total, if for any two distinct terms s and t, $s \succ t$ or $t \succ s$.

The term $Term(v)$ *represented* by a vertex v is recursively defined as follows: If $Symbol(v)$ is a constant, then $Term(v) = Symbol(v)$; if $Symbol(v)$ is a function symbol of arity n, then $Term(v) = Symbol(v)(Term(v_1), \ldots, Term(v_n))$, where $v \rightarrow_S^i v_i$, for $1 \leq i \leq n$. Evidently, $Term(v)$ is a term over signature Σ. We require that distinct vertices of $DAG(E)$ represent different terms. Moreover, we insist that $DAG(E)$ contain no rewrite edges and that each equality edge $u -_E v$ correspond to an equality $s \approx t$ of E (with u and v representing s and t, respectively), and vice versa.

The vertices of the graph $DAG(E)$ also represent flat terms over the extended signature $\Sigma \cup \mathcal{K}$. More specifically, if $Symbol(v)$ is a constant, then $ExtTerm(v) = Constant(v)$, and if $Symbol(v)$ is a function symbol of arity n, then

$$ExtTerm(v) = Symbol(v)(Constant(v_1), \ldots, Constant(v_n)),$$

where $v \rightarrow_S^i v_i$, for $1 \leq i \leq n$.

We should point out that the labels $Constant(v)$ allow us to dispense with the extension rule of abstract congruence closure [2,3]. The initial graph $DAG(E)$ contains only subterm and equality edges. Rewrite edges are introduced during graph transformations. The term representation schema for transformed graphs is also more complex, see Section 4.

3.2 Graph Transformation

We define the graph transformations by rules. The first rule, *Orient*, can be used to replace an equality edge, $v -_E w$, by a rewrite edge, $v \rightarrow_R w$, provided $Constant(v) \succ Constant(w)$. If the ordering $\succ$ is total, then every equality edge for which $Constant(v) \neq Constant(w)$ can be replaced by a rewrite edge (one way or the other).

The ordering $\succ$ needs to be defined on constants in $\mathcal{K}$ only, not on terms over $\Sigma \cup \mathcal{K}$. Term orderings, as employed in SOUR-graphs, are inherently more restrictive and may be detrimental to the efficiency of the congruence closure construction. For instance, well-founded term orderings must be compatible with the subterm relation, whereas we may choose an ordering with $Constant(v) \succ Constant(w)$ for efficiency reasons, even though $Term(v)$ may be a subterm of $Term(w)$.

The *SR* rule replaces one subterm edge by another one. In logical terms it represents the simplification of a subterm by rewriting, or in fact the simultaneous simplification of all occurrences of a subterm, if the graph presentation encodes full structure sharing for terms.

The *RRout* and *RRin* each replace one rewrite edge by another. They correspond to certain equational inferences with the underlying rewrite rules (namely, critical pair computations and compositions, which for ground terms are also simplifications). The *RRin* rule is useful for efficiency reasons, though one can always obtain a congruence closure without it. If the rule is applied exhaustively, the resulting congruence closure will be a right-reduced rewrite system over the extended signature.

The *Merge* rule collapses two vertices that represent the same term over the extended signature into a single vertex. It ensures closure under congruence and full structure sharing.

The graph transformation rules are formally defined as pairs of tuples of the form $(E_s,\ E_e,\ E_r,\ V,\ \mathcal{K},\ KC) \rightarrow (E_s',\ E_e',\ E_r',\ V',\ \mathcal{K}',\ KC')$, where the individual components specify a graph, an extended signature, and an ordering on new constants, before and after rule application. Specifically,

- the first three components describe the sets of subterm, equality, and rewrite edges, respectively;
- the fourth component describes the set of vertices;
- the fifth component describes the extension of the original signature Σ; [4] and
- the last component describes the (partial) ordering on constants. Specifically, KC is a set of "ordering constraints" of the form $\{c_i \succ c_j \mid c_i,\ c_j \in \mathcal{K}\}$. (A set of such constraints is considered *satisfiable* if there is an irreflexive, transitive relation on $\mathcal{K}$ that meets all of them.)

The specific conditions for the various rules are shown in Figures 1 and 2. For example, if two vertices v and w represent the same flat term (over the extended signature), then the merge rule can be used to delete one of the two vertices, say v, and all its outgoing subterm edges. All other edges that were incident on v need to be redirected to w, with the proviso that outgoing rewrite edges have to be changed to equality edges.

Construction of a congruence closure starts from an initial tuple $(E_s,\ E_e,\ E_r,\ V,\ \mathcal{K}, \emptyset)$, the first five components of which are derived from $DAG(E)$ (so that E_r is empty).[5] Transformation rules can then be applied nondeterministically. The rules *SR*, *RRout* and *RRin* are only applied if they result in a *new* edge.

