Stanford Artificial Intelligence Laboratory
Memo AIM-3 14

July 1978

Computer Science Department
Report No. STAN-(X-78-678

Program Verification Group

REASONING ABOUT RECURSIVELY DEFINED DATA STRUCTURES

by

Derek C. Oppen

Research sponsored by

National Science Foundation

COMPUTER SCIENCE DEPARTMENT
Stanford University

REASONING ABOUT RECURSIVELY DEFINED DATA STRUCTURES

by

Derek C. Oppen

Artificial Intelligence Laboratory
Computer Science Department
Stanford University
Stanford, California

Abstract

A decision algorithm is given for the quantifier-free theory of recursively defined data structures which, for a conjunction of length n, decides its satisfiability in time linear in n. The first-order theory of recursively defined data structures, in particular the first-order theory of LISP list structure {the theory of CONS, CAR and CDR), is shown to be decidable but not elementary recursive.

A preliminary version of this paper appeared in the Proceedings of the Fifth ACM Symposium on Principles of Programming Languages, 1978. This research was supported by the National Science Foundation under contract MCS 76-000327.

1. Introduction

We are interested in the problem of reasoning about data structures and the operations associated with them. Fast techniques (if they exist) for reasoning about data structures are useful in program verification, program manipulation, program optimization; and in proving that implementations of data structures satisfy their abstract definition. More generally, knowledge of the complexity of reasoning about particular classes of data structures gives us some intuition as to their inherent complexity.

We will explore in detail the question of reasoning about a particular class of data structures, the recursively *defined data structures.* These are essentially the *recursive data structures* proposed by [Hoare1975] as a structured alternative to pointers. Most programming languages support such data structures either explicitly or implicitly (they can be mimicked by arrays), but the best known example of them is LISP list structure, with constructor CONS and selectors CAR and CDR.

More precisely, ***recursively** defined data structures* are data structures which have associated with them one constructor function c, and k selector functions $s_1,...,s_k$ with the following abstract structural properties:

I. (Construction)

$$c(s_1(x), s_2(x), ..., s_k(x)) = x$$

II. (Selection)

$$s_1(c(x_1,...,x_k)) = x_1$$
$$s_2(c(x_1,...,x_k)) = x_2$$
$$...$$
$$s_k(c(x_1,...,x_k)) = x_k .$$

III. (Acyclicity)

$$s_1(x) \neq x$$
$$s_2(x) \neq x$$
$$...$$
$$s_k(x) \neq x$$
$$s_1(s_1(x)) \neq x$$
$$s_1(s_2(x)) \neq x$$
$$...$$

We consider first order theories (with equality) axiomatized by schemata of the above form.

We give a decision procedure for the quantifier-free theory of recursively defined data structures, which, for a conjunction of equalities and disequalities, determines its satisfiability in linear time. The procedure has possible applications in any theorem prover which handles such data structures, for instance, Boyer and Moore's prover for recursively defined functions [Boyer and Moore 1977], Guttag and Musser's prover for abstract data types [Guttag, Horowitz and Musser 1976], or the simplifier we are developing [Nelson and Oppen 1978bJ.

It follows that the quantifier-free DNF theory of recursively defined data structures (that is, the quantifier-free theory in which every formula is in disjunctive normal form) is decidable in linear time and therefore that the (full) quantifier-free theory of recursively defined data structures is in NP (and hence NP-complete).

We next consider theories in which quantification is allowed, in particular the first order theory of LISP list structure. Our basic decision procedure can be modified to form the basis for a quantifier-elimination method for this theory However, the decidability of this theory and its complexity can be derived from existing results in logic. In particular, the constructor c (CONS) may be treated as the structural-analogue of what is called in logic a "pairing function". There are results in the literature on theories of pairing functions and part of the purpose of this paper is to point out their applicability, We will use these results to show that the first order theory of list structure (recursively defined data structures) is decidable but not elementary recursive. That is, although the theory is decidable, there cannot exist a decision procedure for it which always halts in time $2^{2^{\cdot^{\cdot^{\cdot^{2^n}}}}}$ for any fixed number of 2's (n is the length of the formula).

