Stanford Artificial intelligence Laboratory
Memo MU-309

February 1978

Computer Science Department
Report No. STAN-CS-77-646

FAST DECISION ALGORITHMS BASED ON CONGRUENCE CLOSURE

by

Greg Nelson
and
Derek C. Oppen
Stanford Verification Group

Research sponsored by

National Science Foundation
and
Hertz Foundation

COMPUTER SCIENCE DEPARTMENT
Stanford University

Stanford Artificial Intelligence Laboratory
Memo AIM-309

February 1978

Computer Science Department
Report No. STAN-CS-77-646

FAST DECISION ALGORITHMS BASED ON CONGRUENCE CLOSURE

by

Greg Nelson
and
Derek C. Oppen
Stanford Verification Group

We define the notion of the *congruence* closure of a relation on a graph and give a simple algorithm for computing it. We then give decision procedures for the quantifier-free theory of equality and the quantifier-free theory of LISP list structure, both based on this algorithm. The procedures are fast enough to be practical in mechanical theorem proving: each procedure determines the satisfiability of a conjunction of length n of literals in time $O(n^2)$. We also show 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. We have implemented the decision procedures in our simplifier for the Stanford Pascal Verifier.

An earlier version of this paper appeared in the Proceedings of the 18th Annual Symposium on Foundations of Computer Science, Providence, October 1977. This research was supported by tire National Science Foundation under contract MCS 76000327 and by the Fannie and John Hertz Foundation.

1. Introduction

Let $G=(V, E)$ be a directed graph with labelled vertices and R a relation on V. The *congruence closure* $\sim$ of R on G is the unique minimal extension of R such that $\sim$ is an equivalence relation and any two vertices of G are equrvalent under $\sim$ if they have the same label and the same outdegree, and all their corresponding successors are equivalent under $\sim$.

In section 2, we give a simple algorithm for computing the congruence closure of R on G which requires $O(mn + k)$ time, where n is the number of vertices in G, m is the number of edges in G, and k is the number of pairs in R.

In section 3, we describe a decision procedure for the quantifier-free theory of equality with uninterpreted function symbols based on the congruence closure algorithm. The algorithm determines the satisfiability of a conjunction of equalities and disequalities of length n in time $O(n^2)$.

In section 4, we describe a decision procedure for **the** theory of LISP list structure with the usual functions CAR, CONS, and CDR and the predicate LISTP, which asserts that its argument is non-atomic. The axioms for the theory are:

$$\mathrm{CAR}(\mathrm{CONS}(X, Y)) = X$$
$$\mathrm{CDR}(\mathrm{CONS}(X, Y)) = Y$$
$$\mathrm{LISTP}(X) \supset \mathrm{CONS}(\mathrm{CAR}(X), \mathrm{CDR}(X)) = X$$
$$\mathrm{LISTP}(\mathrm{CONS}(X, Y))$$

The decision procedure determines the satisfiability of a conjunction of length n of literals in time $O(n^2)$. The terms in the literals may contain uninterpreted function signs.

We also show in section 4 that the satisfiability problem for conjunctions of literals is NP-complete if the following axioms are used instead of the above axioms:

$$\mathrm{CAR}(\mathrm{CONS}(X, Y)) = X$$
$$\mathrm{CDR}(\mathrm{CONS}(X, Y)) = Y$$
$$X \neq \mathrm{NIL} \supset \mathrm{CONS}(\mathrm{CAR}(X), \mathrm{CDR}(X)) = X$$
$$\mathrm{CONS}(X, Y) \neq \mathrm{NIL}$$
$$\mathrm{CAR}(\mathrm{NIL}) = \mathrm{NIL}$$
$$\mathrm{CDR}(\mathrm{NIL}) = \mathrm{NIL}$$

In section 5, we give some notes on the implementation of our algorithms.

2. The Congruence Closure Algorithm

Let $G = (V, E)$ be a directed graph with labelled 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 $i \ne j$. Let $|V| = n$, $|E| = m$. We assume that 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 $1 \le i \le \delta(u)$, $(u[i], v[i]) \in R$. There is a unique minimal extension $\sim$ of R which satisfies 1. $\sim$ is an equivalence relation, and 2. if u and v are congruent under $\sim$, then $u \sim v$. The relation $\sim$ is called the *congruence closure* of R. In the congruence closure, two vertices are equivalent if they have the same label and the same outdegree, and all their corresponding successors are equivalent.

