Fast Decision Procedures Based on Congruence Closure

GREG NELSON AND DEREK C. OPPEN

Stanford University, Stanford, California

ABSTRACT. The notion of the *congruence closure* of a relation on a graph is defined and several algorithms for computing it are surveyed A simple proof is given that the congruence closure algorithm provides a decision procedure for the quantifier-free theory of equality A decision procedure is then given for the quantifier-free theory of LISP list structure based on the congruence closure algorithm Both decision procedures determine the satisfiability of a conjunction of literals of length n in average time $O(n \log n)$ using the fastest known congruence closure algorithm It is also shown that if the axiomatization of the theory of list structure is changed slightly, the problem of determining the satisfiability of a conjunction of literals becomes NP-complete The decision procedures have been implemented in the authors' simplifier for the Stanford Pascal Verifier

KEY WORDS AND PHRASES: program verification, mechanical theorem proving, decision procedure, congruence closure, graph algorithms, theory of equality, theory of recursive data types

CR CATEGORIES 5 21, 5.24, 5 25, 5.7

1. *Introduction*

Consider the problem of verifying that one equality is a consequence of several other equalities, for example, that $f(f(a, b), b) = a$ is a consequence of $f(a, b) = a$, or, less obviously, that $f(a) = a$ is a consequence of $f(f(f(a))) = a$ and $f(f(f(f(f(a))))) = a$. A practical algorithm for this problem is essential to mechanical program verification (or to any other kind of mechanical reasoning), since almost all proofs require reasoning about equalities.

In 1954 Ackermann [1] showed that the problem was decidable but did not give a practical algorithm. The problem appears to have been ignored for the next twenty-four years. In 1976 and 1977 several people attacked the problem from quite different points of view. Downey, Sethi, and Tarjan [3] viewed the problem as a variation of the common subexpression problem, Kozen [4] as the word problem in finitely presented algebras, and Shostak [8] and Nelson and Oppen [5] as the decision problem for the quantifier-free theory of equality with uninterpreted function symbols.

All these problems reduce to the problem of constructing the "congruence closure" of a relation on a graph. In Section 2 we define this notion and describe a congruence closure algorithm which we implemented in 1976 for use in the theorem prover of the Stanford Pascal Verifier. Its worst-case time is $O(m^2)$ for graphs with m edges. Downey, Sethi, and Tarjan [3] describe an algorithm with worst-case time $O(m \log^2 m)$, which, by using a hash table, can be made to run in average-case time $O(m \log m)$. We implemented this algorithm but did not find it faster than the simpler algorithm in our application.

An earlier version of this paper appeared in the Proceedings of the 18th Annual Symposium on Foundations of Computer Science, Providence, R I, October 1977

This research was supported by the National Science Foundation under contract MCS 76-000327 and by the Fannie and John Hertz Foundation

Authors' address Artificial Intelligence Laboratory, Computer Science Department, Stanford University, Stanford, CA 94305

In Section 3 we prove that the congruence closure algorithm provides a decision procedure for the quantifier-free theory of equality with uninterpreted function symbols. Other proofs have been given by Shostak [8] and Kozen [4], but ours is simpler.

In Section 4 we give a decision procedure based on congruence closure for another theory of interest in program manipulation: the quantifier-free theory of LISP list structure with uninterpreted function symbols. The axioms of this theory are

$$\begin{aligned}
&\mathrm{CAR}(\mathrm{CONS}(x, y)) = x,\\
&\mathrm{CDR}(\mathrm{CONS}(x, y)) = y,\\
&\neg\mathrm{ATOM}(x) \supset \mathrm{CONS}(\mathrm{CAR}(x), \mathrm{CDR}(x)) = x,\\
&\neg\mathrm{ATOM}(\mathrm{CONS}(x, y)).
\end{aligned}$$

(Terms may contain uninterpreted function symbols, as well as the function symbols CAR, CDR, and CONS.) Using the fastest version of the congruence closure algorithm, the decision procedure requires average time $O(n \log n)$ to determine the satisfiability of a conjunction of length n.