There are various possible strategies for orienting equality edges. One may start with a fixed, total ordering on constants, or else construct an ordering "on the fly." Different strategies may result in different saturated graphs, see Example 2. The choice of the ordering is crucial for efficiency reasons, as discussed in section 6. The ordering also prevents the creation of cycles of with equality or rewrite edges, though the *SR* rule may introduce self-loops or cycles involving subterm edges. (Such a cycle would indicated that a term is equivalent to one of its subterms in the given equational theory.) See section 3.3 and example 2 for more details.

Definition 1. *We say that a graph G is* saturated *if it contains only subterm and rewrite edges and no further graph transformation rules can be applied.*

3.3 Extraction of Rules

We can extract D-rules, C-rules and C-equalities from the initial and transformed graphs as follows. A vertex v_0 with $ExtTerm(v_0) = t$ and $Constant(v_0) = c_0$ induces a D-rule $t \to c_0$. An equality edge $v_1 -_E v_2$ induces a C-equality $c_1 \approx c_2$, where $Constant(v_1) = c_1$ and $Constant(v_2) = c_2$. A rewrite edge $v_1 \to_R v_2$ induces a C-rule $c_1 \approx c_2$, where $Constant(v_1) = c_1$ and $Constant(v_2) = c_2$.

With each tuple $(E_s,\ E_e,\ E_r,\ V,\ \mathcal{K},\ KC)$ we associate a triple $(\mathcal{K}, Ex, Rx)$, where Ex is the set of C-equalities and Rx is the set of D and C-rules extracted from the graph G specified by the first four components of the given tuple. Thus, with the initial graph we associate a triple $(\mathcal{K}_0, Ex_0, Rx_0)$, where Rx_0 is empty and Ex_0 represents the same equational theory over Σ-terms as the given set of equations E. The goal is to obtain a triple $(\mathcal{K}_n, Ex_n, Rx_n)$, where Ex_n is empty and Rx_n is an abstract congruence closure of E.

[4] We have $\mathcal{K}' \subseteq \mathcal{K}$, which is different from abstract congruence closure [2, 3], where new constants can be introduced.

[5] We essentially begin with the same graph as the Nelson-Oppen procedure, as described in the abstract congruence closure framework [2, 3].