In this section we describe an algorithm for computing congruence closures which requires $O(mn + k)$ time and $O(m)$ space in the worst case, where k is the number of pairs in R.

We represent an equivalence relation by its corresponding partition, that is, by the set of its equivalence classes. An equivalence class is represented by a list of its members. We use two procedures for operating on the partition: UNION and FIND. UNION(u, v) combines the equivalence classes of vertices u and v. FIND(u) returns the equivalence class of vertex u.

In the most straightforward implementation of UNION and FIND, each vertex u contains a field EQCLASS(u), pointing to the equivalence class of u, that is, to the head of the list of vertices representing its equivalence class. If u and v are equivalent, then EQCLASS(u) and EQCLASS(v) point to the same list. FIND(v) simply returns EQCLASS(v). UNION(u, v) updates the EQCLASS pointer of all the vertices in v's equivalence class to point to u's equivalence class, and destructively appends the former equivalence class to the latter. In this simple implementation, FIND takes constant time while UNION takes time linear in the sum of the lengths of the two equivalence classes being merged and thus takes worst case time $O(n)$. [Tarjan 1975] analyzes an implementation of UNION and FIND which is much faster in theory and in practice, but which affects only the constant factor of the time bound of our simple congruence closure algorithm.

For each vertex u, define the *signature* of u to be the tuple $<\lambda(u), v_1, \ldots, v_k>$, where k is the outdegree of u and v_i is the first vertex in the equivalence class of $u[i]$. The signature of a vertex is thus an encoding of its label together with the list of its successors' equivalence classes.

Two vertices are congruent if and only if they have identical signatures. When two equivalence classes are merged, the signatures of some vertices in the graph may be changed. To find all new congruences, we sort the vertices on the basis of their signatures. Congruent vertices will be adjacent in the sorted list.

Congruence Closure Algorithm

This algorithm computes the congruence closure of a relation R on a graph G.

i. For each of the k pairs (u, v) in R, if FIND(u) $\neq$ FIND(v) then UNION(u, v).

2. Sort the vertices in G on the basis of their signatures. Let L be the resulting sorted list and L[i] the ith vertex in L.

3. For i $\leftarrow$ 1 to n- 1, if L[i] and L[i+1] have the same signature but FIND(L[i]) $\neq$ FIND(L[i+1]), then UNION(L[i], L[i+ 1]).

4. If any unions were done in step 3, then go to 2. Otherwise, return.

The algorithm is obviously correct. Since there are only n vertices in G, there can be at most n - 1 calls to UNION. Therefore the total cost of calls to UNION in the algorithm is $O(n^2)$. Using lexicographic sorting, the cost of each pass through steps 2, 3 and 4 is O(m + n), excluding the cost of any calls to UNION. There can be at most n passes through these steps of the algorithm. It follows that the worst case running time of the algorithm is O(mn + k). The algorithm requires linear space.

Faster congruence closure algorithms are possible. [Johnson and Tarjan 1977] describe an algorithm which requires, depending on its implementation, $O(m (\log n)^2)$ time and O(m) space in the worst case, or O(m log n) time'and O(mn) space in the worst case, or O(m log n) time on the average and O(m) space. [Downey, Samet and Sethi 1978] have discovered essentially the same algorithm. [Kozen 1977] also gives a polynomial time algorithm.

There is a directional dual to the problem of constructing the congruence closure of a relation R: constructing the equivalence relation ~ containing R such that if u ~ v, then u[i] ~ v[i] for all i such that $1 \leq i \leq \delta(u) = \delta(v)$. In this dual problem, if two vertices are equivalent, then so are all their corresponding successors. This is essentially the problem of determining the equivalence classes of states of a finite automaton. There is an O(n a(n)) algorithm for solving this problem ([Aho, Hopcroft and Ulimann 1974]), where a(n) is the inverse of a version of Ackermann's function.

3. The Quantifier-free Theory of Equality

The language of the quantifier-free theory of equality consists of variables, uninterpreted function symbols, the usual boolean connectives and the predicate =. Every term is either an atomic symbol (which represents an individual variable) or an expression of the form $f(t_1, \ldots, t_k)$ where f is an atomic symbol and each t_i is a term. An example of a formula in the theory is $x = y \supset f(x) = f(y)$. The theory was first proved decidable by [Ackermann 1954].

In this section we give a decision procedure which determines the satisfiability of a conjunction F of literals in time $O(|F|^2)$, where $|F|$ is the length of F. The decision procedure represents the terms of the conjunction by vertices in a directed graph and uses the congruence closure algorithm to make all possible inferences following from the substitutivity of equality.