We conclude Section 4 with a curiosity: If the axiomatization of the theory of list structure is altered by specifying the result of CAR and CDR on atoms, the problem of determining the satisfiability of a conjunction becomes NP-complete.

The two decision procedures given in Sections 3 and 4 have been implemented in the simplifier for the Stanford Pascal Verifier. Details of how these and other decision procedures are combined to form a simplifier are given in [6].

2. *Computing the Congruence Closure*

Let $G = (V, E)$ be a directed graph with labeled vertices, possibly with multiple edges. For a vertex v, let $\lambda(v)$ denote its label and $\delta(v)$ its outdegree, that is, the number of edges leaving v. The edges leaving a vertex are ordered. For $1 \le i \le \delta(v)$, let $v[i]$ denote the ith *successor* of v, that is, the vertex to which the ith edge of v points. A vertex u is a *predecessor* of v if $v = u[i]$ for some i. Since multiple edges are allowed, possibly $v[i] = v[j]$ for $i \neq j$. Let n be the number of vertices of G and m the number of edges of G. We assume there are no isolated vertices and therefore that $n = O(m)$.

Let R be a relation on V. Two vertices u and v are *congruent under R* if $\lambda(u) = \lambda(v)$, $\delta(u) = \delta(v)$, and, for all i such that $1 \le i \le \delta(u)$, $(u[i], v[i]) \in R$. R is *closed under congruences* if, for all vertices u and v such that u and v are congruent under R, $(u, v) \in R$. There is a unique minimal extension R' of R such that R' is an equivalence relation and R' is closed under congruences; R' is the *congruence closure* of R.

For example, let G be the graph shown in Figure 1 and R the relation $\{(V2, V3)\}$. The vertices $V1$ and $V2$ are congruent under R, so the congruence closure of R must include the pairs $(V2, V3)$ and $(V1, V2)$. In fact, the minimal equivalence relation containing these pairs, namely the equivalence relation with associated partition $\{\{V1, V2, V3\}, \{V4\}\}$, is closed under congruences and is therefore the congruence closure of R. Notice that the vertices $V1$, $V2$, $V3$, and $V4$ of G represent in a natural way the terms $f(f(a, b), b)$, $f(a, b)$, a, and b, respectively. Deducing that $V1$ must be equivalent to $V3$ in the congruence closure is analogous to deducing $f(f(a, b), b) = a$ from $f(a, b) = a$ by the substitutivity of equality.

As a second example, let G be the graph shown in Figure 2, R the relation $\{(V1, V6), (V3, V6)\}$, and R' the congruence closure of R. Vertices $V2$ and $V5$ are congruent under R, so $(V2, V5) \in R'$. Since R' is closed under congruences, $(V1, V4) \in R'$. The pairs $(V1, V4)$ and $(V1, V6)$ are both in R', so $(V4, V6) \in R'$. Hence $(V3, V5) \in R'$. Thus all six vertices are equivalent in the congruence closure. Essentially, we have proved the fact that $f(f(f(a))) = a$ and $f(f(f(f(f(a))))) = a$ together imply $f(a) = a$.

We now consider the problem of computing the congruence closure. We represent an equivalence relation by its corresponding partition, that is, by the set of its equivalence classes. We use two procedures for operating on the partition: UNION and FIND.

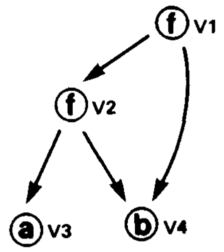

FIGURE 1

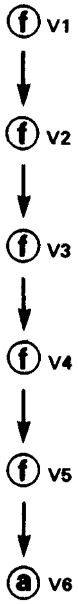

FIGURE 2

UNION(u, v) combines the equivalence classes of vertices u and v. FIND(u) returns the unique name of the equivalence class of vertex u.