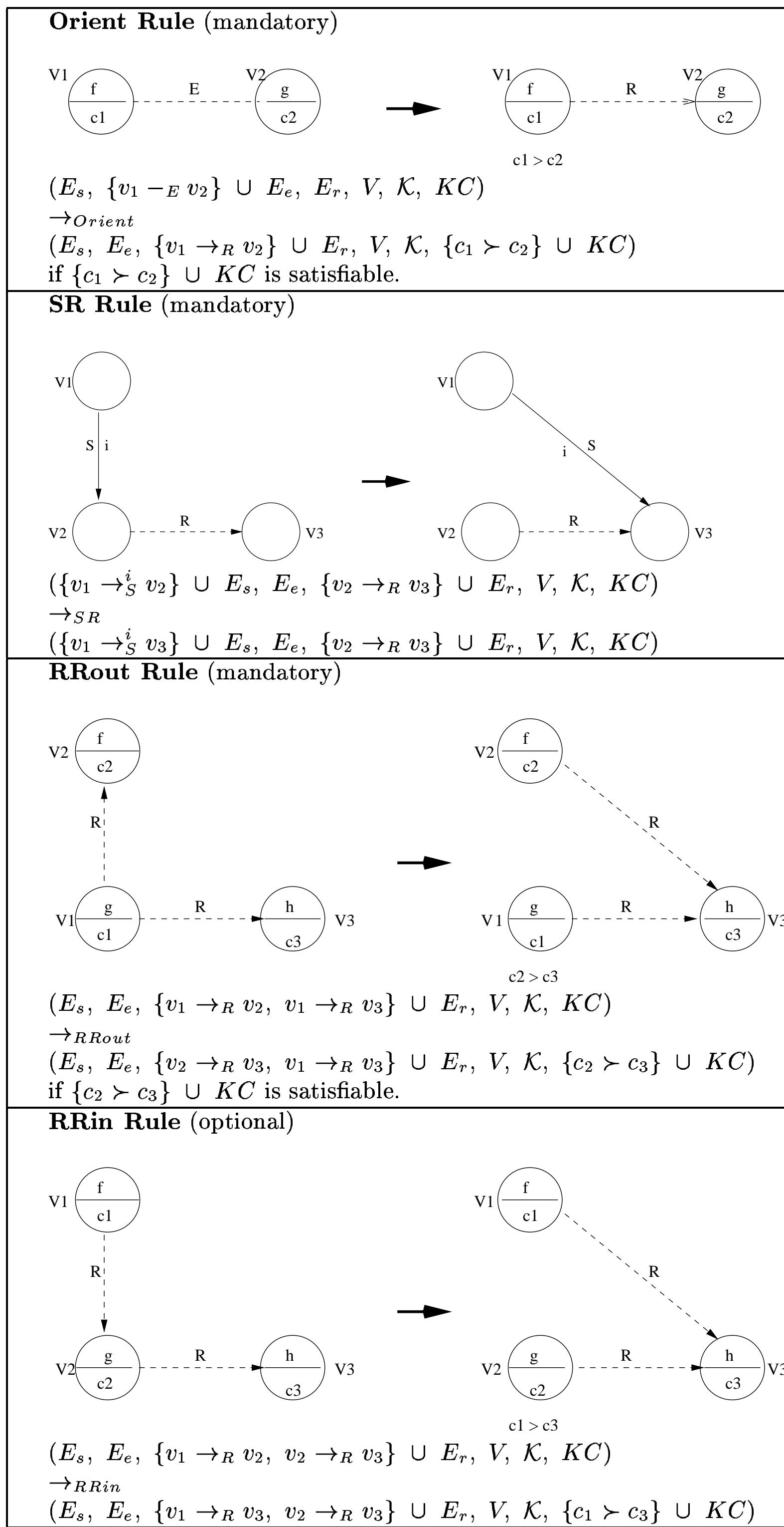

Fig. 1. *Orient*, *SR*, *RRout* and *RRin* graph transformation rules

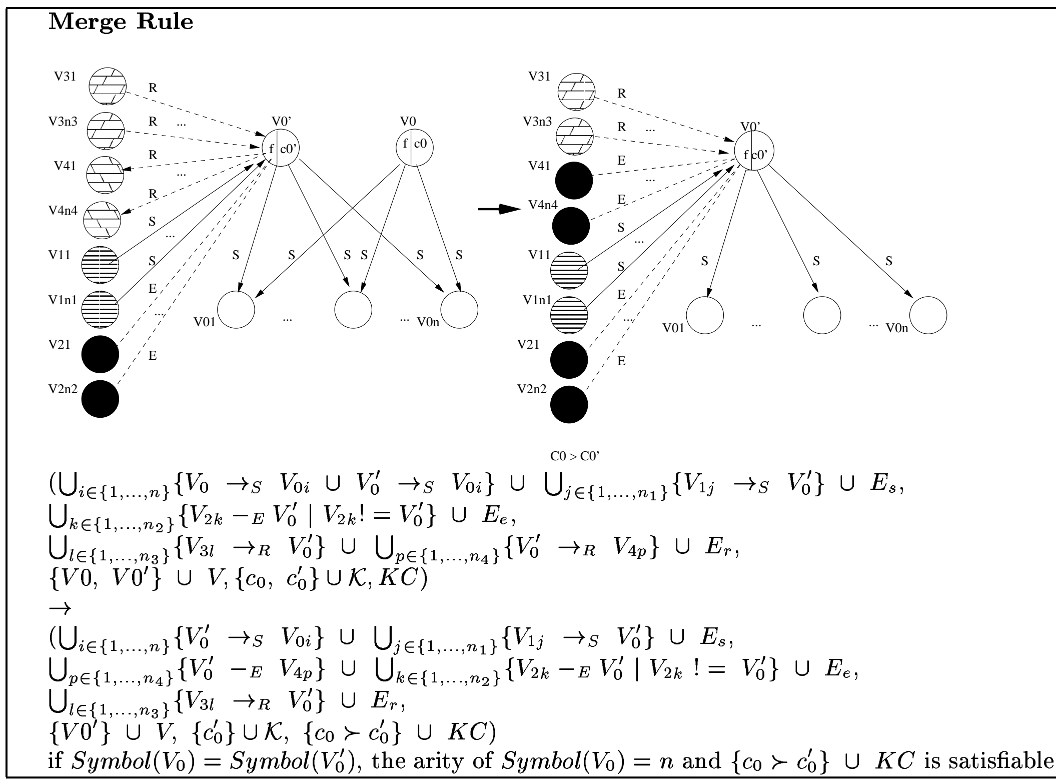

Fig. 2. *Merge* graph transformation rule

3.4 Correctness

The following can be established:

- Exhaustive application of the graph transformation rules is *sound* in that the equational theory represented over Σ-terms does not change.
- Exhaustive application of the graph transformation rules *terminates*. This can be proved by assigning a suitable weight to graphs that decreases with each application of a transformation rule.
- Exhaustive application of the rules is *complete* in that the rewrite system that can be extracted from the final graph is convergent and an abstract congruence closure for E. (If the optional *RRin* rule has been applied exhaustively, the final rewrite system over the extended signature is right-reduced.)