The question of the decidability of the first order theory of list structure has recently been raised by John McCarthy [McCarthy 1978]; by the above it is decidable. McCarthy shows that if one includes the predicate SUBEXPR(X,Y), which asserts that x is a subexpression (subtree) of y, then the theory is undecidable.

If one drops the acyclicity axiom schema III, different results obtain. [Nelson and Oppen 1978aJ give a decision procedure for the quantifier-free theory of possibly-cyclic list structure which, for a conjunction of equalities and disequalities of length n, decides its satisfiability in time $O(n^2)$. A variant of this procedure has been implemented in our simplifier [Nelson and Oppen 1978b]. [Johnson and Tarjan 1977] have improved the underlying graph algorithm to run in time $O(n \log^2 n)$.

2. Decision Procedure for the Quantifier-Free Theory

2.1 Introduction

The language of the theory consists of variables, function symbols c, $s_1, \ldots, s_k$, and the predicate =. The decision procedure described in this section determines the satisfiability of a con junction of atomic literals in time linear in the length of the conjunction.

Assume we are given a conjunction. The basic strategy of the procedure is to construct a directed graph whose vertices represent the terms of the conjunction and an equivalence **relation on** the vertices of the graph representing all the equalities that are entailed by the conjunction. The procedure then checks if any asserted disequality conflicts with any of these equalities or if any of the acyclicity axioms are violated. If so, the conjunction is unsatisfiable; otherwise, it is satisfiable.

The algorithm represents terms in the conjunction by (the equivalence classes of) vertices in a directed, acyclic graph possibly with multiple edges. A vertex in the graph may have outdegree zero or outdegree k (corresponding to the k selector functions). The edges leaving a vertex are ordered. If u is a vertex, then, for $1 \leq i \leq$ outdegree(let u[i] denote the ith *successor* of u, that is, the vertex to which the ith edge of u points. Since multiple edges are allowed, possibly $u[i] = u[j]$ for $i \neq j$.

Every term in the conjunction is either an atomic symbol or an expression of one of the forms $s_i(t)$ or $c(t_1, t_2, \ldots, t_k)$ where $t, t_1, t_2, \ldots, t_k$ are terms. An atomic term x will be represented by a vertex labelled x. A term of the form $s_i(t)$ will be represented by a vertex v such that $v = u[i]$ for some vertex u representing t. (If necessary, "dummy" successors of u are added to represent the $s_j(t), j \neq i$, if these do not appear in the formula.) A term of the form $c(t_1, t_2, \ldots, t_k)$ will be represented by a node with k successors representing respectively $t_1, t_2, \ldots, t_k$. To represent the fact that two terms are equal, we will merge, that is, make equivalent the vertices that represent them.

The first step taken by the decision procedure is to construct the graph representing the terms in the conjunction. Vertices representing terms asserted equal in the conjunction are then merged. Vertices representing the same atomic symbol (that is, vertices having the same label) are also merged.

The main work of the algorithm is to close the graph under all entailed equivalences of vertices, checking as it does so that no cycles are being introduced into the graph (since such cycles would violate the acyclicity condition). First, if two vertices u and v are equivalent and both have non-zero outdegree, then the equivalence classes of their corresponding successors must be merged (since $x = y \supset s_1(x) = s_1(y) \wedge \ldots \wedge s_k(x) = s_k(y)$). Secondly, if all the corresponding successors of two vertices u and v with non-zero outdegree are equivalent, then the equivalence classes of u and v must be merged (since $s_1(x) = s_1(y) \wedge \ldots \wedge s_k(x) = s_k(y) \supset x = y$).

The following fragment of an procedure carries out the above step, but does not check for *cycles.*

```
1. For all pairs of vertices u, v with nonzero outdegree

    if u and v are equivalent
        then (if any corresponding successors of u and v are not equivalent
            then merge the corresponding successors
            also restart step 1)
        else if all the corresponding successors of u and v are equivalent
            then merge u and v
            also restart step 1.

2. Return.
```

This algorithm is‘ non-linear. In the next section we will describe a linear algorithm for computing what will be called the *bidirectional closure* of a graph and in the following section show how this graph algorithm gives a linear decision procedure.