We represent a term t by the root of a tree T(t) in the obvious way: if t is atomic, T(t) contains a single vertex labelled t with no successors; if t is of the form $f(t_1, \ldots, t_k)$, $T(t)$ has a root labelled f, whose successors are the roots of $T(t_1), \ldots, T(t_k)$. *We* call the root of $T(t)$ the *representative* of t; we use $T(t)$ to denote this root as well as the tree itself when the context makes the meaning clear.

The decision algorithm first constructs the disjoint union of the trees representing the terms in the conjunction. It then merges (makes equivalent) each pair of vertices which represents a pair of terms asserted equal in the formula and closes this initial relation under congruences. We will show that two vertices are equivalent in the congruence closure if and only if the terms they represent are entailed equal by the formula. It therefore suffices for the decision algorithm to check if the representatives of any two terms asserted unequal are equivalent in the congruence closure. If so, the algorithm returns UNSATISFIABLE; if not, it returns SATISFIABLE.

Figures 1 and 2 illustrate how our decision procedure determines that the formula $F \equiv f(a) = a \wedge g(f(f(a)), a) \neq g(a, a)$ is unsatisfiable. The algorithm first constructs the disjoint union G of the trees representing the four terms a, f(a), g(f(f(a)), a), and g(a, a). (In the figure, vertices have been numbered for the purpose of this description.) The algorithm then merges vertices 1 and 2, which represent the terms a and f(a) asserted equal in F. The result is illustrated in figure 1; we use a dotted line to represent the fact that vertices 1 and 2 are equivalent. The decision algorithm next computes the congruence closure on G of the initial equivalence relation in which vertices 1 and 2 are equivalent. Figure 2 illustrates the resulting equivalence relation: vertices 1, 2, 3, 5, 6, 7, 8, 10 and 11 are all equivalent to each other, as are vertices 4 and 9. In the final step, the decision algorithm checks whether the representatives of any terms asserted unequal by F are equivalent in the congruence closure. In our example, the terms represented by vertices 4 and 9 were asserted unequal, but have been merged. The decision algorithm therefore terminates with UNSATISFIABLE.

Decision Algorithm

Let $F \equiv t_1 = u_1 \wedge \ldots \wedge t_p = u_p \wedge r_1 \neq s_1 \wedge \ldots \wedge r_q \neq s_q$ be a conjunction of equalities and disequalities. This algorithm determines whether F is satisfiable.

1. Construct a graph G, the disjoint union of $T(t_1), T(u_1), \ldots, T(t_p), T(u_p), T(r_1), T(s_1), \ldots, T(r_q), T(s_q)$. Let R be $\{(T(t_i), T(u_i)) \mid 1 \leq i \leq p\}$. Construct $\sim$, the congruence closure of R on G.

