J. Symbolic Computation (1989) 7, 427–444

On the Relationship of Congruence Closure and Unification*

PARIS C. KANELLAKIS PETER Z. REVESZ

Department of Computer Science, Brown University, P.O. Box 1910, Providence, Rhode Island 02912, U.S.A.

(Received 18 February 1988)

Congruence closure is a fundamental operation for symbolic computation. Unification closure is defined as its directional dual, i.e., on the same inputs but top-down as opposed to bottom-up. Unifying terms is another fundamental operation for symbolic computation and is commonly computed using unification closure. We clarify the directional duality by reducing unification closure to a special form of congruence closure. This reduction reveals a correspondence between repeated variables in terms to be unified and equalities of monadic ground terms. We then show that: (1) single equality congruence closure on a directed acyclic graph, and (2) acyclic congruence closure of a fixed number of equalities, are in the parallel complexity class *NC*. The directional dual unification closures in these two cases are known to be log-space complete for *PTIME*. As a consequence of our reductions we show that if the number of repeated variables in the input terms is fixed, then term unification can be performed in *NC*; this extends the known parallelizable cases of term unification. Using parallel complexity we also clarify the relationship of unification closure and the testing of deterministic finite automata for equivalence.

1 Introduction

Congruence closure and unification are fundamental notions in symbolic computation. The unification of terms is the basic operation for most logic programming languages (Lloyd 1984) and the congruence closure of equalities among terms is a central pattern matching task in all systems which compute with equations (Huet & Oppen 1980; Nelson & Oppen 1980; Oppen 1980). In this paper we clarify the relationship between these two notions.

All problems we examine here have polynomial time sequential algorithms, (i.e., they are in the complexity class *PTIME*). Our analysis and comparisons are based on the theory of parallel algorithms and complexity. Let us briefly mention the few but central concepts that we use from this theory. *The complexity class NC* (Pippenger 1979) contains those problems solvable on a *PRAM* (Fortune 1978) in polylogarithmic parallel time using a polynomial number of processors. Intuitively, *NC* consists of those problems whose solution can be significantly sped-up using a multiprocessor. It has been shown that $NC \subseteq PTIME$ and it is strongly conjectured that this containment is proper. A

*This research was supported partly by NSF grant IRI-8617344 and partly by ONR-DARPA grant N00014-83-K-0146, ARPA Order No. 4786. The work of the first author was also supported by an Alfred P. Sloan Foundation Fellowship.

0747–7171/89/030427+18 $03.00/0

problem is *log-space complete for PTIME* if it is in *PTIME* and every problem in *PTIME* is reducible to it using only logarithmic auxiliary space. Any log-space reduction can be computed in *NC* and hence (unless the unlikely fact $PTIME \subseteq NC$ is true) problems log-space complete for *PTIME* do not have *NC* algorithms. Intuitively, problems that are log-space complete for *PTIME* are inherently sequential. A prototypical such problem is the *circuit value problem* (Ladner 1975). The class NC^2 is the subclass of *NC* restricted to log-squared parallel time.

In formalizing the notions of "congruence" and "unification" we follow Downey *et al.* (1980). The two definitions we use exhibit a certain directional duality on the same inputs, namely, congruence closure is defined bottom-up and unification closure top-down.

Let $G = (V, A)$ be a directed graph such that for each vertex v in G, the successors of v are ordered. Let C be any equivalence relation on V. The *congruence closure* CC and the *unification closure*[1] UC of C are the finest equivalence relations on V that contain C and satisfy the following properties for all vertices v and w in G:

Let v and w have successors $v_1, v_2, \ldots, v_k$ and $w_1, w_2, \ldots, w_l$, respectively. If $k = l \geq 1$ and $(v_i, w_i) \in CC$ for $1 \leq i \leq k$, then $(v, w) \in CC$.

Let v and w have successors $v_1, v_2, \ldots, v_k$ and $w_1, w_2, \ldots, w_l$, respectively. If $k = l \geq 1$ and $(v, w) \in UC$, then $(v_i, w_i) \in UC$ for $1 \leq i \leq k$.

Congruence closure is common in decision procedures for formal theories, where it is necessary to determine equivalent expressions. An important use is in solving the following expression equivalence problem, which is called the *uniform word problem for finitely presented algebras:* "determine whether an equality $t_1 = t_2$ logically follows from a set of equalities $S = \{t_{11} = t_{12}, t_{21} = t_{22}, \cdots, t_{k1} = t_{k2}\}$, where the t's are ground terms constructed from constant and function symbols". For this application the directed graph G is a representation of the t's and therefore an acyclic graph.

If the set of equalities S above is empty, we have the well-known *common subexpression elimination* problem, which occurs often in compiling. If the set of equalities S above contains only a *single equality* we have a problem that is relevant to our exposition and that arises in verifying a class of array assignment programs in Downey & Sethi (1978). If the set of equalities S above is fixed and therefore not part of the input we have the (nonuniform) *word problem for finitely presented algebra S*. As shown in Kozen (1977), the uniform word problem for finitely presented algebras is log-space complete for *PTIME*, even when there is only a unique constant and a unique *binary* function symbol in the input terms.

Several authors have suggested algorithms for congruence closure. Downey *et al.* (1980) have the fastest known sequential algorithms for various cases of congruence closure. Their algorithm for the general case requires $O(N log N)$ time, where N is the input size. They also provide $O(N)$ and therefore optimal sequential time algorithms for two cases that are of interest to us here: (1) congruence closure when G is a directed acyclic graph and C contains a single pair of distinct vertices, (2) congruence closure when we get an acyclic graph from G if we contract the equivalence classes of CC.

Unification closure is the directional dual of congruence closure and has a number of important applications. It can be used in testing equivalence of finite automata (Hopcroft

[1] Congruence closure is the terminology used in Downey *et al.* (1980). Unification closure is slightly different from "unifier" defined in Downey *et al.* (1980) and is terminology introduced here to emphasize the directional duality.

& Karp 1971) and in determining a most general set of substitutions (i.e., a most general unifier) to make two terms equal (Martelli & Montanari 1982; Paterson & Wegman 1978; Robinson 1965). The technique of Hopcroft & Karp (1971) combined with the fast *UNION-FIND* method of Tarjan (1975) provides an $O(N\alpha(N))$ time algorithm for unification closure, where $\alpha(N)$ is a functional inverse of Ackermann's function. Huet (1976) and Robinson (1975) independently provided similar bounds for computing most general unifiers. Paterson & Wegman (1978) have given an $O(N)$ time algorithm for the case where we get an acyclic graph G if we contract the equivalence classes of UC; the acyclicity condition here is critical for the linear-time behavior.