Suppose that R is a relation on the vertices of a graph G, that R is closed under congruences, and that u and v are vertices of G. The following procedure MERGE(u, v) constructs the congruence closure of the relation $R \cup \{(u, v)\}$.

```
MERGE(u, v)
1 If FIND(u) = FIND(v), then return
2 Let $P_u$ be the set of all predecessors of all vertices equivalent to u, and $P_v$ the set of all predecessors of all
  vertices equivalent to v.
3 Call UNION(u, v)
4 For each pair (x, y) such that $x \in P_u$ and $y \in P_v$, if FIND(x) $\neq$ FIND(y) but CONGRUENT(x, y) = TRUE,
  then MERGE(x, y).
CONGRUENT(u, v).
1 If $\delta(u) \neq \delta(v)$, then return FALSE
2 For $1 \leq i \leq \delta(u)$, if FIND(u[i]) $\neq$ FIND(v[i]), then return FALSE
3. Return TRUE
```

Since the algorithm calls UNION only on the initial pair of vertices and on congruent pairs, the final equivalence relation is not too coarse. Suppose that it is too fine. Then there are two vertices x and y which are congruent but not equivalent. Since R was closed under congruences initially, there was some call UNION(a, b) such that x and y were not congruent before the call but were congruent after it. Obviously some successor of x was equivalent to either a or b before this call to UNION, and the same holds for y. Furthermore, this call UNION(a, b) was made from step 3 of some call MERGE(a, b), so step 4 of that call to MERGE must merge x and y, contrary to assumption. Thus the algorithm is correct.

CLAIM 1. *The number of calls to* CONGRUENT *is bounded by* $O(mn)$, *for any sequence of calls to* MERGE.

PROOF. Two vertices u and v are checked for congruence only when two of their successors are merged; this can happen at most $\delta(u) + \delta(v) - 1$ times. Thus the number of calls to CONGRUENT is bounded by

$$\begin{aligned}\sum_{u,v} (\delta(u) + \delta(v) - 1) &= \sum_{u,v} \delta(u) + \sum_{u,v} \delta(v) - \sum_{u,v} 1 \\ &= 2mn - n(n-1)/2 \\ &= O(mn). \qquad \square\end{aligned}$$

CLAIM 2. *The number of calls to* FIND *from* CONGRUENT *is bounded by* $O(m^2)$, *for any sequence of calls to* MERGE.

PROOF. Let n_k be the number of vertices with outdegree k. Each pair of vertices with outdegree k can be checked for congruence at most $2k - 1$ times; each check requires at most $2k$ calls to FIND. Thus the total number of calls to FIND is bounded by

$$\sum_k 4k^2 n_k^2 \leq \left(\sum_k 2kn_k\right)^2 = 4m^2. \qquad \square$$

We associate with each equivalence class a "predecessor list" of all vertices with successors in the class. No vertex appears more than once in the predecessor list. When UNION combines two equivalence classes, it merges their predecessor lists into one, eliminating any duplicates, and associates the new predecessor list with the new equivalence class. With each vertex is associated a unique number from 1 to n; the predecssor lists are kept sorted by vertex number. Thus the cost of merging two predecessor lists and eliminating duplicates is proportional to the sum of their lengths.

We are now ready to compute the cost of $O(n)$ top-level calls to MERGE. Since there are only n vertices, these top-level calls can result in only $n - 1$ additional calls in step 4, or $O(n)$ calls in all. There are $O(n)$ calls to FIND from step 1, $O(mn)$ from step 4, and $O(m^2)$ from CONGRUENT, or $O(m^2)$ calls in all. There are no more than $n - 1$ calls to UNION. In the fast implementation of UNION and FIND analyzed in [9], UNION takes constant time and $O(m^2)$ calls to FIND take $O(m^2)$ time. The total cost of splicing predecessor lists is $O(n^2)$. Thus, the asymptotic worst-case time for $O(n)$ merges is $O(m^2)$.

The double loop used in step 4 to find new congruent pairs is not very sophisticated. If the set of predecessors of the two vertices is lexicographically sorted on the sequences of their successors' equivalence classes, then congruent vertices will be adjacent in the sorted list. The cost of finding all new congruent pairs is proportional to the sum of the lengths of the predecessor lists instead of to the product. If step 4 is changed in this way, the time bound for the algorithm becomes $O(mn)$.

In the sophisticated algorithm of Downey, Sethi, and Tarjan [3], the vertices are kept in a hash table keyed by the list of equivalence classes of their successors. Step 4 can be implemented so that only the vertices in the shorter predecessor list need be rehashed. The average time for $O(n)$ merges using their algorithm is $O(m \log m)$.