4 Rewrite System over the Original Signature

In this section we explain how to obtain a convergent rewrite system over the original signature Σ (independent from the ordering on constants of $\mathcal{K}$) from a graph G saturated with respect to *Orient*, *SR*, *RRout*, *RRin* and *Merge*. Basically, at this point, constructing

the convergent rewrite system over Σ from G consists of eliminating the constants of $\mathcal{K}$ from G. Indeed, the constants of $\mathcal{K}$ are "names" of equivalence classes, and the same equivalence class may have several "names." There are two methods. The first method works on the convergent rewrite system over $\Sigma \cup \mathcal{K}$ extracted from G. Redundant constants (constants appearing on the left-hand side of a C rules) are eliminating by applications of *compression* rules and non-redundant constants are eliminated by *selection* rules as described in [2, 3]. The second method, that we propose in this article, permits us to work directly and only on the graph G, by transforming the graph in a way reminiscent of the compression and selection rules. We define the notion of redundant constants on a saturated graph with respect to *Orient*, *SR*, *RRout*, *RRin* and *Merge*, and introduce three mandatory graph transformation rules: *Compression*, *Selection 1* and *Selection2*. These inference rules eliminate constants from $\mathcal{K}$, and redirect R and S edges. They also remove cycles of S edges from the graph. Their exhaustive nondeterministic application produces a DAG representing a convergent rewrite system over the original signature. The graph data structure permits us to visualize in parallel what happens on the original and extended signatures. For example, the *Selection 1* rule redirects R edges that were oriented in the "wrong" way during the abstract congruence process. Indeed, if we need to redirect an R edge, it means that the ordering on the constants of $\mathcal{K}$ that was used or constructed "on the fly" is in contradiction with any ordering on terms of $\mathcal{T}(\Sigma)$.

4.1 Graph-based Inference Rules

Definition 2. *A constant c_0 of $\mathcal{K}$ labeling a vertex with an incident outgoing R edge is called* redundant.

The *Compression*, *Selection 1* and *Selection 2* rules are provided in Figure 3. Redundant constants of $\mathcal{K}$ are eliminated by the *Compression* rule. Both *Selection 1* and *Selection 2* implement the original selection rule of [2, 3]. *Selection 1* is implemented by finding an R edge from a vertex v representing a term t of $\mathcal{T}(\Sigma)$ (i.e. all the vertices reachable from v following S edges are labeled by constants of Σ only) to a vertex labeled by a non redundant constant c_0 of $\mathcal{K}$ on the graph, inversing the direction of this R edge, redirecting all the incoming S and R edges incident to c_0 to v, and eliminating c_0 from the graph. Hence, it consists of picking a representative term t over Σ for the equivalence class of c_0. *Selection 2* is described as follows. If a vertex v is labeled by a constant c_0 of $\mathcal{K}$, and all the vertices reachable from v following S edges are labeled by constants of Σ only, then the constant c_0 of $\mathcal{K}$ is eliminated from the graph. A particular case of this rule occurs when a vertex is labeled by a constant c_0 of $\mathcal{K}$ and a constant c of Σ.

4.2 Correctness

When applying *Compression*, *Selection 1* and *Selection 2*, the graph is in its maximal structure sharing form. It can be represented by a state $(\mathcal{K}, R)$, where $\mathcal{K}$ is the set of constants disjoint from Σ labeling the vertices of the graph, and R is the set of rewrite rules read from the graph over $\Sigma \cup \mathcal{K}$. We use the symbol $\vdash$ to denote the one-step transformation relation on states induced by *Compression*, *Selection 1*, and *Selection 2*. A *derivation* is a sequence of states $(\mathcal{K}_0, R_0) \vdash (\mathcal{K}_1, R_1) \vdash \dots$, and $\mathcal{K}_i \subseteq \mathcal{K}_j$ for $i \geq j \geq 0$. We call a state $(\mathcal{K}, R)$ *final*, if no *mandatory* transformation rules (*Compression*, *Selection 1*, *Selection 2*) are applicable to this state. We prove that the *Compression*, *Selection 1* and *Selection 2* are sound in that