Let us briefly comment on the relationship of computing unification closures and computing most general unifiers. Given two terms constructed out of variables, constants and function symbols, the problem of computing the *most general unifier, mgu* is: "finding a most general substitution, if it exists, which makes the two terms equal". One way to compute the mgu is to first compute a unification closure and then test it for two conditions, called homogeneity and acyclicity in Paterson & Wegman (1978). If the acyclicity test is omitted then we have a most general unifier that permits infinite terms as substitutions (mgu^{∞}). Both homogeneity and acyclicity are testable in NC and determine if the mgu exists. Therefore from a parallel complexity point of view the unification closure is the operation of greater interest.

Computing unification closure is shown to be log-space complete in *PTIME* in Dwork *et al.* (1984) and Yasuura (1983). This lower bound is strengthened in Dwork *et al.* (1988). Parallel algorithms for unification closure and a number of its NC^2 subcases are examined in (Auger 1985; Dwork *et al.* 1988; Ramesh *et al.* 1987; Vitter & Simons 1986).

> The main contribution of this paper is in clarifying the directional duality between congruence closure and unification closure (Theorems 3.1 and 4.1). Based on this duality we extend the class of unification problems known to be in NC (Theorem 5.1). We also clarify the relationship of unification closure and deterministic finite automata equivalence (Theorem 6.1).

We first log-space reduce unification closure to congruence closure. Given that both problems were known to be log-space complete in *PTIME*, such a reduction was in principle possible. The particular reduction that we use, however, has some nice properties that accurately capture the directional duality. In Theorem 3.1 we reduce computing the mgu$^{\infty}$ of two terms to the uniform word problem for *monadic* finitely presented algebras. Multiple occurrences of variables in the terms are transformed into algebra axioms. If k = (number of occurrences of variables in the terms) - (number of distinct variables in the terms), then the uniform word problem has $1 + k$ axioms. This reduction and the lower bounds in (Dwork *et al.* 1984; Dwork *et al.* 1988) extend the log-space completeness results of Kozen (1977) to uniform word problems for terms constructed out of one constant and two monadic function symbols. This is syntactically tight because we also show that for terms constructed out of any number of constants and one monadic function symbol the uniform word problem is in NC. This can be shown to follow from the proofs in Auger (1985). We simplify these proofs to a large degree and extend them from the *mgu* to the mgu^{∞} cases (Proposition 3.4). If the uniformity condition is removed we also have word problems in NC (Proposition 3.6). This is based on the theory of finite tree automata of Thatcher & Wright (1968) and also follows from the properties of context free languages (Ruzzo 1980).

We next restrict our attention to inputs consisting of a directed acyclic graph G and an equivalence relation C defined by k pairs of distinct vertices. Let us call these restricted problems *dag-CC[k axioms]* and *dag-UC[k axioms]* respectively. As noted there is a practical application for *dag-CC[1 axiom]* Downey & Sethi (1978). In Theorem 4.1 we show that the problem *dag-CC[1 axiom]* is in NC^2, whereas it is known that *dag-UC[1 axiom]* is log-space complete for *PTIME*. This demonstrates that a straightforward view of the directional duality may be misleading and that the transformation of multiple occurrences of variables into axioms from Theorem 3.1 provides a better view of this duality. In Theorem 4.1 we also show (by a simple modification of the proof in Kozen (1977)) that *dag-CC[3 axioms]* is log-space complete for *PTIME*. The status of *dag-CC[2 axioms]* is an interesting open question. As part of the proof of Theorem 4.1 we show that when C is the trivial equivalence relation, that is each distinct vertex is an equivalence class, then congruence closure is in NC^2. The tricky issue here is the possible existence of cycles in G in the general case. The acyclic G case was already known to be in NC^2 via common subexpression elimination for directed acyclic graphs.

Having investigated the relationship of congruence closure and unification closure we then proceed to examine the acyclicity condition that is often added to these computations. We say that G is acyclic under the equivalence relation C' if the directed graph we get by contracting the vertices in each equivalence class of C' is still acyclic. Acyclic congruence closure returns the congruence closure for instances where G is acyclic under CC and the message "has cycle" otherwise. In Theorem 5.1 we show that acyclic congruence closure is in NC^2 if C has a fixed number of nontrivial equivalence classes, i.e., classes with more than a single vertex. Together with Theorem 3.1 this leads to a new class of instances where computing the mgu of two terms is in NC^2. These instances consist of two terms with a fixed number of distinct variables that occur more than once in the instance. The use of the acyclicity condition is important for this proof and we do not know how to remove it. There is an analog here with the use of acyclicity made in the linear time algorithms for acyclic congruence and unification closure in Downey *et al.* (1980) and Paterson & Wegman (1978).

Our final contribution is in clarifying the relationship of unification with the testing of two deterministing finite automata for equivalence. Let G be a graph with vertices having outdegree 0 or 2 and let us call outdegree 0 vertices leaves and outdegree 2 vertices internal nodes. There is no loss of generality from the point of view of parallel complexity if we thus restrict G. In Theorem 6.1 we show that if G has a fixed number of leaves then computing the unification closure is is NC^2. Note that for the deterministic finite automata application there are no leaves. This result also extends the circuit bounds of Yasuura (1983) on unification of terms with a fixed number of variables, because the graph G can have cycles and is thus more general than the acyclic representation of terms.

In Section 2 we give brief but formal definitions of the problems examined in this paper; Section 3 contains our duality theorem; Section 4 the analysis of *dag-CC* with a fixed number of axioms; Section 5 the analysis of acyclicity; and Section 6 the relationship of unification closure and deterministic finite automata equivalence. In Section 7 we have our conclusions (shown graphically in Figure 5) and open questions.

2 The Problems

A *term* is a finite string that is either a *variable* symbol, or a *constant* symbol, or a string $f(t_1, \ldots, t_a)$, where f is a *function* symbol of *arity* $a \geq 1$ and $t_1, \ldots, t_a$ are terms.

A *ground term* is a term that does not contain any occurrences of variables.

A set of terms is naturally represented as a *simple directed acyclic graph* (*sdag*). Sdags are directed acyclic graphs, where only the leaves (outdegree 0 vertices) can have indegree larger than one (Dwork *et al.* 1984), i.e., the graph looks like a forest except at the leaves.