We have implemented both the $O(m^2)$ and the $O(m \log m)$ algorithms. The more sophisticated algorithm is not faster for our applications. The reason is that the predecessor lists are short, so that the double loop runs at about the same speed as in the more sophisticated method.

3. *The Quantifier-Free Theory of Equality*

In this section, we show how the decision problem for the quantifier-free theory of equality with uninterpreted function symbols reduces to the congruence closure problem.

The language of the quantifier-free theory of equality consists of variables, uninterpreted function symbols, the usual Boolean connectives, and the predicate =. An example of a formula in the theory is $x = y \supset f(x) = f(y)$. To determine the validity of a formula, it suffices to determine the unsatisfiability of the disjunctive normal form of its negation.

The disjunction is unsatisfiable if and only if each of its disjuncts is unsatisfiable, so it suffices to describe an algorithm for determining the satisfiability of conjunctions of literals.

DECISION PROCEDURE. This algorithm determines the satisfiability of the conjunction of equalities and disequalities

$$t_1 = u_1 \wedge \cdots \wedge t_p = u_p \wedge r_1 \neq s_1 \wedge \cdots \wedge r_q \neq s_q.$$

1 Construct a graph G which corresponds to the set of all terms appearing in the conjunction For each term t appearing in the conjunction, let $\tau(t)$ be the vertex in G representing t Let R be the identity relation on the vertices of G

2 For $1 \leq i \leq p$, merge $\tau(t_i)$ with $\tau(u_i)$ using the MERGE procedure given above.

3. For $1 \leq i \leq q$, if $\tau(r_i)$ is equivalent to $\tau(s_i)$, the conjunction is unsatisfiable

4. Otherwise, the conjunction is satisfiable

Using the sophisticated version of MERGE, the above procedure determines the satisfiability of a conjunction of length n in average time $O(n \log n)$.

It is straightforward to verify that the algorithm is correct if it returns UNSATISFIABLE. To show that it is correct if it returns SATISFIABLE, we construct an interpretation ψ satisfying F.

Let S be the partition of the vertices of G corresponding to the final equivalence relation. ψ maps individual variables into elements of S (that is, equivalence classes of vertices) and k-ary function symbols into functions from S^k to S.

If x is an individual variable, let $\psi(x)$ be the equivalence class of any vertex labeled x with outdegree 0. (Since all such vertices are equivalent, this definition is unambiguous.) If f is a function variable, let $\psi(f)(Q_1, \ldots, Q_k)$ be the equivalence class of any vertex v in V such that $\lambda(v) = f$, $\delta(v) = k$, and for all i between 1 and k, $v[i] \in Q_i$. ($\psi(f)$ is well-defined because, if two vertices u and v both satisfy these conditions, they are congruent and therefore in the same equivalence class.) If no such vertex v exists, then $\psi(f)(Q_1, \ldots, Q_k)$ is arbitrary.

It is straightforward to verify that for all terms t in F, $\psi(t)$ is the equivalence class of $\tau(t)$. Thus ψ satisfies F, since $\tau(t_i)$ is in the same equivalence class as $\tau(u_i)$ for each i, and $\tau(r_i)$ is in a different equivalence class than $\tau(s_i)$ for each i.

4. *Extension to Theories of List Structure*

The congruence closure algorithm forms the basis for a fast decision procedure for determining satisfiability in the quantifier-free theory of list structure with uninterpreted function symbols. The language of this theory is the language of the quantifier-free theory of equality plus distinguished function symbols CAR, CDR, and CONS and predicate symbol ATOM, satisfying the following axioms:

$$\begin{aligned}&\mathrm{CAR}(\mathrm{CONS}(x, y)) = x,\\ &\mathrm{CDR}(\mathrm{CONS}(x, y)) = y,\\ &\neg\mathrm{ATOM}(x) \supset \mathrm{CONS}(\mathrm{CAR}(x), \mathrm{CDR}(x)) = x,\\ &\neg\mathrm{ATOM}(\mathrm{CONS}(x, y)).\end{aligned} \tag{1}$$