2.2 Bidirectional Closure

Let G = (V, E) be a directed graph possibly with multiple edges such that the edges leaving each vertex are ordered. If R is an equivalence relation on the vertices of G, then G is acyclic under R if there is no sequence of vertices $u_0, u_0', u_1, u_1', \ldots u_p = u_0$ of G, $p > 0$, such that $<u_i, u_i'> \epsilon$ R and $<u_i', u_{i+1}> \epsilon$ E for $0 \le i < p$.

Let R be an equivalence relation on the vertices. of G. Define the *congruence closure* $R\uparrow$ of R on G to be the unique minimal extension of R such that 1. $R\uparrow$ is an equivalence relation and 2. any two vertices u and v with equal, nonzero outdegree are equivalent under $R\uparrow$ if all their corresponding successors are equivalent under $R\uparrow$. If G under $R\uparrow$ is acyclic, there are linear algorithms for constructing $R\uparrow$ ([Downey and Sethi 1977], [Johnson and Tarjan 1977]); these algorithms abort if G under $R\uparrow$ is not acyclic.

Let R be an equivalence relation on the vertices of G. Define the *unification* closure $R\downarrow$ of R on G to be the unique minimal. extension of R such that 1. $R\downarrow$ is an equivalence relation and 2. if any two vertices u and v with equal, nonzero outdegree are equivalent under $R\downarrow$, then all their corresponding successors are equivalent under $R\downarrow$. If G under $R\downarrow$ is acyclic, there are linear algorithms for constructing $R\downarrow$ (for instance, the linear unification algorithm of [Paterson and Wegman 1977]); this algorithm aborts if G under $R\downarrow$ is not acyclic.

We use the notation $R\uparrow$ and $R\downarrow$ to suggest the directional duality of the two notions of closure.

Let R be an equivalence relation on the vertices of G. Define the *bidirectional closure* $R\updownarrow$ of R on G to be the unique minimal extension of R such that 1. $R\updownarrow$ is an equivalence relation and 2. for any two vertices u and v with equal, nonzero outdegree, u and v are equivalent under $R\updownarrow$ if and only if all their corresponding successors are equivalent under $R\updownarrow$.

Consider now the problem of constructing the bidirectional closure. First, it is apparent that if a congruence closure algorithm and a unification closure algorithm are run alternately enough times *over* G that eventually G will be bidirectionally closed. That is, $R\downarrow\uparrow\downarrow\uparrow\ldots = R\updownarrow$. However, if G is such that the outdegree of each vertex is either 0 or k, for some fixed k, then one pass of each algorithm is sufficient, by the following lemma.

Lemma: Let $G = (V,E)$ be a directed graph with multiple edges such that the edges leaving each vertex are ordered. Assume that the outdegree of each vertex in G is either 0 or k for some fixed k. Let R be an equivalence relation on the vertices of G. Then $R\updownarrow = R\downarrow\uparrow$.

Proof:

It suffices to prove that $R\downarrow\uparrow$ is unification closed.

We first need a property of unification closed relations. Let R_1 be a unification closed relation on G. Let u and v be a pair of vertices in G with outdegree k such that $<u[i],v[i]> \in R_1$ for all $1 \le i \le k$. Then we claim that the minimal equivalence relation R_2 containing R_1 and $<u,v>$ is also unification closed. Note first that R_2 is R_1 except that the equivalence classes of u and v have been merged. Consider any pair of vertices x and y with outdegree k such that $<x,y> \in R_2$. If $<x,y> \in R_1$ then certainly $<x[i],y[i]> \in R_2$ for all $1 \le i \le k$. So suppose $<x,y>$ is not in R_1. Then $<x,u> \in R_1$ and $<y,v> \in R_1$ (or $<x,v>$ and $<y,u>$ are in R_1). It follows that, for all $1 \le i \le k$, $<x[i],u[i]> \in R_1$ and $<y[i],v[i]> \in R_1$ (since R_1 is unification closed and the outdegree of all the vertices x, y, u, v is k), and thus that $<x[i],y[i]> \in R_1$, since $<u[i],v[i]> \in R_1$ by assumption. Thus, merging u and v did not affect the unification closure property.

Therefore, starting out with $R\downarrow$ and making equivalent any two vertices with outdegree k, all of whose corresponding sons are equivalent, leaves the resulting minimal equivalence relation unification closed. By induction, it follows that $R\downarrow\uparrow$ is unification closed.