If a term is a variable or a constant symbol it is denoted by a tree of one vertex labeled by that symbol. If a term is $f(t_1, \ldots, t_a)$ it is denoted by a tree, whose root is a vertex labeled by symbol f and such that the root has as a ordered successors the trees denoting $t_1, \ldots, t_a$. Given a set of terms we represent them by the sdag that results from the trees denoting the terms, if we merge all vertices labeled by the same variable or constant symbol into one such vertex. For example, Figure 1a is the sdag representation of the set of terms $\{f(f(a,y),x)), f(x,(f(y,b)))\}$.

> **UWORD:** The *uniform word problem for finitely presented algebras* is defined as follows. Given ground terms $t_{11}, \ldots, t_{n1}, t_{12}, \ldots, t_{n2}, \ldots, t_1, t_2$ decide whether the implication $S \models \{t_1 = t_2\}$ is true, where $S = \{t_{11} = t_{12}, \ldots, t_{n1} = t_{n2}\}$.

If S is a fixed set of equalities we have the problem *S-WORD* (the word problem for finitely presented algebra S). If S has k equalities we use the notation *UWORD[k axioms]*. If the function symbols in all the input ground terms are monadic then we have the problem *mon-UWORD*. We use *mon-UWORD[k functions]* if the input has only k distinct function symbols.

> **MGU:** The problem of computing the *most general unifier*, (*mgu*) is defined as follows. Given terms t_1, t_2 find the most general substitution of terms for variables in t_1 and t_2 that makes them equal or report that there is no such substitution.

Note that if there is such a substitution there is a most general one (Robinson 1965). For example, the mgu for the terms $f(x,x)$ and $f(g(y), g(g(z)))$ consists of substituting $g(z)$ for y and $g(g(z))$ for x. The terms $f(x,x)$ and $g(x)$ are not unifiable and neither are the terms $g(x)$ and x.

> **MGU$^\infty$:** This is an extension of the mgu of two terms where we allow substitutions of infinite terms for variables in t_1, t_2. For example, we say that the terms $g(x)$ and x are *unrestricted unifiable* by substituting $g(g(g(\ldots)))$ for x, (see Dwork *et al.* 1984; Paterson & Wegman 1978 for the technical definitions).

If the input terms have at most k distinct variables, we have the problems *MGU[k vars]* and *MGU$^\infty$[k vars]*. A variable is *repeated* if it occurs more than once in the input, (i.e., it occurs in both t_1 and t_2, or it occurs twice in t_1 or t_2). If the input terms have at most k repeated variables we have the problems *MGU[k repeated vars]* and *MGU$^\infty$[k repeated vars]*. Clearly if we have *[k vars]* we have *[k repeated vars]* but not inversely. If one of the two terms contains no repeated variables we have the problems *linear-MGU* and *linear-MGU$^\infty$* (Dwork *et al.* 1988). Finally, if we are given a *set* of input pairs $\{t_{11}, t_{12}\}, \ldots, \{t_{k1}, t_{k2}\}$, where all the function symbols have arity 1, and we want the most general substitution that simultaneously makes t_{11} equal to t_{12}, ..., t_{k1} equal to t_{k2}, then we have the problems *mon-MGU* and *mon-MGU$^\infty$*.

EDFA: This is the problem of determining whether two given deterministic finite automata accept the same language.

The above problems represent a wide spectrum of applications which we will now reduce to two combinatorial problems.

CC: Let $G = (V, A)$ be a directed graph such that each vertex $v \in V$ has 0 or 2 ordered successors. Let C be any equivalence relation on V. The *congruence closure* $\approx$ of C is the finest equivalence relation on V that contains C such that for all vertices v and w with corresponding successors v_1, w_1 and v_2, w_2 we have:

$$v_1 \approx w_1, v_2 \approx w_2 \Longrightarrow v \approx w$$

UC: Let $G = (V, A)$ be a directed graph such that each vertex $v \in V$ has 0 or 2 ordered successors. Let C be any equivalence relation on V. The *unification closure* $\sim$ of C is the finest equivalence relation on V that contains C such that for all vertices v and w with corresponding successors v_1, w_1 and v_2, w_2 we have:

$$v_1 \sim w_1, v_2 \sim w_2 \Longleftarrow v \sim w$$

We distinguish among several cases of congruence closure and unification closure depending on the structure of G and C. We use the notation *[k axioms]* when C is the reflexive, symmetric, and transitive closure of *k pairs of distinct vertices.* We use the notation *[k classes]* when C has at most *k nonsingleton equivalence classes.* We use the notation *[k leaves]* when G has at most k leaves. *(s)dag-CC* and *(s)dag-UC* refer to cases where the input graph is a (simple) directed acyclic graph.

Remark on outdegree: In the introduction *CC* and *UC* are presented without any restrictions on the outdegrees of vertices in G. In our formalization we restrict the outdegrees to 0 or 2. This makes the combinatorial problems easier to state and simplifies the notation in our proofs. It is used for the same purposes in Downey *et al.* (1980). More importantly, using the techniques of Downey *et al.* (1980), one can easily show that both for sequential algorithms and for *NC* algorithms the restriction can be made without any loss of generality. For example, vertex labels in the sdag representation of terms can be eliminated without loss of generality, using sdags with vertex outdegrees 0 or 2.

Many applications of unification closure and congruence closure require that the graph formed from the input graph by contracting the equivalence classes of the closures be acyclic. We define *ACC* and *AUC* to have the same inputs as *CC* and *UC*. They return the closure (if the graph formed by the input graph by contracting the closure's equivalence classes is acyclic) or the message "has cycle" (otherwise).

We state four propositions from the literature, which relate the applications *UWORD*, *MGU*, *MGU*$^\infty$, and *EDFA* to the combinatorial problems *CC* and *UC*. Two problems are log-space equivalent if each one is log-space reducible to the other. All the reductions in Propositions 2.1 to 2.4 involve simple and straightforward manipulations of the representations commonly used for terms and for finite automata. Hence, for each one of these cases we will use the combinatorial problems to reason about the application problems.

Proposition 2.1. *UWORD[k axioms] is log-space equivalent to sdag-CC[k axioms].*

Kozen (1977) reduces *UWORD[k axioms]* to the more general version of *sdag-CC[k axioms]*, where the vertices in the input graph may have other than 0 or 2 successors. This is done based on straightforward representation of terms via sdags. Downey *et al.* (1980) reduce this more general case to *sdag-CC[k axioms]* where the vertices in the input graph have 0 or 2 successors.