CAR, CDR, CONS, and ATOM are the well-known functions and predicates of LISP. An example of a theorem in this theory is

$$\mathrm{CAR}(x) = \mathrm{CAR}(y) \wedge \mathrm{CDR}(x) = \mathrm{CDR}(y) \wedge \neg\mathrm{ATOM}(x) \wedge \neg\mathrm{ATOM}(y) \supset f(x) = f(y).$$

Notice that we do not restrict the domain of the LISP functions to noncircular lists, so that a formula like $\mathrm{CAR}(x) = x$ is satisfiable. If we include axioms enforcing acyclicity of list structure (as in Pure LISP) and exclude uninterpreted function symbols, a faster algorithm is possible than the one described here. Oppen [7] describes a decision procedure which determines the satisfiability of conjunctions over the quantifier-free theory of (pure) LISP in linear time.

The algorithm represents terms by vertices in a directed graph as in Section 3. The basic idea of the decision procedure is to add all relevant instances of axiom schema (1) to this graph. For each term CONS(x, y) represented in the graph, we will add the equalities x = CAR(CONS(x, y)) and y = CDR(CONS(x, y)) to the graph.

We assume that each literal ATOM(t) appearing negatively has been eliminated from the conjunction and replaced by an equality t = CONS(u, v), where u and v are variables appearing nowhere else in the formula. Therefore, the only literals involving ATOM are positive.

DECISION PROCEDURE. This algorithm determines the satisfiability of a conjunction F of the form

$$\begin{aligned}&\mathrm{ATOM}(u_1) \wedge \mathrm{ATOM}(u_2) \wedge \cdots \wedge \mathrm{ATOM}(u_q) \wedge \\ &v_1 = w_1 \wedge \cdots \wedge v_r = w_r \wedge \\ &x_1 \neq y_1 \wedge \cdots \wedge x_s \neq y_s,\end{aligned}$$

where the terms in the literals may contain uninterpreted function symbols as well as the functions CAR, CDR, and CONS.

1 Construct a graph G which corresponds to the set of all terms appearing in the conjunction For each term t appearing in the conjunction, let $\tau(t)$ be the vertex in G representing t For $1 \leq i \leq r$, call MERGE($\tau(v_i)$, $\tau(w_i)$)

2 For each vertex u in G labeled CONS, add vertices v, labeled CAR, and w, labeled CDR, both with outdegree 1, such that $v[1] = w[1] = u$ Call MERGE(v, $u[1]$) and MERGE(w, $u[2]$) (That is, given a term CONS(x, y), add vertices representing CAR(CONS(x, y)) and CDR(CONS(x, y)) and merge them with the vertices representing x and y)

3 For i from 1 to s, if $\tau(x_i)$ is equivalent to $\tau(y_i)$, return UNSATISFIABLE For i from 1 to q, if the equivalence class of $\tau(u_i)$ contains a vertex labeled CONS, return UNSATISFIABLE Otherwise, return SATISFIABLE

If the length of the formula F is n, the size of G after step 2 is $O(n)$. The average time required by this decision procedure to determine the satisfiability of a conjunction of literals is therefore $O(n \log n)$ using the fast congruence closure algorithm.

PROOF OF CORRECTNESS. It is straightforward to verify that the algorithm is correct if it returns UNSATISFIABLE. Suppose that it returns SATISFIABLE; we construct an interpretation satisfying F.

Let S_0 be the partition of the vertices of G corresponding to the final equivalence relation. We define two functions CAR_0 and CDR_0 from S_0 to S_0, and a function CONS_0 from a subset of $S_0 \times S_0$ to S_0. If the equivalence class Q contains a vertex v with a predecessor u labeled CAR, then $\mathrm{CAR}_0(Q)$ is the equivalence class of u; otherwise $\mathrm{CAR}_0(Q)$ is arbitrary. If Q contains a vertex v with a predecessor u labeled CDR, then $\mathrm{CDR}_0(Q)$ is the equivalence class of u; otherwise $\mathrm{CDR}_0(Q)$ is arbitrary. The pair (Q_1, Q_2) is in the domain of CONS_0 only if there exists a vertex v labeled CONS such that $v[1] \in Q_1$ and $v[2] \in Q_2$; in this case $\mathrm{CONS}_0(Q_1, Q_2)$ is the equivalence class of v. Note that CAR_0, CDR_0, and CONS_0 are well defined because the graph is closed under congruences.