It is important for this proof that the vertices have the same outdegree if they have nonzero outdegree. Otherwise, in the above proof it is not necessarily the case that if $<x,u> \in R_1$ then all their corresponding successors are equivalent.

The order of the passes is also important; $R\uparrow\downarrow$ is not necessarily equal to $R\updownarrow$.

If G under $R\updownarrow$ is acyclic, there is therefore a linear algorithm for constructing $R\updownarrow$. One first constructs $R\downarrow$ using a linear unification closure algorithm and then closes $R\downarrow$ under congruences (that is, constructs $R\downarrow\uparrow$) using a linear congruence closure algorithm. If G under $R\updownarrow$ is not acyclic, one of these algorithms will abort.

2.3 The Decision Procedure

We will now state more precisely the decision procedure described informally in Section 2.1. We start by describing the data structures manipulated by the procedure.

First, corresponding to every term t in a formula, there is a directed, acyclic graph G(t). G(t) will contain a vertex $V_{G(t)}(t)$ "representing" t.

1. If t is an atomic symbol, G(t) has a single vertex with zero outdegree labelled with t. $V_{G(t)}(t)$ will be this vertex.

2. If t is of the form $s_i(\alpha)$, then G(t) will be G(a) and $V_{G(t)}(\beta)$ will be $V_{G(\alpha)}(\beta)$ for all subexpressions β of a. However, if $V_{G(\alpha)}(a)$ has outdegree 0, we will add k successors to $V_{G(t)}(\alpha)$ (each successor will be a new unlabelled vertex with outdegree zero). In either case, $V_{G(t)}(t)$ will be the ith successor of $V_{G(t)}(a)$.

3. If t is of the form $c(\alpha_1, \ldots.$ a,), then G(t) is the disjoint union of $G(\alpha_1), \ldots. G(\alpha_k)$ together with a new vertex u with k successors. For all $1 \le i \le k$, u[i] is $V_{G(\alpha_i)}(\alpha_i)$. $V_{G(t)}(t)$ is u. (In taking the disjoint union, we will always assume that the label of any vertex in the union is its old label in the graphs whose union we are taking. Similarly, for any term β, if $V_{G(\alpha_i)}(\beta)$ exists in $G(\alpha_i)$, then $V_{G(t)}(\beta)$ will be the same vertex.)

Notice that the only labelled vertices are those representing atomic terms, and that all vertices either have outdegree 0 or outdegree k.

In what follows, we may refer to V(t) instead of $V_{G(t)}(t)$.

Decision Procedure

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

$$v_1 = w_1 \wedge \ldots, \wedge v_r = w_r \wedge$$
$$x_1 \neq y_1 \wedge \ldots . \wedge x_s \neq y_s$$

1. Construct G, the disjoint union of $G(v_1), \ldots G(v_r), G(w_1), \ldots G(w_r), G(x_1), \ldots G(x_s), G(y_1), \ldots G(y_s)$. Let R be $\{(V(v_i), V(w_i)) \mid 1 \leq i \leq r\} \cup \{(a, \beta) \mid a$ and β are vertices in G with the same label$\}$. That is, the initial equivalence relation R makes equivalent vertices representing terms asserted equal in F and vertices representing the same atomic term in F.

2. Construct $R\updownarrow$, the bidirectional closure of R on G. Let $[\![u]\!]$ denote the equivalence class of vertex u in G under $R\updownarrow$. If G under $R\updownarrow$ is not acyclic, return UNSATISFIABLE.