2. For i from 1 to q, if $T(r_i) \sim T(s_i)$ return UNSATISFIABLE. Otherwise, return SA TISFIA BLE.

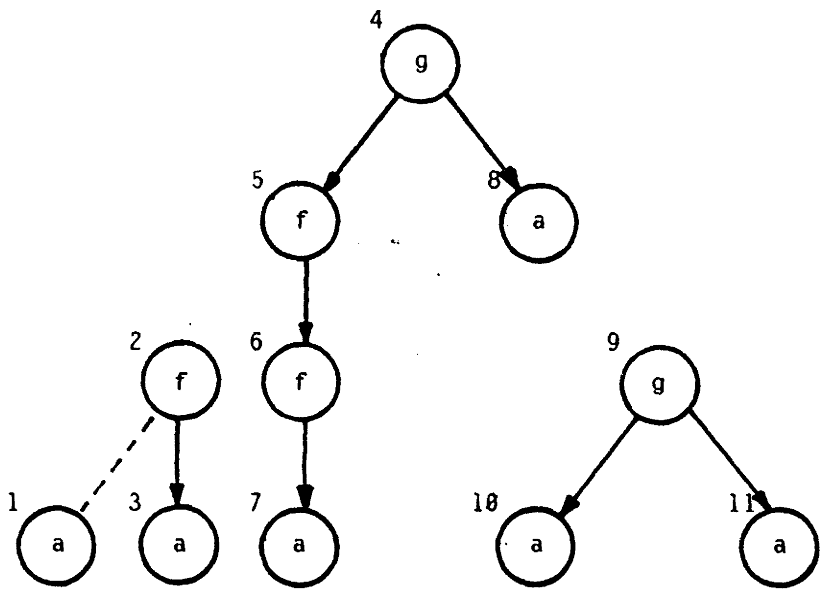

Figure 1

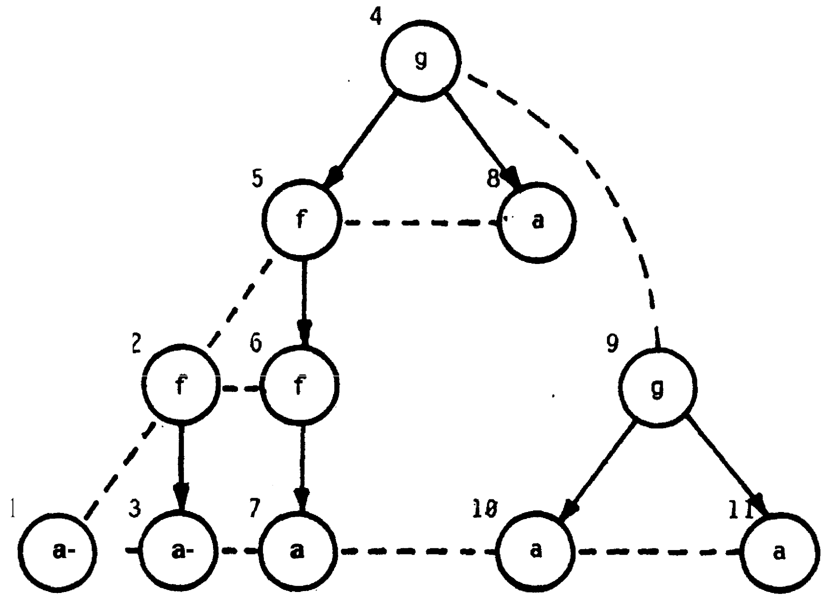

Figure 2

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 equivalence relation $\sim$. 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 labelled x with outdegree zero. (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 chat $\lambda(v) = f$, $\delta(v) = k$, and for all i between i 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.

[Shostak 1977] proves a similar result.

Let m be the number of edges and n the number of vertices in C. Since $m \leq |F|$, $n \leq |F|$, and $q \leq |F|$, step 1 requires time $O(|F|^2)$, step 2 time $O(|F|)$, and the whole algorithm time $O(|F|^2)$.

As presented, the algorithm is not incremental in the sense of [Nelson and Oppen 1978]; that is, it does not accept literals one by one and determine unsatisfiability as soon as the conjunction becomes inconsistent. It is easy to modify the procedure so that it is incremental. We keep a list of all pairs of vertices which have been asserted unequal, adding a new pair to the list every time a disequality is presented. The list never contains more than q pairs, so checking if a merge violates some disequality requires O(q) time. Since there can be at most n-1 merges, whether or not they are done incrementally, this incremental version of the algorithm spends O(nm) time in the congruence closure algorithm and O(nq) time checking if merges violate disequalities, or $O(|F|^2)$ time in all.

4. Extension to Theories of List Structure

In this section we show how the decision procedure given in the previous section can be modified to handle the function symbols CAR, CDR and CONS and the predicate LISTP in addition to uninterpreted function symbols. An example of a formula in this theory is $CAR(x) = CAR(y) \wedge CDR(x) = CDR(y) \wedge LISTP(x) \wedge LISTP(y) \supset f(x) = f(y)$. The decision procedure determines the satisfiability of a conjunction of length n of literals in time $O(n^2)$.

We assume the LISP functions satisfy the following axioms.

$$\begin{aligned}
&\mathrm{CAR}(\mathrm{CONS}(x, y)) = x \\
&\mathrm{CDR}(\mathrm{CONS}(x, y)) = y \qquad\qquad (1)\\
&\mathrm{LISTP}(x) \supset \mathrm{CONS}(\mathrm{CAR}(x), \mathrm{CDR}(x)) = x \\
&\mathrm{LISTP}(\mathrm{CONS}(x, y))
\end{aligned}$$

Notice that we do not restrict the domain of the LISP functions to non-circular lists, so that a formula like CAR(x) = x is satisfiable. If we include axioms enforcing acyclicity of list structure, and exclude uninterpreted function symbols, a linear algorithm is possible. [Oppen1978] describes a decision algorithm which determines the satisfiability of a conjunction of length n in time O(n).