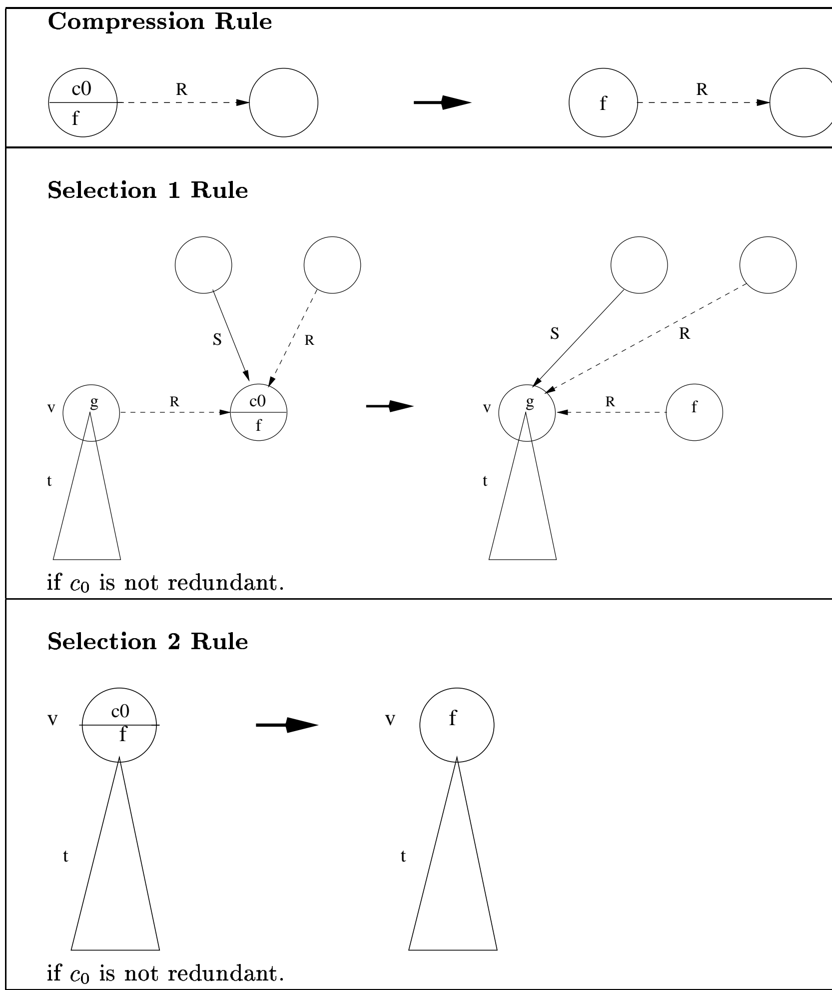

Fig. 3. *Compression*, *Selection 1* and *Selection 2* graph transformation rules

the equational theory represented over Σ-terms does not change. If there is a constant of $\mathcal{K}$ labeling a vertex of the graph, then either *Compression*, *Selection 1* or *Selection 2* can be applied, and the exhaustive application of *Compression*, *Selection 1* and *Selection 2* terminates. The termination is easily shown because the application of *Compression*, *Selection 1* or *Selection 2* reduces the number of constants of $\mathcal{K}$ by one. The final state of a derivation describes a DAG labeled by constants of Σ only, that does not contain cycles of S edges, and, represents a convergent rewrite system over Σ.

5 Examples

Example 1. Figure 4 a) presents the construction of $DAG(E)$, where $E = \{f(f(f(a))) \approx a,\ f(f(a)) \approx a,\ g(c,c) \approx f(a),\ g(c,h(a)) \approx g(c,c),\ c \approx h(a),\ b \approx m(f(a))\}$. $\mathcal{K} =$

$\{c_1, \ldots, c_{10}\}$. We apply all the following mandatory transformations on $DAG(E)$ in a certain order constructing the order on the constants of $\mathcal{K}$ "on the fly." *Orient* orients the E edge between c_1 and c_4 into an R edge from c_4 to c_1 ($c_4 \succ c_1$), and the E edge between c_1 and c_3 into an R edge from c_1 to c_3 ($c_1 \succ c_3$). We can apply an *SR* transformation that replaces the S edge from c_2 to c_3 by an S edge from c_2 to c_1. Let $c_2 \succ c_4$. We merge c_2 and c_4; c_2 is removed from the graph, the S edges between c_2 and c_3 are removed, the E edge from c_6 to c_2 is replaced by an E edge from c_6 to c_3, and a self-loop composed of an S edge is added from c_3 to itself. c_4 and c_3 can be merged, and c_4 is removed from the graph ($c_4 \succ c_3$). The S edge between c_4 and c_3 is removed from the graph, and the R edge from c_4 to c_1 is simply removed, because there is already an R edge from c_1 to c_3. We orient the E edge between c_{10} and c_9 from c_{10} to c_9 ($c_{10} \succ c_9$), the E edge between c_7 and c_5 from c_7 to c_5 ($c_7 \succ c_5$), and the E edge between c_6 and c_3 from c_6 to c_3 ($c_6 \succ c_3$), and the E edge between c_8 and c_6 from c_8 to c_6 ($c_8 \succ c_6$). We can apply an *SR* transformation that adds an S edge from c_8 to c_5, and removes the S edge from c_8 to c_7. c_8 and c_6 are merged, and c_8 is removed from the graph. The R edge between c_8 and c_6 is removed from the graph. We obtain the saturated graph labeled by $\{c_1, c_3, c_5, c_6, c_7, c_9, c_{10}\}$ such that $\{c_6 \succ c_3, c_7 \succ c_5, c_1 \succ c_3, c_{10} \succ c_9\}$ presented in Figure 4 b). The convergent rewrite system over the extended signature is: $\{c_7 \to c_5,\ c_6 \to c_3,\ c_1 \to c_3,\ c_{10} \to c_9,\ c \to c_5,\ a \to c_1,\ b \to c_9,\ h(c_3) \to c_7,\ g(c_5, c_5) \to c_6,\ f(c_3) \to c_3,\ m(c_3) \to c_{10}\}$. By applying *Compression* for the redundant constants c_{10}, c_7, c_6 and c_1, *Selection 2* to eliminate c_5 and c_9, and *Selection 1* to eliminate c_3, we obtain $\{h(a) \to c,\ g(c, c) \to a,\ f(a) \to a,\ m(a) \to b\}$, the convergent rewrite system over Σ that we read from 4 c).