Proposition 2.2. *MGU is log-space equivalent to sdag-AUC[1 axiom].*

Proposition 2.3. *MGU^{∞} is log-space equivalent to sdag-UC[1 axiom].*

The reductions follow from Paterson & Wegman (1978).

Proposition 2.4. *EDFA is log-space equivalent to UC[0 leaves].*

This reduction follows from Hopcroft & Karp (1971).

We close this section with algorithms for solving CC and UC. Let G and C be as in the definitions of CC and UC above and let u and v be vertices of G. We define symmetric and reflexive relations E and F on pairs of vertices of G. These relations are represented by undirected edges added to G and labeled E or F. For each two vertices u and v that are in the same equivalence class of C we *add* undirected edges uEv and uFv to the graph. Also,

Add undirected edge uEv if it is not present and either:

1. *u_1Ev_1 and u_2Ev_2 are present, where u_1, u_2 and v_1, v_2 are the ordered successors of u, v. In this case u and v are distinct vertices. This is called up-propagation step uPv.*

2. *uEw and wEv are present, where w is some vertex in G. In this case u and v are distinct vertices. This is called transitivity step uTv.*

Add undirected edge uFv if it is not present and either:

1. *$u'Fv'$ is present, where u and v are corresponding successors of u', v'. In this case u and v are distinct vertices. This is called down-propagation step $uP'v$.*

2. *uFw and wFv are present, where w is some vertex in G. In this case u and v are distinct vertices. This is called transitivity step uTv.*

From Kozen (1977) and Paterson & Wegman (1978) we have the following characterization of the congruence closure relation ($\approx$) and the unification closure relation ($\sim$).

Proposition 2.5. *$u \approx v$ ($u \sim v$) iff undirected edge uEv (uFv) is added after some finite sequence of up(down)-propagation and transitivity steps.*

3 Unification Closure Reduces to Congruence Closure

Let I be an instance of *UC* and u, v be two vertices in I. In this section we will transform the question whether the pair (u, v) is in the unification closure of I (i.e., $u \sim v$) into a uniform word problem for monadic finitely presented algebras. This together with Proposition 2.1 reduces unification closure to congruence closure.

Given I, u, v as above we now produce a set of equations $S(I)$, an equation $s(I)$, and an instance of *CC* we call *dual(I)* as follows:

1. For each vertex x_1 in I with indegree $i > 1$ do the following modification. If $(z_1, x_1), \ldots, (z_i, x_1)$ are the arcs coming into x_1, with arc labels 1 or 2, then replace $(z_2, x_1), \ldots, (z_i, x_1)$ by $(z_2, x_2), \ldots, (z_i, x_i)$, with the same arc labels, where $x_2, \ldots, x_i$ are *new* vertices. Add *new* axioms $x_1 \sim x_2, \ldots, x_1 \sim x_i$.
2. The graph resulting from step 1 is a forest, thus there is at most one arc coming into each vertex. Add vertex labels using the following procedure. If vertex x has incoming arc (z, x) with arc label 1 (2) then label x with monadic function symbol h (g). If vertex x has no incoming arc then label x with a unique constant symbol.
3. In the graph resulting from step 2 reverse the directions of the arcs and change all arc labels to 1. The resulting graph is the sdag representation of a set of monadic ground terms. These monadic ground terms are constructed from constants and the symbols h and g. In this sdag every vertex x denotes a monadic ground term t_x.
4. $S(I)$ is $\{t_x = t_y \mid \text{where } x \sim y \text{ is an axiom of } I \text{ or a new axiom from step 1}\}$; $s(I)$ is $t_u = t_v$.
5. The instance of *CC dual(I)* consists of a graph G' and an equivalence relation C'. The graph G' is the directed graph with arc labels resulting from step 3 with the following modification. Add two new vertices h and g and make new vertex h (g) the second successor of each vertex labeled by symbol h (g). The equivalence relation C' is the one defined by axioms of I and the new axioms from step 1.

Figure 1 illustrates this method of reduction. The *sdag-UC[1 axiom]* instance in Figure 1a (ignore vertex labels) is transformed into Figure 1b. This in turn is the sdag representation for the *mon-UWORD* instance $\{g(h(c)) = h(g(d)), g(c) = h(d), c = d\} \models \{h(h(c)) = g(g(d))\}$. Compare this sdag with the sdag in Figure 1a which can be used for computing the unification closure of terms $f(f(a, y), x))$ and $f(x, f(y, b))$. The implication in the word problem holds. In the unification closure of the two terms the vertices labeled a and b are in the same class; this leads to a failure of the homogeneity test of Paterson & Wegman (1978) for mgu's.

Theorem 3.1. *Let I be an instance of UC and u, v be two vertices in I. Let $S(I)$ be a set of equations, $s(I)$ an equation, and dual(I) an instance of CC defined as above. Then the following are equivalent:*
(a) $u \sim v$ is in the unification closure of I.
(b) $S(I) \models s(I)$.
(c) $u \approx v$ is in the congruence closure of dual(I).

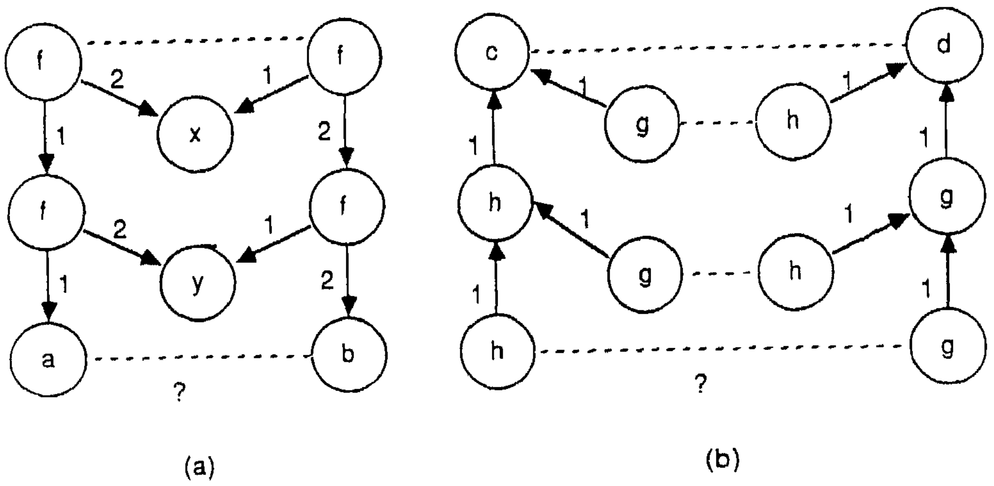

Figure 1: Example of reduction from sdag-UC to mon-UWORD.