The algorithm represents terms by vertices in a directed graph as in section 3. The basic idea of our decision algorithm is to add all reievant instances of (I) 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.

It is convenient in the statement and proof of correctness of the algorithm to assume that each positive literal LISTP(t) 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. We can therefore assume that the oniy literals involving LISTP are negative.

Decision Algorithm

This algorithm determines the satisfiability of a conjunction F of the form:

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

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

I. Construct G, the disjoint union of $\mathcal{T}(u_1), \ldots \mathcal{T}(u_q), \mathcal{T}(v_1), \ldots \mathcal{T}(v_r), \mathcal{T}(w_1), \ldots \mathcal{T}(w_r), \mathcal{T}(x_1), \ldots \mathcal{T}(x_s), \mathcal{T}(y_1), \ldots \mathcal{T}(y_s)$. Let R be $\{(\mathcal{T}(v_i), \mathcal{T}(w_i)) \mid 1 \leq i \leq r\}$.

2. For each vertex u in G labelled CONS, add vertices v, labelled CAR, and w, labelled CDR, both with outdegree I, such that v[1] = w[1] = u. Add the pairs (v, u[1]) and (w, u[2]) to R. (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 for x and y.)

3. Construct ~, the congruence closure of R on G.

4. For i from 1 to s, if $\tau(x_i) = \tau(y_i)$, return UNSATISFIABLE. For i from 1 to q, if the equivalence class of $\tau(u_i)$ contains a vertex labelled CONS, return UNSATISFIABLE. Otherwise, return SATISFIABLE.

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

Let S_0 be the partition of the vertices of G corresponding to the final equivalence relation $\sim$. 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 labelled CAR, then $CAR_0(Q)$ is the equivalence class of u; otherwise $CAR_0(Q)$ is arbitrary. If Q contains a vertex v with a predecessor u labelled CDR, then $CDR_0(Q)$ is the equivalence class of u; otherwise $CDR_0(Q)$ is arbitrary. The pair (Q_1, Q_2) is in the domain of $CONS_0$ only if there exists a vertex v labelled CONS such that $v[1] \in Q_1$ and $v[2] \in Q_2$; in this case $CONS_0(Q_1, Q_2)$ 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.

CAR_0, CDR_0 and $CONS_0$ have the following two properties:

1. If (Q_1, Q_2) is in the domain of $CONS_0$, then $CAR_0(CONS_0(Q_1, Q_2)) = Q_1$ and $CDR_0(CONS_0(Q_1, Q_2)) = Q_2$.

2. If Q is in the range of $CONS_0$, then $(CAR_0(Q), CDR_0(Q))$ is in the domain of $CONS_0$, and $CONS_0(CAR_0(Q), CDR_0(Q)) = Q$.

Proof of property 1: If (Q_1, Q_2) *is* in the domain of $CONS_0$, then there is a vertex u with $\lambda(u) = $ CONS, $u[1] \in Q_1$, and $u[2] \in Q_2$. Since u is a CONS, two vertices v and w labelled CAR and CDR respectively were added as predecessors of u. These vertices satisfy the requirements of the definitions of CAR_0 and CDR_0, so $CAR_0(CONS_0(Q_1, Q_2))$ is the equivalence class of v and $CDR_0(CONS_0(Q_1, Q_2))$ 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 Q_1 and Q_2 respectively.

The proof that the functions have the second property is similar.

To construct an interpretation, we must extend $CONS_0$ so that it is defined on all of $S_0 \times S_0$. We will 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 domain S_0 of the interpretation must be extended to a domain S_1 which includes both S_0 and part of $S_0 \times S_0$. Now $CONS_1$ 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 $(S_0, CONS_0, CAR_0, CDR_0), (S_1, CONS_1, CAR_1, CDR_1), \ldots, (S_i, CONS_i, CAR_i, CDR_i)$, ... We define the next quadruple $(S_{i+1}, CONS_{i+1}, CAR_{i+1}, CDR_{i+1})$ by the following ruies.

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

3. $CAR_{i+1}(x) = CAR_i(x)$ 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 $CAR_{i+1}(x) = y$.

4. $CDR_{i+1}(x) = CDR_i(x)$ 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 $CDR_{i+1}(x) = z$.

It is trivral to verify that if $CONS_i$, CAR_i and CDR_i satisfy properties I and 2, then so do $CONS_{i+1}$, CAR_{i+1} and CDR_{i+1}. Since the properties are satisfied for $i = 0$, they are satisfied for every i. Let S' be the union of ail the S_i. Let CAR'(x) be $CAR_i(x)$, 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' x S'.