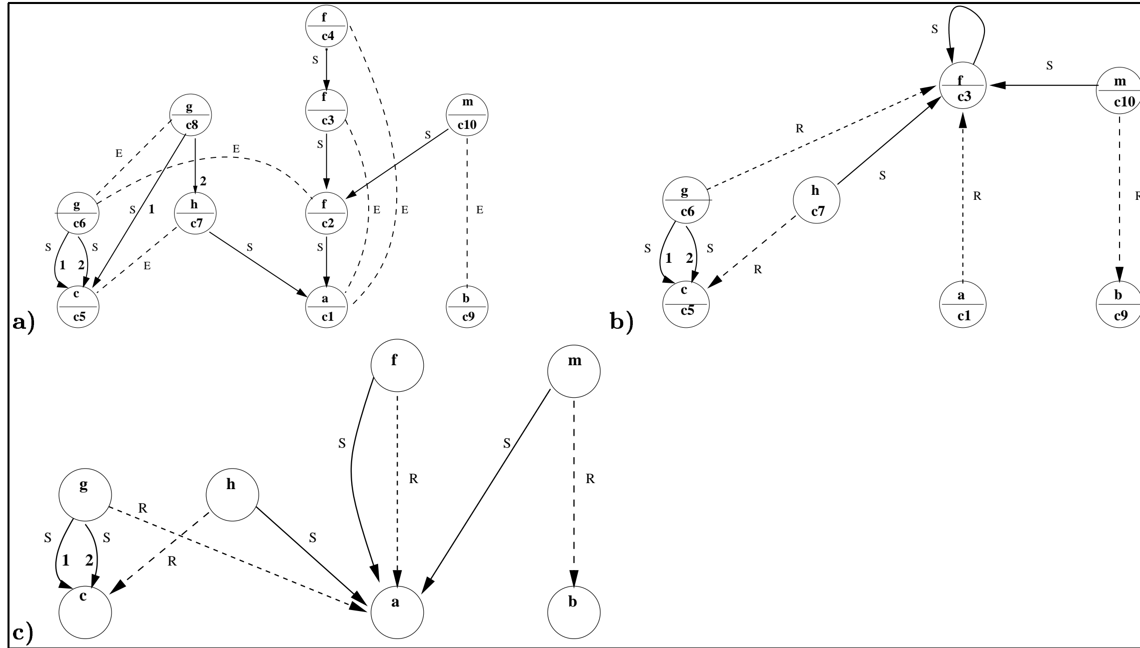

Fig. 4. a) Initial DAG for $E = \{f(f(f(a))) \approx a,\ f(f(a)) \approx a,\ g(c, c) \approx f(a),\ g(c, h(a)) \approx g(c, c),\ c \approx h(a),\ b \approx m(f(a))\}$, b) Saturated graph for E on the extended signature, c) DAG on the original signature

Example 2. Figure 5 presents the constructions of a) $DAG(E)$ where $E = \{f(a,b) \approx a\}$, and its saturated counter-parts following two strategies. In b), $c_3 \succ c_1$ and in c), $c_1 \succ c_3$. In c), there is a self-loop composed of an S edge. The two saturated graphs with respect to *Orient*, *SR*, *RRout*, *RRin* and *Merge* are labeled by $\{c_1, c_2, c_3\}$. In b), the rewrite system on the extended signature is $\{f(c_1, c_2) \to c_3,\ c_3 \to c_1,\ a \to c_1,\ b \to c_2\}$, and in c), it is $\{f(c_3, c_2) \to c_3,\ c_1 \to c_3,\ a \to c_1,\ b \to c_2\}$. Both saturated graphs generate the DAG d) on the original signature representing $\{f(a,b) \to a\}$.