3. For i from 1 to s, if $[\![V(x_i)]\!] = [\![V(y_i)]\!]$, return UNSATISFIABLE. Otherwise, return SATISFIABLE.

2.4 Correctness of the Decision Procedure

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 R_0 be the partition of the vertices of G corresponding to the final equivalence relation $R\updownarrow$. We define k functions $s_{10}, \ldots s_{k0}$ from a subset of R_0 to R_0, and a function c_0 from a subset of R_0^k to R_0. For $1 \leq i \leq k$, an equivalence class Q is in the domain of s_{i0} if Q contains a vertex u with outdegree k; in this case, $s_{i0}(Q) = [\![u[i]]\!]$. (Since every vertex in G has outdegree either 0 or k, Q is in the domain of a particular s_{i0} if and only if it is in the domain of s_{i0} for all $1 \leq i \leq k$.) A k-tuple $(Q_1, \ldots Q_k)$ of equivalence classes is in the domain of c_0 if there exists a vertex u with outdegree k such that $u[i] \in Q_i$ for $1 \leq i \leq k$; in this case, $c_0(Q_1, \ldots Q_k) = [\![u]\!]$. Note that $c_0, s_{10}, \ldots s_{k0}$ are well-defined, since G is bidirectionally closed. and every vertex in G has outdegree either 0 or k. However, these functions are not necessarily defined over the whole of R_0^k and R_0. To construct an interpretation, we must extend these functions; in the process we will construct an infinite domain for the interpretation. We now describe this construction.

Let $G_0 = G$. Construct as above the tuple $(G_0, R_0, c_0, s_{10}, \ldots s_{k0})$. Suppose we have constructed the first $j+1$ tuples $(G_0, R_0, c_0, s_{10}, \ldots s_{k0}), \ldots (G_j, R_j, c_j, s_{1j}, \ldots, s_{kj}), \ldots$ Construct $(G_{j+1}, R_{j+1}, c_{j+1}, s_{1j+1}, \ldots, s_{kj+1})$ to be the following extension of $(G_j, R_j, c_j, s_{1j}, \ldots s_{kj})$:

1. For each equivalence class Q of R_j which is not in the domain of any s_{ij}, choose any vertex u in Q (u therefore has outdegree 0 in G_j). In G_{j+1}, add k new vertices as successors to u, each in an equivalence class of its own in R_{j+1}. Let $c_{j+1}([\![u[1]]\!], \ldots, [\![u[k]]\!]) = Q$ and $s_{ij+1}(Q) = [\![u[i]]\!]$, for $1 \leq i \leq k$. By this construction, the domain of s_{ij+1} is R_j.

2. For each tuple $(Q_1, \ldots Q_k)$ of equivalence classes of R_j not in the domain of c_j, add a new vertex u to G_{j+1} in an equivalence class of its own in R_{j+1}. Let u have outdegree k and, for $1 \leq i \leq$

k, let $u[i] = v$ for some v in Q_i. (Since $(Q_1, \ldots Q_k)$ is not in the domain of c_j, there is no other vertex w in G_{j+1} with outdegree k such that $w[i] \in Q_i$, for $1 \le i \le k$.) Let $c_{j+1}(Q_1, \ldots, Q_k) = [\![u]\!]$, and, for $1 \le i \le k$, let $s_{ij+1}([\![u]\!]) = Q_i$. By this construction, the domain of c_{j+1} is R_j^k.

G_{j+1} is thus G_j except for the new vertices $u_1, \ldots u_p$ added in steps 1 and 2 above. R_{j+1} is R_j together with the additional singleton equivalence classes $[\![u_1]\!], \ldots [\![u_p]\!]$. $c_{j+1}, s_{1j+1}, \ldots, s_{kj+1}$ are c_j, $s_{1j}, \ldots, s_{kj}$ extended as described in steps 1 and 2 above. The extensions are well-defined, Notice in particular that if any $s_{ij+1}(Q)$ is defined, then all the $s_{ij+1}(Q)$ are defined.

Lemma: Suppose $Q, Q_1, \ldots Q_k$ are equivalence classes of R_j. Then the following hold:

1. If Q is in the domain of s_{ij}, for $1 \le i \le k$, then $(s_{1j}(Q), \ldots s_{kj}(Q))$ is in the domain of c_j and $c_j(s_{1j}(Q), \ldots, s_{kj}(Q)) = Q$.

2. If $(Q_1, \ldots Q_k)$ is in the domain of c_j, then $c_j(Q_1, \ldots Q_k)$ is in the domain of s_{ij}, and $s_{ij}(c_j(Q_1, \ldots Q_k)) = Q_i$, for $1 \le i \le k$.