Proof: The equivalence of (b) and (c) follows from Kozen's algorithm for the uniform word problem for finitely presented algebras in Kozen (1977).

Consider the instance I' of UC that we get right after step 1 of the reduction above, i.e., after the addition of the x vertices and the new axioms. It is easy to see that $u \sim v$ is in the unification closure of I iff $u \sim v$ is in the unification closure of I'. The arc label changes in steps 2, 3 and 5 are such that every down-propagation step on I' corresponds to an up-propagation step in *dual(I)* and vice versa. The same is true for the transitivity steps on I' and *dual(I)*. Thus by Proposition 2.5 (a) and (b) are equivalent. □

As we described in Proposition 2.2 computing MGU^{∞} is intimately related to computing the unification closure of *sdag-UC[1 axiom]* instances. The additional homogeneity test can be performed in NC. Based on the above reduction we have:

Corollary 3.2. *Let I be an instance of sdag-UC[1 axiom] and u, v be two vertices in I. Let $k = \sum_x [indegree(x)-1]$, where x is a leaf in I of indegree ≥ 1. The $S(I), s(I)$ defined above are an instance of mon-UWORD[1+k axioms], such that, $u \sim v$ iff $S(I) \models s(I)$.*

Consider the MGU^{∞} instance t_1, t_2. If represented in sdag form then there are leaves denoting both variable and constant symbols. One can replace each occurrence of a constant with a unique new variable and use the unification closure on the sdag of the new terms t'_1, t'_2. This closure can be used to find the mgu^{∞} of t_1, t_2, because constants can only be unified with themselves. Therefore for the MGU^{∞} application the k used in Corollary 3.2 is $k =$ (number of occurrences of variables in t_1, t_2) - (number of distinct variables in t_1, t_2).

Using the reduction of Theorem 3.1 and reductions from (Dwork *et al.* 1984; Dwork *et al.* 1988) one can show that:

Corollary 3.3. *mon-UWORD[2 functions] is log-space complete in PTIME.*

One constant and two monadic function symbols suffice for the proof of Corollary 3.3. However, for one monadic function symbol we have the following.

Proposition 3.4. *mon-UWORD[1 function] is in NC^2.*

Proof: Consider a directed graph $G = (V, A)$ such that each vertex $v \in V$ has 0 or 1 successor. Let C be an equivalence relation on V. The closure of C is the finest equivalence relation C^* such that for all vertices v, w with successors v', w' we have: $(v, w) \in C^* \Longrightarrow (v', w') \in C^*$.

Computing C^* is the monadic outdegree version of *UC*. By reversing the direction of the arcs it is easy to see that computing C^* in NC^2 would suffice to prove this theorem.

Each *component* of graph G is either a tree directed towards the root, or a single directed cycle onto whose vertices such trees are rooted. The NC^2 algorithm consists of two parts. In the first part components are *merged*. In the second part the computation is performed on *separate* components.

> *Merge:* If an axiom (v, w) of C has vertices in two components we merge these components into one component. This merge operation is performed by merging descendants of v and w that have the same distance from v and w. One merge operation can be performed in $O(logN)$ parallel time. Merges can be performed so that after $O(logN)$ phases there are no axioms between components.
>
> *Separate:* We have reduced the computation of C^* to subcomputations where G is one component. Each one of these subcomponents is a special case of *UC*, namely, *UC[0 leaves]* or *UC[1 leaf]*. These can be performed in NC^2. We will give a more general proof for *UC[k leaves]* k fixed in Theorem 6.1. □

Using the proof of this theorem we have, (as in Auger 1985 for *mon-MGU*):

Corollary 3.5. *mon-MGU^∞ is in NC^2.*

Let us close this section by noting that the uniformity of the word problem is important for the log-space completeness. The proposition below can also be shown to follow from Ruzzo (1980), so we only provide a sketch of the proof.

Proposition 3.6. *S-WORD is in NC^2 for any fixed S.*

Proof Sketch: For a fixed S we can produce in constant time a tree automaton presentation of the algebra of Thatcher & Wright (1968). To check for an equality $t_1 = t_2$ in the algebra all we have to do is run this automaton on t_1 and t_2. This can be performed in NC^2. □

4 Congruence Closure With a Fixed Number of Axioms

In this section we show that there is a distinction between *UC* and *CC*. Namely, we show that *dag-CC[1 axiom]* is in NC^2 whereas it is known that *sdag-UC[1 axiom]* is log-space complete for *PTIME*.

Theorem 4.1. *CC[0 axioms] and dag-CC[1 axiom] are in NC^2. CC[k axioms] and dag-CC[k+1 axioms] are log-space complete in* PTIME *for each fixed $k \geq 2$.*

We first prove two lemmas. In this section by propagation steps we mean up-propagation steps, and by a (congruence) proof we mean any valid sequence of up-propagation and transitivity steps.

Lemma 4.2. *CC[0 axioms] is in* NC^2.

Proof: The lemma follows from two observations: (1) C has no axioms then all proofs containing some transitivity steps can be replaced by proofs containing only propagation steps and (2) sequences of propagation steps can be done in NC^2.

To show (1) we apply repeatedly the claim below for any proof, always replacing the rightmost transitivity step until none remains.

Claim: When C has no axioms, any transitivity step preceded by propagation steps only can be replaced by a sequence of propagations.

Let vTw be a counterexample to the claim, such that, it is preceded by the fewest number of propagation steps. Without loss of generality the sequence must look as follows:

$$P_1, vPu, P_2, uPw, P_3, vTw$$

where P_1, P_2 and P_3 are (possibly empty) sequences of propagation steps.

Since C has no axioms, $v \approx u$ and $u \approx w$ were proven by propagation and all of v, u, w have exactly two children, say v_1, v_2, u_1, u_2, and w_1, w_2, respectively. Note that either v_1 is u_1 or $v_1 P u_1$ must precede vPu, and either u_1 is w_1 or $u_1 P w_1$ must precede uPw. Therefore, either v_1 is w_1 or we can replace the end of the original sequence to get

$$P_1, vPu, P_2, v_1 T w_1$$

Since $v_1 T w_1$ is preceded by fewer propagation steps than vTw, it cannot be a counterexample. Hence $v_1 T w_1$ can be replaced by a sequence of propagations, showing that $v_1 \approx w_1$ can be proven by propagations only. Reasoning similarly, $v_2 \approx w_2$ can also be proven by propagations only. Therefore $v \approx w$ also has a propagation proof, which is a contradiction to the counterexample vTw. This completes (1).