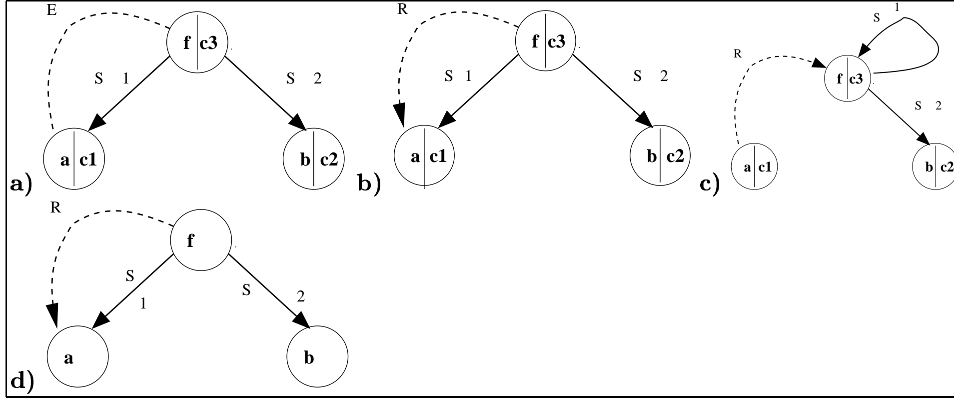

Fig. 5. a) Initial DAG for $E = \{f(a,b) \approx a\}$, b) Saturated graph for E with $c_3 \succ c_1$, c) Saturated graph for E with $c_1 \succ c_3$, and d) DAG on the original signature

6 Complexity Results and Implementation

Our algorithms to obtain rewrite systems over extended and original signatures use only polynomial space to store the rewrite systems. Let n be the number of vertices of the initial DAG. During the abstract congruence process, there can only be n^2 subterm edges, and $n(n-1)/2$ equality and rewrite edges, because an edge can only be added once to a graph, and there are never two equality or rewrite edges between the same vertices (because of the ordering on the constants of $\mathcal{K}$.) Moreover, the number of vertices can only decrease (as a result of merging).

In comparing our graph-based approach with the logic-based approach of [2, 3], we find that graphs support full structure sharing and consequently our approach will tend to lead to fewer applications of transformation rules than corresponding applications of logical inference rules in the standard method. This by itself does not imply that our approach will be more efficient as full structure sharing depends on systematic application of the *Merge* rule, which can be expensive.

Efficient implementations of congruence closure require specialized data structures. Also, abstract congruence closure efficiency is parametrized by the choice or construction "on the fly" of the ordering on the constants of $\mathcal{K}$. In our case, E edges are oriented into R edges using this ordering. There exist naive ways to choose the ordering. For example, we can use a

total and linear ordering on the constants of $\mathcal{K}$. There is a tradeoff between the effort spent in constructing an ordering, the time spent in comparing two constants, and the lengths of derivations (number of graph transformations). The graph data structure permits us to understand the influence of the ordering on the length of any derivation, and to construct "on the fly" orderings that control the number of inferences. It suggests new efficient procedures for constructing efficient abstract congruence closures. Indeed, the number of times we apply *SR* and *RRout* depends on how we apply *Orient*. Applying an *RRout* rule can create an *SR* configuration; so we would orient an E edge in the direction that creates the less SR and *RRout* configurations. *Merge* configurations can be created after the application of an SR rule.

When constructing the ordering, we do not allow backtracking, so we are only interested in *feasible* orderings, i.e. orderings that will produce *unfailing* derivations, i.e. derivations terminating with a saturated graph (as defined in definition 1). Starting from an initial state representing the initial DAG, the maximal derivation is in $O(n\delta)$, where n is the number of vertices of the initial DAG, and δ is the depth of the ordering (i.e. the longest chain $c_0 \succ c_1 \succ \ldots \succ c_\delta$.) So, any maximal derivation starting from an initial state is bounded by a quadratic upper bound. Indeed, any total and linear order is feasible and can be used. There exits a feasible ordering with smaller depth that can be computed "on the fly," and produces a maximal derivation of length $n\ log(n)$ [2, 3]. This ordering is based on orienting an E edge from v_0 to v_1 if the number of elements of the equivalence class of $Constant(v_0)$ is less than or equal to the number of elements of the equivalence class of the $Constant(v_1)$. This can be applied only if all *RRout* and *RRin* inference rules have been processed for v_0 and v_1. The number of elements in an equivalence class of a constant $Constant(v)$ is computed by counting the number of incoming R edges in a vertex v.