3. G_j under R_j is acyclic.

Proof:

Base step: $j = 0$. If Q is in the domain of the s_{i0}, then there exists a vertex u in G_0 with outdegree k. Therefore $(s_{10}(Q), \ldots s_{k0}(Q))$ is in the domain of c_0 and $c_0(s_{10}(Q), \ldots s_{k0}(Q)) = Q$. So the first clause of the lemma holds. If $(Q_1, \ldots Q_k)$ is in the domain of c_0, then there is a vertex u with outdegree k such that $c_0(Q_1, \ldots Q_k) = [\![u]\!]$, and, for $1 \le i \le k$, $u[i] \in Q_i$. Therefore, for $1 \le i \le k$, $c_0(Q_1, \ldots, Q_k)$ is in the domain of s_{i0}, and $s_{i0}(c_0(Q_1, \ldots Q_k)) = [\![u[i]]\!] = Q_i$. So the second clause of the lemma holds. Since G_0 under R_0 is acyclic, the third clause holds.

Suppose the lemma holds for j; we show it also holds for j t 1.

Proof of clause 1. If Q is in the domain of one (and hence all) of $s_{1j}, \ldots s_{kj}$, then the result follows from the induction hypothesis and the fact that $(G_{j+1}, R_{j+1}, c_{j+1}, s_{1j+1}, \ldots s_{kj+1})$ extends $(G_j, R_j, c_j, s_{1j}, \ldots, s_{kj})$. If Q is not in the domain of the s_{ij}, then, in constructing $(G_{j+1}, R_{j+1}, c_{j+1}, s_{1j+1}, \ldots, s_{kj+1})$, we added k vertices as successors to some vertex u in Q and defined $c_{j+1}(s_{1j+1}(Q), \ldots s_{kj+1}(Q)) = c_{j+1}([\![u[1]]\!], \ldots, [\![u[k]]\!]) = Q$.

Proof of clause 2. If $(Q_1, \ldots Q_k)$ is in the domain of c_j, then the result follows from the induction hypothesis and the fact that $(G_{j+1}, R_{j+1}, c_{j+1}, s_{1j+1}, \ldots s_{kj+1})$ extends $(G_j, R_j, c_j, s_{1j}, \ldots, s_{kj})$. Otherwise, in constructing $(G_{j+1}, R_{j+1}, c_{j+1}, s_{1j+1}, \ldots s_{kj+1})$, we added a vertex u such that $[\![u]\!] = c_{j+1}(Q_1, \ldots Q_k))$, and $u[i] \in Q_i$ for $1 \le i \le k$. $c_{j+1}(Q_1, \ldots Q_k))$ is thus in the domain of s_{ij+1} and

$s_{ij+1}(c_{j+1}(Q_1, ..., Q_k)) = Q_i$ for $1 \le i \le k$. The second clause therefore holds.

The third clause holds from the construction.

Let R' be the union of the R_i. Let $s_i'(Q)$ be $s_{ij}(Q)$ for the first j such that $s_{ij}(Q)$ is defined. Let c' be defiried similarly. It follows that $c', s_1', \ldots s_k'$ satisfy the axioms and are defined on all of R_k'.

We will now define an interpretation ψ which satisfies F. ψ interprets c, $s_1, \ldots s_k$ as c', $s_1', \ldots s_k'$. It follows that this interpretation satisfies the axioms. It remains to show that ψ satisfies F. It is straightforward to show that for every term t in the formula, $\psi(t) = [\![V(t)]\!]$. But $V(t_i)$ and $V(w_i)$ have been merged, for $1 \le i \le r$, so ψ satisfies the equalities in F. $V(x_i)$ and $V(y_i)$ are in different equivalence classes since Step 3 returned SATISFIABLE, so ψ satisfies the disequalities in F.

2.5 Linearity of the Decision Procedure

G can be constructed in several ways, but some care must be taken if it is to be constructed in linear time, that is, in time O(n) where n is the length of the formula F. We describe one way of doing so.

Step 1. For each term t in the formula, we construct G(t). We do not bother to identify common subexpressions; distinct occurrences of similar subterms of t will be represented by distinct vertices in G(t). However, we keep a list of pairs <t,V(t)> for each term v_i, w_i, x_i and y_i in the formula. We also keep a list of pairs <a, V(a)> for each occurrence of each atomic symbol a in the formula. We then form G, the disjoint union of these graphs. The number of vertices and edges in G is O(n) and the time required to construct G is also O(n).

Step 2. We next add to the graph the equalities asserted in the formula by merging vertices $V(v_i)$ and $V(w_i)$ for each equality $v_i = w_i$ in the formula. Since in Step 1 we kept track of each $V(v_i)$ and $V(w_i)$, we can do Step 2 in time O(n).