To show (2) we reduce *CC[0 axioms]* with $G = (V, A)$ to transitive closure of the directed graph $G' = (V', A')$, where $V' = \{(v, w) : v, w \in V\} \cup \{x\}$ where x is a new node, nd A' is as follows:

For all $v \in V$ let (v, v) have successor x in G'. For all $v, w \in V$ with successors v_1, v_2 and w_1, w_2 in G, respectively, let (v, w) have successors (v_1, w_1) and (v_2, w_2) in G'.

Then for all $v, w \in V$, $v \approx w$ if and only if the following conditions both hold:

Each descendant of (v, w) has x as a descendant (except for x itself).
The descendants of (v, w) form an acyclic graph.

The correctness of this reduction can be easily proven by induction on the length of propagation sequences. This completes (2) and the lemma. □

The single pair congruence closure problem seems harder than congruence closure with no axioms. Given a directed acyclic graph like that in Figure 2, we need transitivity steps, to show for example that $x \approx z_1$. Moreover, to show that $x \approx z_i$, we need i alternations

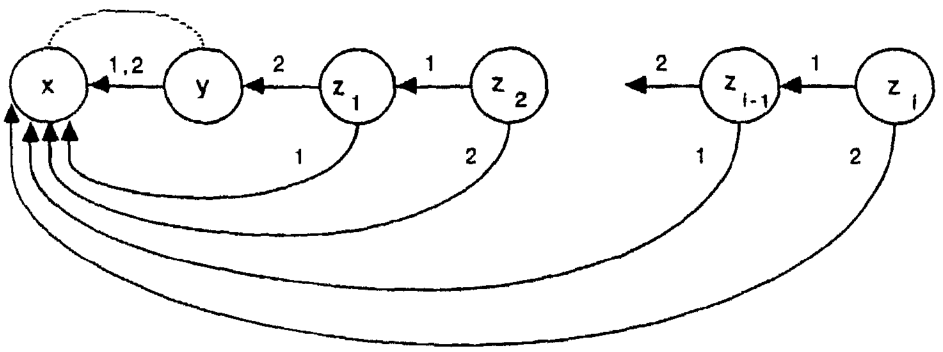

Figure 2: Example of dag-CC[1 axiom]

in propagation and transitivity steps. However, since x does not have any children, in this case we could *merge* it with y and then perform propagation and transitivity steps. In this way the problem reduces to the no axioms case, and only propagation steps are needed. The next theorem shows that this can be done in general.

Lemma 4.3. *dag-CC[k+1 axioms] log-space reduces to CC[k axioms],* $k \geq 0$.

Proof: Let us first give the proof for $k = 0$.

Let $G = (V, A)$ be any dag with $x \approx y$ an axiom in C, where x and y are two *distinct* vertices in V, and let T be an arbitrary topological ordering of the vertices. Without loss of generality $T(y) > T(x)$. Let us assume that there are no common subexpressions, i.e., we performed congruence closure with 0 axioms. This is because this computation can be done in NC^2 by Lemma 4.2. Now we prove the following claim.

Claim: When G is acyclic and $T(y) > T(x)$, if $u \approx v$ holds, then $T(u), T(v) \geq T(x)$. The claim is shown by induction on the length of the proof for $u \approx v$.

Base: Suppose $u \approx v$ has a proof of length 1. Then it must be a propagation proof. Let u_1 and u_2 be the successors of u, and v_1 and v_2 be the successors of v. Since $u \not\approx v$ when C has no axioms (because of common subexpression elimination), the proof must depend on $x \approx y$. Then without loss of generality $(u_1, v_1) = (x, y)$. Therefore, $T(u), T(v) > T(x)$, and the claim holds for proofs of length 1.

Induction hypothesis: "If $u \approx v$ has a proof of length $i \geq 1$, then $T(u), T(v) \geq T(x)$." Then suppose $u \approx v$ has a proof of length $i + 1$. There are two cases:

1. The last step was transitivity of the form: $uEw, wEv \Longrightarrow uTv$. Then $u \approx w$ and $w \approx v$ have proofs shorter than $i + 1$, hence by the induction hypothesis, $T(u), T(v), T(w) \geq T(x)$. Hence $T(u), T(v) \geq T(x)$.

2. The last step was propagation. Then $u_1 \approx v_1$, and $u_2 \approx v_2$ have proofs shorter than $i+1$, hence by the induction hypothesis, $T(u_1), T(u_2), T(v_1), T(v_2) \geq T(x)$. Since the graph is acyclic, $T(u), T(v) \geq T(x)$ also holds.

Since nodes that are greater than x cannot use the descendants of x, by the above claim, we can make the successors of x be the same as the successors of y. This modification of G will not change the computation of the congruence closure but will allow us to merge vertex x and y and still get a directed graph with outdegrees 0 or 2. Then the problem reduces to congruence closure with no axioms, which by Lemma 4.2 is in NC^2. Note that this reduces *dag-CC* to *CC* and not to *dag-CC*.

Finally if $k \geq 0$ the same technique can be used to eliminate one equality by starting from the vertex with the lowest number in the topological order. This completes the lemma. □

Proof of Theorem 4.1: The theorem follows for $k = 0$ by Lemmas 4.2 and 4.3 and for $k \geq 2$ by a reduction from the circuit value problem (CVP) which was proven logspace complete for *PTIME* by Ladner (1975). The circuit value problem is a sequence $g_1, g_2, \ldots, g_n$, where each g_i is either (i) a Boolean variable, which is assigned true or false, or (ii) $NOR(j,k)$, with $j, k < i$. The circuit value problem operation is: for a given circuit and an assignment to the variables find the output of the circuit.

To do the reduction, we introduce two special vertices 1 and 0. Every boolean variable g_i that is assigned true is assigned to 1, and every boolean variable g_i that is assigned false is assigned to 0. In addition, for each g_i that is not a variable we create a vertex with first successor g_j and second successor g_k. We can encode into the congruence closure problem the function of a NOR gate by adding three congruences in Figure 3. Out of these the congruence $0 \approx z$ can be eliminated by merging the vertices 0 and z (see Figure 4).