A first implementation of our method is available for online experimentation at:

`http://www.csis.pace.edu/~scharff/CC.`

It is written in java, and uses java servlets, and XML. The implemented system contains (i) a parsing component that also transforms a given set of equalities into a DAG, (ii) a graph component that manages the graph, and (iii) an inferences component that deals with the application of the transformation rules. In particular, the strategy for application of rules is coded in XML, and therefore, is modifiable. An efficient strategy applies simplifications before orienting equalities (one at a time):

$(Orient\ .\ ((SR\ .\ Merge)^*)^*\ .\ (RRout\ .\ RRin\ .\ ((SR\ .\ Merge)^*)^*)^*)^*$ [6]

The ordering that is used in the implementation is a total and linear ordering. Given a signature and a set of equalities, the system displays the initial C equalities and D rules, the convergent rewrite system over the extended signature and the convergent rewrite system over the original signature. Some statistics results are also provided: the number of inferences of each type, and the processing times.

7 Conclusion and Future Work

We have presented a new graph-based method for constructing a congruence closure of a given set of ground equalities. The method combines the key ideas of two approaches,

[6] * means that the strategy is applied 0 or more times. $X\ .\ Y$ means that Y is applied after X.

completion and standard congruence closure, in a natural way by relying on a data structure, called "SER graphs," that represents a specialized and optimized version of the more general, but less efficient, SOUR-graphs. We believe that our approach allows for efficient implementations and a visual presentation that better illuminates the basic ideas underlying the construction of congruence closures. In particular it clarifies the role of original and extended signatures and the impact of rewrite techniques for ordering equalities. Our approach should therefore be suitable for educational purposes.

A first implementation of our method is available. The method we described processes all equalities at once during construction of the initial graph. It is relatively straightforward to devise a more flexible, incremental approach, in which equalities are processed one at a time. Once the first equality is represented by a graph, transformation rules are applied until a "partial" congruence closure has been obtained. Then the next equation is processed by extending the current graph to represent any new subterms, followed by another round of graph transformations. This process continues until all equations have been processed. An advantage of the incremental approach is that simplifying graph transformations can be applied earlier. We expect to make implementations of both incremental and non-incremental available.

References

[1] Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press (1998).

[2] Bachmair, L., Ramakrishnan, I.V., Tiwari, A., Vigneron, L.: Congruence Closure Modulo Associativity and Commutativity. 3rd Intl Workshop FroCoS 2000, Kirchner, H. and Ringeissen, C. Editors, Lecture Notes in Artificial Intelligence, Springer-Verlag, **1794** (2000) 245–259.

[3] Bachmair, L., Tiwari, A., Vigneron, L.: Abstract congruence closure. J. of Automated Reasoning, Kluwer Academic Publishers **31(2)** (2003) 129–168.

[4] Dershowitz, N., Jouannaud, J.-P.: Handbook of Theoretical Computer Science, Chapter 6: Rewrite Systems. Elsevier Science Publishers **B** (1990) 244–320.

[5] Downey, P. J., Sethi, R.,Tarjan, R. E.: Variations on the common subexpression problem. Journal of the ACM **27(4)** (1980) 758–771.

[6] Kapur, D.L: Shostak's congruence closure as completion. Proceedings of the 8th International Conference on Rewriting Techniques and Applications, H. Comon Editor, Lecture Notes in Computer Science, Springer-Verlag **1232** (1997).

[7] Knuth, D. E., Bendix, P. B.: Simple word problems in universal algebras. Computational Problems in Abstract Algebra, Pergamon Press, Oxford (1970) 263–297.

[8] Kozen, D.: Complexity of Finitely Presented Algebras. PhD. thesis. Cornell University (1977).

[9] Lynch, C., Strogova, P.: SOUR graphs for efficient completion. Journal of Discrete Mathematics and Theoretical Computer Science **2(1)** (1998) 1–25.

[10] Nelson, C. G., Oppen, D. C.: Fast Decision Procedures based on Congruence Closure. Journal of the ACM **27(2)** (1980) 356–364.

[11] Plaisted, D.A., Sattler-Klein, A.: Proof lengths for equational completion. Journal of Inf. Comput., Academic Press, Inc. **125(2)** (1996) 154–170.

[12] Shostak, R.E.: Deciding Combinations of Theories. Journal of the ACM **31** (1984) 1–12.

[13] Snyder, W.: A Fast Algorithm for Generating Reduced Ground Rewriting Systems from a Set of Ground Equations. Journal of Symbolic Computation **15** (1993) 415–450.