Step 3. We now make equivalent all vertices with the same label. Each such vertex represents an atomic symbol in the original formula and so appears in the list of pairs <a, V(a)> constructed in Step 1. Under a reasonable model, we can sort this list on the first argument of each pair <a,V(a)> in time O(n) using lexicographic sorting. We then scan through this list; for each pair of adjacent elements $<a_1, V(a_1)>$ and $<a_2, V(a_2)>$ in this list, if $a_1 = a_2$, then we make equivalent $V(a_1)$ and $V(a_2)$. This step again takes time O(n).

(In practice, this elaborate method would not be used. Instead, we would use a hash table to store V(a) for each a, and would never create two vertices with the same label. Languages such as LISP support this very efficiently.)

Step 4. Finally we construct G, the bidirectional closure of the relation on G_0 constructed in the previous steps. Again we can do this in linear time, as shown in Section 2.2. Notice that in constructing the bidirectional closure, we will automatically identify (make equivalent) all common subexpressions.

3. The First-order Theory

For concreteness, we will consider the first order theory of list structure (with function symbols CONS, CAR and CDR and predicate symbols = and ATOM).

First, the decision procedure given in the previous section for quantifier-free conjunctions can be modified to be the basis for a quantifier-elimination method for this theory, However, it is more interesting to derive the decidability and complexity of this theory from existing results in logic on theories of *pairing functions.*

A *pairing function* on a set S is a one-one map $J : S \times S \rightarrow S$. An example of a pairing function over the natural numbers is the function $J(x,y) = 2^x 3^y$.

Associated with each pairing function J are its projection functions K and L. These are partial functions $S \rightarrow S$ satisfying $K(J(x,y)) = x$ and $L(J(x,y)) = y$. Since K and L are partial, we will formally consider all functions as relations but will continue to write, for instance, $K(z) = x$ instead of $K(z,x)$. (An alternative would be to make all functions total by introducing $\perp$, the undefined element, in to the logic.)

K and L satisfy the axioms

1. $\forall x \; \forall y \; \exists! z \, [K(z) = x \wedge L(z) = y]$

2. $\forall z \, [\exists x \, (K(z) = x \vee L(z) = x) \supset \exists! x \, \exists! y \, (K(z) = x \wedge L(z) = y)]$

. The pairing function J is defined in terms of K and L by $J(x,y) = z \equiv K(z) = x \wedge L(z) = y$.

The first order theory of pairing functions (the first order theory with these axioms) is undecidable (unpublished results by Hanf, Scott, and Morley). However, with appropriate additional axioms, the theory is decidable. These additional restrictions on K and L correspond to the acyclicity condition we put on our recursively defined data structures together with the decidability of the theory of atoms.

First, we partition the set S into two disjoint parts, the set A of *atom* and the set S - A of non-atoms. ATOM(x) holds if and only if x is an atom.

The following infinite axiom schema requires that the pairing function be acyclic on all non-atoms.

3. (Acyclicity)

$\forall z [\neg \mathrm{ATOM}(z) \wedge \exists x (K(z) = x) \supset K(z) \neq z]$

$\forall z [\neg \mathrm{ATOM}(z) \wedge \exists x (L(z) = x) \supset L(z) \neq z]$

$\forall z [\neg \mathrm{ATOM}(z) \wedge \exists x (K(L(z)) = x) \supset K(L(z)) \neq z]$

...

Next, if z is not an atom, it must have projections.

4. $\forall z [\neg \mathrm{ATOM}(z) \supset \exists x (K(z) = x)]$

$\forall z [\neg \mathrm{ATOM}(z) \supset \exists x (L(z) = x)]$

Finally, once an element z lies in A, all iterations of projection functions from z (as long as they are defined) must lie in A.

5. $\forall z [\mathrm{ATOM}(z) \wedge \exists x (K(z) = x) \supset \mathrm{ATOM}(K(z)) \wedge \mathrm{ATOM}(L(z))]$

A pairing function satisfying these axioms is defined to be *acyclic except for* A.

If A is empty, the first order theory with the above as axioms is decidable ([Mal'cev 1961, 1962]). If A is non-empty, the theory may or may not be decidable: [Tenney 1972, 1977] reduced the question of decidability to the decidability of the theory restricted to the atoms; if the latter is decidable then so is the former. It is the latter result that we now use.