Now it is easy to prove by induction that the CVP is true if and only if the node g_n will be congruent to 1. Hence the CVP problem can be reduced to dag congruence closure with 3 axioms and to congruence closure with 2 axioms. The cases for $k > 2$ are also immediately implied. □

Note that the complexity of *CC[1 axiom]* and *dag-CC[2 axioms]* is open. *dag-CC[2 axioms]* reduces to *CC[1 axiom]* by Lemma 4.2. Finally, the large number of paths in a dag was important in the proof of Theorem 4.1. The complexity of *sdag-CC[k axioms]* for fixed $k \geq 2$ is open.

5 Acyclic Congruence Closure

In this section we show that there is a further distinction between *UC* and *CC*.

Theorem 5.1. *ACC[k classes] is in NC^2 for each fixed $k \geq 0$.*

Proof: Suppose that we have an input graph with k classes. If the input graph is cyclic, then return a "has cycle" message, else eliminate common subexpressions. Then take the graph G formed from the input graph by contracting the equivalence classes in C. If G is cyclic, then return a "has cycle" message, else pick an arbitrary topological ordering of the vertices in G. Find the vertex in some nontrivial equivalence class, such that this vertex has the least topological number. Similarly to Lemma 4.3 we can show

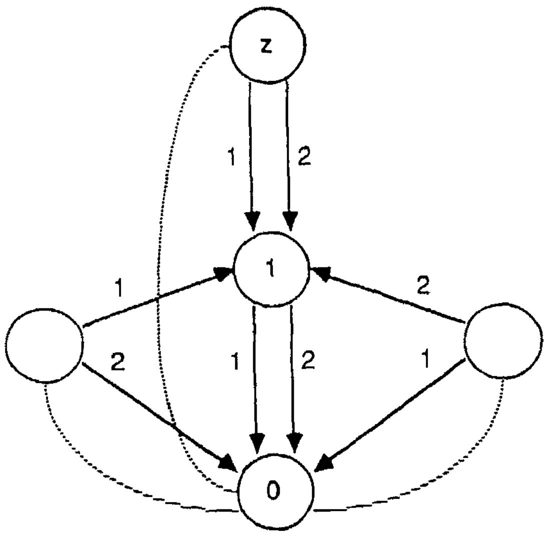

Figure 3: Example of dag-CC[3 axioms].

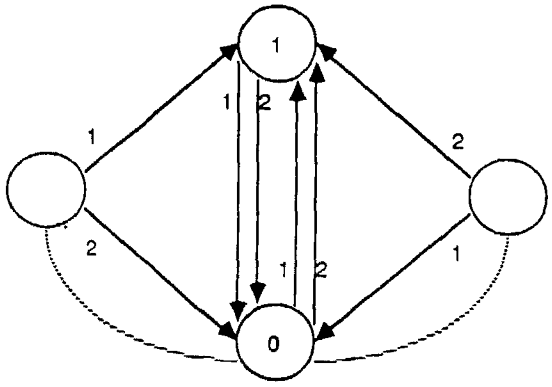

Figure 4: Example of CC[2 axioms].

that the descendants of this merged vertex are not needed. By acyclicity this is true for the other vertices in its class. Hence we can take the input graph and merge only these vertices, whose descendants are not needed and discard their descendants. Since the merged vertices correspond to one nontrivial equivalence class in C, this yields a new graph with $k-1$ classes. The new graph is also acyclic. Since this reduction is in NC^2, we can repeat it k times, and the theorem must hold. □

Based on this theorem and on Proposition 2.2 we can show the following.

Corollary 5.2. *MGU[k repeated vars] is in NC^2 for each fixed $k \geq 0$.*

As after Corollary 3.2 there is only one fine point. Consider the *MGU* instance t_1, t_2. If represented in sdag form then there are leaves denoting both variable and constant symbols. One can replace each occurrence of a constant with a unique new variable and use the unification closure on the sdag of the new terms t'_1, t'_2. This closure can be used to find the *mgu* of t_1, t_2, because constants can only be unified with themselves.

In Corollary 5.2 we have a new class of term unification problems that is shown to be in NC^2. The previously known cases were *linear-MGU* (Dwork *et al.* 1988), *mon-MGU* (Auger 1985), and *MGU[k vars]* (Yasuura 1983) for each fixed $k \geq 0$.

6 On Deterministic Finite Automata Equivalence

Theorem 6.1. *UC[k leaves] is in NC^2 for each fixed $k \geq 0$.*

Proof: Let us implement the procedure of Proposition 2.5 as the following algorithm *NU* (for naive unification).

1. On the graph G add the axioms of C as undirected edges uFv, as well as, all self-loops uFu.
2. Perform as many down-propagation steps as possible.
3. Perform as many transitivity steps as possible.
4. If a leaf is connected via an undirected edge to another then merge it with that vertex.
5. If any new propagation is possible then go to step 2 else terminate.

By Proposition 2.5 this algorithm will produce the closure, provided we keep track of which vertices the leaves are merged with. Steps 2 and 3 can be performed in NC. The problem with this algorithm as a general parallel algorithm is that Steps 2 and 3 might have to alternate $O(N)$ times (Dwork *et al.* 1984).

Let us first examine the $k=0$ case; (by Proposition 2.4 this is log-space equivalent to *EDFA*). In this case we can argue that executing Steps 2 and 3 once suffices. Suppose it does not. Then some new propagation step is enabled, i.e, at Step 3 we have shown some x_1Fx_i and y_1, y_i are corresponding successors of x_1, x_i for which we have not discovered y_1Fy_i. Now in order to show x_1Fx_i we have found a (perhaps empty) sequence of vertices $x_2, \ldots, x_{i-1}$ such that $x_1Fx_2, \ldots, x_{i-1}Fx_i$. Because there are no leaves and all outdegrees are 2 there exist $y_2, \ldots, y_{i-1}$, which are are the corresponding successors of $x_2, \ldots x_{i-1}$. In Steps 2 and 3 we would have already discovered $y_1Fy_2, \ldots, y_{i-1}Fy_i$ and therefore y_1Fy_i. This is the desired contradiction.