Unfortunately, CONS_0 is not a total function. To construct an interpretation, we must extend CONS_0 to be defined on all of $S_0 \times S_0$. We first extend it to a function CONS_1 which agrees with CONS_0 where CONS_0 is defined, and otherwise just returns the ordered pair of its arguments. Since CONS_1 returns elements of $S_0 \times S_0$, the range S_0 of the interpretation must be extended to a set S_1 that includes both S_0 and part of $S_0 \times S_0$. But then CONS_1 is not total and must be extended so that it is defined on all of $S_1 \times S_1$. To construct an interpretation we repeat this extension step infinitely many times.

More precisely, suppose that we have defined the first $i + 1$ quadruples in the infinite sequence (S_0, CONS_0, CAR_0, CDR_0), (S_1, CONS_1, CAR_1, CDR_1), ..., (S_i, CONS_i, CAR_i, CDR_i), ... We define the next quadruple (S_{i+1}, CONS_{i+1}, CAR_{i+1}, CDR_{i+1}) by the following rules.

Let D_i be the domain of CONS_i.

(1) $S_{i+1} = S_i \cup S_i \times S_i - D_i$.

(2) The domain of CONS_{i+1} is $S_i \times S_i$. $\text{CONS}_{i+1}(x, y) = \text{CONS}_i(x, y)$ if (x, y) is in the domain of CONS_i; $\text{CONS}_{i+1}(x, y) = (x, y)$ otherwise.

(3) $\text{CAR}_{i+1}(x) = \text{CAR}_i(x)$ if $x \in S_i$. Otherwise $x \in S_i \times S_i - D_i$ and is thus an ordered pair (y, z); in this case define $\text{CAR}_{i+1}(x) = y$.

(4) $\text{CDR}_{i+1}(x) = \text{CDR}_i(x)$ if $x \in S_i$. Otherwise $x \in S_i \times S_i - D_i$ and is thus an ordered pair (y, z); in this case define $\text{CDR}_{i+1}(x) = z$.

We now show that, for each i, CAR_i, CDR_i, and CONS_i have the following two properties.

(1) If (x, y) is in the domain of CONS_i, then $\text{CAR}_i(\text{CONS}_i(x, y)) = x$ and $\text{CDR}_i(\text{CONS}_i(x, y)) = y$.

(2) If x is in the range of CONS_i, then $(\text{CAR}_i(x), \text{CDR}_i(x))$ is in the domain of CONS_i, and $\text{CONS}_i(\text{CAR}_i(x), \text{CDR}_i(x)) = x$.

Consider first the base case, when $i = 0$ and x and y are equivalence classes. If (x, y) is in the domain of CONS_0, then there is a vertex u with $\lambda(u) = \text{CONS}$, $u[1] \in x$, and $u[2] \in y$. Since u is a CONS, two vertices v and w labeled CAR and CDR, respectively, were added as predecessors of u. The vertices v and w satisfy the requirements of the definitions of CAR_0 and CDR_0, so $\text{CAR}_0(\text{CONS}_0(x, y))$ is the equivalence class of v and $\text{CDR}_0(\text{CONS}_0(x, y))$ is the equivalence class of w. Furthermore the pairs $(v, u[1])$ and $(w, u[2])$ were added to R in step 2, so v and w are in the equivalence classes x and y, respectively. The proof that CAR_0, CDR_0, and CONS_0 have the second property is similar.