Consider the first-order theory of list structure. CONS is the pairing function J, CAR is the left projection K, CDR is the right projection L, S is the set of s-expressions, and A is the set of atoms. By the above, the first order theory of list structure is decidable if the theory of atoms under CAR, CDR and = is decidable.

There are many possible choices for A and its associated theory. First, A might be infinite (as in LISP) or consist of the single atom NIL (as in Boyer and Moore's original prover). Secondly, CAR and CDR may or may not be defined on all or some of the atoms. If defined, CAR and CDR may be cyclic or acyclic (for instance, we might choose CAR(NIL) and CDR(NIL) to be NIL as in MACLISP). Regardless of the choice, as long as the theory of atoms is decidable, so is the overlying theory of list structure. For a reasonable choice of the theory of atoms, its decidability is apparent.

Therefore, for any "reasonable" axiomatization of the theory of LISP list structure, its first order theory is decidable. Unfortunately, an efficient decision procedure for the theory cannot exist.

[Rackoff 1975] has shown that no theory of pairing functions admits an elementary recursive decision procedure, that is, one which always halts in time $2^{2^{\cdot^{\cdot^{2^n}}}}$ for any fixed number of 2's (n is the length of the formula). It follows that any decision procedure for the theory of list structure must be very inefficient in the worst case.

Although Tenney proved his result for pairing functions $S \times S \to S$, his argument holds as well for k-ary pairing functions, that is pairing functions $S^k \to S$ which satisfy the obvious generalization of the above axioms. Similarly, Rackoff proves that his lower bound also applies to any k-ary pairing function. It follows that, given a recursive data structure with constructor c and selectors $s_1, \ldots, s_k$ satisfying the obvious generalization of the above axioms, the associated first order theory is decidable but not elementary recursive.

Acknowledgments

I am indebted to Greg Nelson, Dave Stevenson and Bob Tarjan for numerous helpful discussions.

References

[Boyer and Moore 1977] R. Boyer and J Moore, "A Lemma Driven Automatic Theorem Prover for Recursive Function Theory", Proceedings of the Fifth IJCAI, 1977.

[Downey and Sethi 1977] P. Downey and R. Sethi, "Finding Common Subexpressions", submitted for publication,

[Guttag, Horowitz, Musser 19761 J. Guttag, E. Horowitz and D. Musser, "Abstract Data Types and Software Validation", Technical Report ISI/RR-76-48, Information Sciences Institute, University of Southern California, August 1976, to appear CACM.

[Hoare 1975] C. A. R. Hoare, "Recursive Data Structures", International Journal of Computer and Information Sciences, June 1975.

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

[Mal'cev 1961] A. Mal'cev, "On the Elementary Theories of Locally Free Universal Algebras", Soviet Mathematics - Doklady, 1961.

[Mal'cev 1962] A. Mal'cev, "Axiomatizable Classes of Certain Types of Locally Free Algebras", Sibirskii Matematicheskii Zhurrial, 1962. ,

[McCarthy 1978] J. McCarthy, "Representation of Recursive Programs in First Order Logic", to be presented at International Conference on Mathematical Studies of Information Processing, Kyoto, Japan.

[Nelson and Oppen 1978a] C. G. Nelson and D. C. Oppen, "Fast Decision Procedures based on Congruence Closure", AI Memo AIM309, CS Report No. STAN-CS-77-646, Stanford University,

[Nelson and Oppen 1978b] C. G. Nelson and D. C. Oppen, '*Simplification by Cooperating Decision Procedures", Proceedings of the Fifth ACM Symposium on Principles of Programming Languages, 1978 (also Stanford CS report AIM 311).

[Paterson and Wegman 1977] M. Paterson and M. Wegman, "Linear Unification", to appear JCSS.

[Rackoff 19751 C. Rackoff, "The Computational Complexity of some Logical Theories", Ph. D. thesis, M. I. T., 1975.

[Tenney 1972] R. Tenney, "Decidable Pairing Functions", Ph. D. thesis, Cornell University, 1972.

[Tenney 1977] R. Tenney, "Decidable Pairing Functions", submitted for publication.