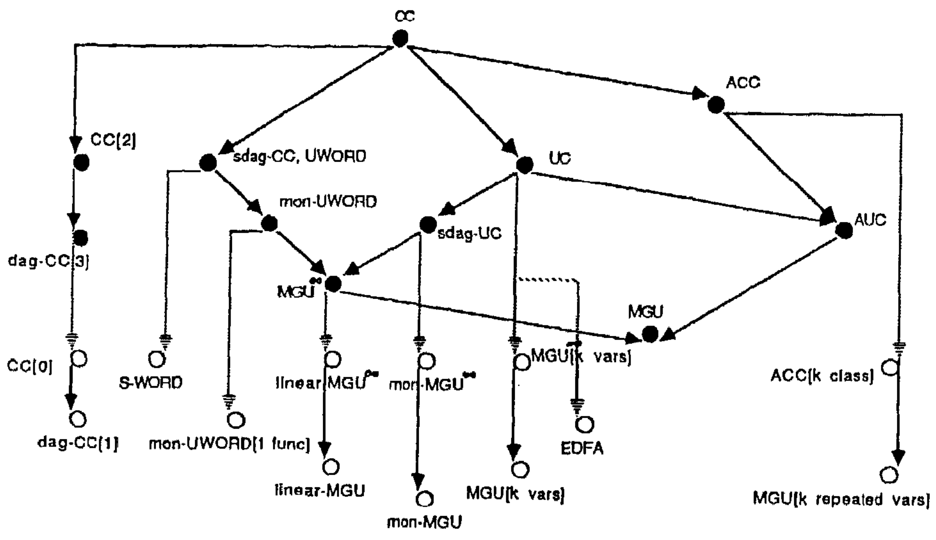

Figure 5: The reduction map, ○ in NC, ● is P-complete.

For the $k \geq 0$ case all we have to note is that, in Step 4 at every iteration one leaf is eliminated at least. Since k is fixed we reduce to the $k = 0$ case after a fixed number of NC^2 computations. This completes the proof of the theorem. □

An immediate Corollary of this theorem is that *MGU^∞[k vars]* for fixed k is in NC^2. Since the mgu^∞ has only k variables, by counting the number of possible substitutions (i.e., $O(N^k)$) we could reach a similar conclusion. However, the proof of Theorem 6.1 gives a more structured way of building NC^2 circuits for mgu$^\infty$. A more involved construction for *MGU[k vars]* is contained in Yasuura (1983).

Corollary 6.2. *MGU^∞[k vars] is in NC^2 for each fixed $k \geq 0$.*

7 Open Problems

In Figure 5 we summarize the known results about subcases of congruence closure and unification. The edges (P, Q) between problems can be read as "Q reduces to P".

There are a few problems whose complexity is open. These are *CC[1 axiom]*, *dag-CC[2 axioms]*, *sdag-CC[k axioms]* and *MGU^∞[k repeated vars]*, where $k \geq 2$ and is fixed. We conjecture that these problems are in NC.

Acknowledgement: The authors would like to thank Foto Afrati for her helpful comments on a previous draft of this paper.

References

Auger I.E., Krishnamoorthy M.S. (1985). A Parallel Algorithm for the Monadic Unification Problem, *BIT* **25**, 302-306.

Downey, P.J., Sethi, R. (1978). Assignment Commands with Array References, *J. ACM* **25**, (4), 652-666.

Downey, P.J., Sethi, R., Tarjan, R.E. (1980). Variations on the Common Subexpression Problem, *J. ACM* **27**, (6), 758-771.

Dwork, C., Kanellakis, P., Mitchell, J. (1984). On the Sequential Nature of Unification, *Journal of Logic Programming* **1**, (1), 35-50.

Dwork, C., Kanellakis, P., Stockmeyer, L. (1988). Parallel Algorithms for Term Matching, IBM Research Report, RJ 5328, (to appear in the *SIAM Journal of Computing*).

Fortune, S., Wyllie, J. (1978). Parallelism in Random Access Machines, *Proc.* 10th *ACM STOC*, pp. 114-118.

Hopcroft, J.E., Karp, R.M. (1971). An Algorithm for Testing the Equivalence of Finite Automata, Tech. Rep. 71-114, Computer Science Dept., Cornell Univ., Ithaca, N.Y.

Huet, G. (1976). Résolution d'equations dans les langages d'ordre 1,2, $\ldots, \omega$. Thèse d'état de l'Université de Paris 7.

Huet, G., Oppen, D. (1980). Equations and Rewrite Rules: a Survey, In *Formal Languages: Perspectives and Open Problems*, Book, R., Ed., Academic Press, 349-403.

Kozen, D. (1977). Complexity of Finitely Presented Algebras, *Proc.* 9th *ACM STOC,* pp. 164-177.

Ladner, R. (1975). The Circuit Value Problem is Log Space Complete for P, *SIGACT News* **7**, (1), 18-20.

Lloyd, J.W. (1984). *Foundations of Logic Programming.*, Springer-Verlag.

Martelli, A., and Montanari, U. (1982). An Efficient Unification Algorithm, *ACM Trans. on Prog. Lang. and Syst.* **4**,, (2), 258-282.

Nelson G., and Oppen, D. (1980). Fast Decision Procedures based on Congruence Closure, *J. ACM* **27**, (2), 356-364.

Oppen, D. (1980). Reasoning about Recursively Defined Data Structures, *J. ACM* **27**, (3), 403-411.

Paterson, M.S., Wegman, M.N. (1978). Linear Unification, *JCSS* **16**, 158-167.

Pippenger, N. (1979). On Simultaneous Resource Bounds, in *Proc.* 20th *IEEE FOCS*, pp. 307-311.

Ramesh, R., Verma, R.M., Krishnaprasad, T., Ramakrishnan, I.V. (1987). Term Matching on Parallel Computers, *ICALP '87*, in Springer-Verlag Lec. Notes Comp. Sci. **267**, 336-346.

Robinson, J.A. (1965). A Machine Oriented Logic Based on the Resolution Principle, *J. ACM* **12**, (1), 23-41.

Robinson, J.A. (1975). private communication in Paterson & Wegman (1978).

Ruzzo, W.L (1980). Tree-size Bounded Alternation, *JCSS* **21**, (2), 218-235.

Tarjan, R.E. (1975). Efficiency of a Good but not Linear Set Union Algorithm, *J. ACM* **22**, 215-225.

Thatcher, J.W., Wright, J.B. (1968). Generalized Finite Automata Theory with an Application to a Decision Problem of Second Order Logic. *Math. Syst. Th.* **2**.

Vitter, J.S. and Simons, R. (1986). New Classes for Parallel Complexity: a Study of Unification and Other Complete Problems for P. *IEEE Transactions on Computers* **C-35**, (5), 406-418.

Yasuura, H. (1983). On the Parallel Computational Complexity of Unification. ER 83-01, Yajima Lab.