Suppose now that CONS_i, CAR_i, and CDR_i satisfy properties 1 and 2. If (x, y) is in the domain of CONS_i, then $\text{CONS}_{i+1}(x, y) = \text{CONS}_i(x, y)$; it follows that $\text{CAR}_{i+1}(\text{CONS}_{i+1}(x, y)) = \text{CAR}_i(\text{CONS}_i(x, y)) = x$ and $\text{CDR}_{i+1}(\text{CONS}_{i+1}(x, y)) = \text{CDR}_i(\text{CONS}_i(x, y)) = y$ by the induction hypothesis. Otherwise, $\text{CONS}_{i+1}(x, y)$ is the ordered pair (x, y), and $\text{CAR}_{i+1}(\text{CONS}_{i+1}(x, y)) = x$ and $\text{CDR}_{i+1}(\text{CONS}_{i+1}(x, y)) = y$ by definition. A similar argument shows that CONS_{i+1}, CAR_{i+1}, and CDR_{i+1} satisfy property 2.

It follows by induction that, for all i, the functions CAR_i, CDR_i, and CONS_i have the two properties.

Let S' be the union of all the S_i. Let $\text{CAR}'(x)$ be $\text{CAR}_i(x)$ for the first i such that $x \in S_i$. Let CDR′ and CONS′ be defined similarly. It follows that CAR′, CDR′, and CONS′ have properties 1 and 2 and that CONS′ is defined on all of $S' \times S'$.

We are finally ready to define an interpretation ψ satisfying F. The range of ψ is S'. ψ interprets CAR, CDR, and CONS as CAR′, CDR′, and CONS′. An element of S' is interpreted to be nonatomic if and only if it is in the range of CONS′. If f is an uninterpreted function symbol, $Q_1, \ldots, Q_k$ are in S, and there exists a vertex v such that $\lambda(v) = f$, $\delta(v) = k$, and $v[i] \in Q_i$ for each i from 1 to k, then $\psi(f)(Q_1, \ldots, Q_k)$ is the equivalence class of v. If this definition does not determine the value of $\psi(f)$, then the value is arbitrary.

It follows from properties 1 and 2 and the fact that the set of nonatoms is exactly the range of CONS′ that this interpretation satisfies axiom schema (2). It remains to show that ψ satisfies F.

It is straightforward to show that for each term t in the original formula, $\psi(t)$ is the equivalence class of $\tau(t)$. But $\tau(v_i)$ and $\tau(w_i)$ have been merged for each i from 1 to r, so ψ satisfies the equalities in F. $\tau(x_i)$ and $\tau(y_i)$ are in different equivalence classes (since step 3 returned SATISFIABLE), so ψ satisfies the disequalities in F. Finally, no equivalence class of any $\tau(u_i)$ contains a node labeled CONS; hence these classes are not in the range of CONS_0. They are certainly not in the range of any of the other functions CONS_i, so they are interpreted as atoms by ψ. Hence ψ satisfies F.

This completes the proof of correctness of the decision procedure.

Somewhat surprisingly, when the result of a selector function on an atom is specified by the axioms, the problem of determining the satisfiability of a conjunction of literals becomes NP-complete. Consider the following axioms for the theory of CAR, CDR, and CONS with the single atom NIL:

$$\begin{aligned}
&\mathrm{CAR}(\mathrm{CONS}(X, Y)) = X,\\
&\mathrm{CDR}(\mathrm{CONS}(X, Y)) = Y,\\
&X \neq \mathrm{NIL} \supset \mathrm{CONS}(\mathrm{CAR}(X), \mathrm{CDR}(X)) = X,\\
&\mathrm{CONS}(X, Y) \neq \mathrm{NIL},\\
&\mathrm{CAR}(\mathrm{NIL}) = \mathrm{CDR}(\mathrm{NIL}) = \mathrm{NIL}.
\end{aligned}$$

We show that the problem of determining the satisfiability in this theory of a conjunction of equalities and disequalities between terms containing CAR, CDR, CONS, NIL, and uninterpreted function signs is NP-complete.

It is straightforward to show that the problem is in NP, since a nondeterministic program can guess the equivalence relation on the set of terms in the conjunction and then check that the equivalence relation does not violate any of the above axioms or the substitutivity of equality.

To show that the problem is NP-hard, we will reduce the 3-CNF satisfiability problem for propositional calculus to it. (See [2].)

Let $P_1, \ldots, P_n$ be propositional variables and F a conjunction of three-element clauses over the P_i. We construct a conjunction G of equalities and disequalities between list-structure terms involving CAR, CDR, CONS, NIL, and the $2n$ variables $X_1, Y_1, \ldots, X_n, Y_n$ such that G is satisfiable if and only if F is.