We are finally ready to define an interpretation ψ which satisfies 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 non-atomic 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 non-atoms is exactly the range of CONS' that this interpretation satisfies the axioms (1). 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 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 4 returned SATISFIABLE), so ψ satisfies the disequalities in F. Finally, no equivalence class of any $\tau(u_i)$ contains a. node labelled 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 algorithm.

Somewhat surprisingly, when the result of a selector function on an atom is specified by the ax 101115, 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:

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

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 non-deterministic piogram 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-satisfiability problem for propositional calculus to it. (See [A ho, Wopcroft and Ullmann 1974].)

Let $P_1, \ldots, P_n$ be propositional variables and F a conjunction of 3-element clauses over the P_i. We will construct a conjunction G of equalities and disequalities between list-structure terms involving CAR, CDR, CONS, NIL, and the 2n variables $X_1, Y_1, \ldots, X_n$, Yn such that G is satisfiable if and only if F is and $|G| = O(|F|)$.

The first part of G is:

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

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 $\psi(X_i) =$ NIL. The remaining conjuncts in G will 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 $(X_1 = NIL \vee X_2 \neq NIL \vee X_3 = NIL)$. In light of (?), this is equivalent to

$$\neg (Y_2 = NIL \wedge X_2 = NIL \wedge Y_3 = NIL)$$

or to the single literal

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

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 $F(ATOM(x)) \neq F(ATOM(y))$ are allowed.

5. Notes on Implementation

The decision procedures described in this paper have been implemented in our simplifier ([Nelson and Oppen 1978]). A detailed description of their implementation is beyond the scope of this paper since many constraints are Imposed by other components of the simplifier but we will make a few general remarks.

The simplifier represents all terms of formulas as vertices in an graph, essentially as described previously. This graph replaces the conventional list structure representation of formulas used by most theorem provers. It is a global data structure used by all components of the simplifier.

The decision procedures we have implemented are incremental; that is, the graph is kept closed under congruences at all times. Whenever some component of the simplifier deduces an equality, the equality is added to the graph by merging the equivalence classes representing the two terms deduced equal, and the congruence closure algorithm is then run.

Instead of the congruence closure algorithm described in section 2, we use another algorithm which is slower in the worst case but which may be faster in practice. We plan to implement the fast algorithm described in [Johnson and Tarjan 1977] and compare it with our currently implemented version.

Our experience suggests that a fast congruence closure algorithm is the best method available for handling equalities in mechanical theorem provers. .

Acknowledgment

Our original congruence closure algorithm required $O(n^2)$ space. We are indebted to Bob Tarjan for suggesting the improvement incorporated in this paper.

References

[Ackermann 1954] W. Ackermann, “Solvable Cases of the Decision Problem”, North-Holland, Amsterdam.

[Aho, Hopcroft and Ullmann 1974] A. V. Aho, J. E. Hopcroft and J. D. Ullmann, “The Design and Analysis of Computer Algorithms”, Addison-Wesley, Reading, Massachusetts.

[Downey, Samet and Sethi 1978] P. J. Downey, H. Samet and R. Sethi, “Off-line and On-line Algorithms for Deducing Equalities”, Proceedings of the Fifth ACM Symposium on Principles of Programming Languages.

[Johnson and Tarjan 1977] D. S. Johnson and R. E. Tarjan, “Finding Equivalent Expressions”, manuscript.

[Kozen 1977] D. Kozen, “Complexity of Finitely Represented Algebras”, Proceedings of the Ninth Annual ACM Symposium on Theory of Computing.

[Nelson and Oppen 1978] C. G. Nelson and D. C. Oppen, “A Simplifier Based on Efficient Decision Algorithms”, Proceedings of the Fifth ACM Symposium on Principles of Programming Languages.

[Oppen 1978] D. C. Oppen, “Reasoning about Recursively Defined Data Structures”, Proceedings of the Fifth ACM Symposium on Principles of Programming Languages.

[Shostak 1977] R. Shostak, “An Algorithm for Reasoning about Equality”, Proceedings of the Fifth Annual International Conference on Artificial Intelligence, 1977.

[Tarjan 1975] R. E. Tarjan, “Efficiency of a Good But Not Linear Set Union Algorithm”, Journal of the: ACM, pp. 215-225.