The first part of G is

$$\begin{aligned}
&\mathrm{CAR}(X_1) = \mathrm{CAR}(Y_1) \wedge \mathrm{CDR}(X_1) = \mathrm{CDR}(Y_1) \wedge X_1 \neq Y_1 \wedge\\
&\mathrm{CAR}(X_2) = \mathrm{CAR}(Y_2) \wedge \mathrm{CDR}(X_2) = \mathrm{CDR}(Y_2) \wedge X_2 \neq Y_2 \wedge\\
&\quad\vdots\\
&\mathrm{CAR}(X_n) = \mathrm{CAR}(Y_n) \wedge \mathrm{CDR}(X_n) = \mathrm{CDR}(Y_n) \wedge X_n \neq Y_n.
\end{aligned} \tag{2}$$

For no i can X_i and Y_i both be non-NIL, since then X_i and Y_i would be equal by the third axiom and the substitutivity of equality. One of them must be NIL and the other CONS(NIL, NIL).

Given an interpretation ψ for G, we construct an interpretation ϕ for F by defining $\phi(P_i)$ to be TRUE if and only if $\psi(X_i) = \mathrm{NIL}$. The remaining conjuncts in G guarantee that ψ satisfies G if and only if ϕ satisfies F.

We demonstrate the construction with an example. If one of the clauses of F is $P_1 \vee \neg P_2 \vee P_3$, we want to add a conjunct to G which is equivalent to $(X_1 = \mathrm{NIL} \vee X_2 \neq \mathrm{NIL} \vee X_3 = \mathrm{NIL})$. In light of (2), this is equivalent to

$$\neg(Y_2 = \mathrm{NIL} \wedge X_2 = \mathrm{NIL} \wedge Y_3 = \mathrm{NIL}),$$

or to the single literal

$$\mathrm{CONS}(Y_2, \mathrm{CONS}(X_2, Y_3)) \neq \mathrm{CONS}(\mathrm{NIL}, \mathrm{CONS}(\mathrm{NIL}, \mathrm{NIL})).$$

Note that we have shown the problem is NP-hard even without uninterpreted function symbols. A similar construction can be used whenever the result of a selector function on an atom is specified. The problem is also NP-complete with the axiomatization (1) if predicates are interpreted as Boolean-valued functions and literals such as $F(\mathrm{ATOM}(x)) \neq F(\mathrm{ATOM}(y))$ are allowed.

ACKNOWLEDGMENT. We are indebted to Bob Tarjan for many helpful discussions.

REFERENCES

1. ACKERMANN, W *Solvable Cases of the Decision Problem* North-Holland, Amsterdam, 1954
2. AHO, A V, HOPCROFT, J E, AND ULLMANN, J D *The Design and Analysis of Computer Algorithms* Addison-Wesley, Reading, Mass 1974

3. DOWNEY, P. J , SETHI, R , AND TARJAN, R E. Variations on the common subexpression problem To appear in *J. ACM.*

4. KOZEN, D. Complexity of finitely represented algebras. Proc. 9th Annual ACM Symp. on Theory of Comptg., Boulder, Colo., May 1977, pp. 164–177.

5. NELSON, C G., AND OPPEN, D. C Fast decision procedures based on UNION and FIND. Proc Eighteenth Annual Symp on Foundations of Comptr. Sci , Providence, R I., 1977 (This is an earlier version of the present paper.)

6 NELSON, C G., AND OPPEN, D. C Simplification by cooperating decision procedures To be published

7 OPPEN, D C. Reasoning about recursively defined data structures Proc 5th ACM Symp on Principles of Programming Languages, Tucson, Ariz , January 1978, pp 151–157

8. SHOSTAK, R An algorithm for reasoning about equality *Comm ACM 21,* 7 (July 1978), 583–585

9 TARJAN, R. E Efficiency of a good but not linear set union algorithm *J ACM 22,* 2 (April 1975), 215–225

RECEIVED JUNE 1978, REVISED APRIL 1979, ACCEPTED APRIL 1979