COMPUTER SCIENCES DEPARTMENT
University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

A CORRECTNESS PROOF OF THE FISHER-GALLER ALGORITHM USING INDUCTIVE ASSERTIONS

by

Ralph L. London

Technical Report #102

October 1970

A CORRECTNESS PROOF OF THE FISHER-GALLER ALGORITHM USING INDUCTIVE ASSERTIONS*

by

Ralph L. London

INTRODUCTION

In Morris (1970b), J. H. Morris uses truncation induction (Morris 1970a) to give a correctness proof for the Fisher-Galler algorithm of recording equivalence relations (Knuth 1968, pp. 353-355). When Morris presented his proof at the Symposium, he remarked that he had been unable, after a brief attempt, to give a proof by inductive assertions. Accordingly, the purpose of this note is to record a proof for the algorithm using inductive assertions. The proof will follow the style of London (1970b) and, in addition, will use backward substitution, briefly described in London (1970a). Some comments about Morris' and my proof are also given at the end.

THE AIM OF THE ALGORITHM

The Fisher-Galler algorithm "is to keep track of an arbitrary equivalence relation over a finite set of integers [1,k]. We must

*This work is supported by the National Science Foundation under Grant GJ-583.

provide two subroutines, Enter and Equiv. After initialization Enter may be called with pairs of integers in the range [1,k] and is expected to record the fact that they are equivalent. Equiv(x,y) should return true if x and y are equivalent -- either because Enter(x,y) has been executed previously or because the laws of reflexivity, symmetry, or transitivity make them so, given other explicit equivalences. Otherwise Equiv returns false." (Morris 1970b, Section B.)

The Fisher-Galler algorithm starts with an array A[1:k] set to all zeros. A subroutine Rep(x) is defined (see below), intended to return a canonical Representative of x's equivalence class. Initially, Rep(x) = x for $1 \leq x \leq k$. The value of Equiv(x,y) is true if Rep(x) = Rep(y) and false otherwise. The subroutine Enter, which uses Rep twice, is defined below. If $\equiv$ denotes the equivalence relation, the correctness statement is: After each call of Enter(x,y), then $z \equiv w$ iff Equiv(z,w).

Informally (and not part of the proof), the array A is viewed as a forest of trees. Rep(x) is, in fact, the root of the (unique) tree containing x. Initially the array consists solely of roots (hence Rep(x) = x). Enter(x,y) makes x and y equivalent by making x's tree into a subtree of the root of y's tree.

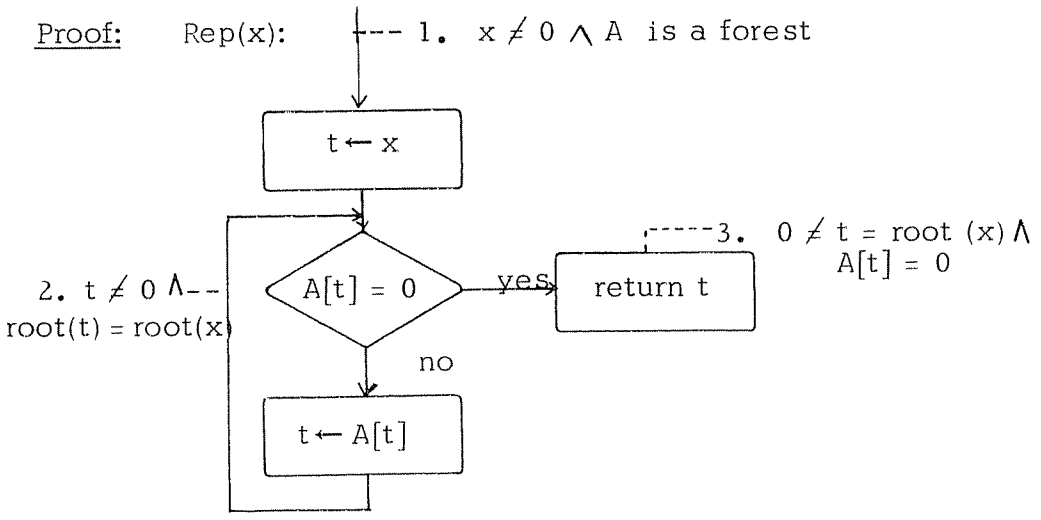
THE RIGOROUS PROOF OF CORRECTNESS

All inputs are assumed to be positive integers and the array A is initially zeros. It is easy to verify, therefore, that all quantities manipulated by the subroutines are non-negative integers. Part of this verification is omitted.

Lemma 1: The integer-valued subroutine Rep(x) is defined by the following flowchart. If $x \neq 0$ and A is a forest on entry to Rep(x) and if (A1) and (A2) below hold for root (x), then on exit the returned value t of Rep(x) satisfies

$$0 \neq t = \text{root}(x) \wedge A[t] = 0 .$$

Proof:



First note a global property of Rep(x), namely A is unaltered by Rep(x) since only t is changed. Second we assume for a forest A that if $x \neq 0$, then

$$\text{root}(x) = \text{root}(A[x]) \qquad \text{if} \quad A[x] \neq 0 \qquad \text{(A1)}$$

$$\text{root}(x) = x \qquad \text{if} \quad A[x] = 0\,. \qquad \text{(A2)}$$

(This property will be shown to hold initially as part of the proof of Theorem 1. The property will be shown to be preserved as part of the proof of lemma 2 for Enter, the only place where the array A is changed.)

The assertions are verified by backward substitution:

Path 1-3: verification condition (vc):

$x \neq 0 \wedge A$ is a forest $\wedge\ A[x] = 0 \supset 0 \neq x = \text{root}(x) \wedge A[x] = 0$.

True by (A2) and identities.

Path 2-3: vc: $t \neq 0 \wedge \text{root}(t) = \text{root}(x) \wedge A[t] = 0 \supset 0 \neq t = \text{root}(x) \wedge A[t] = 0$.

Use (A2) to obtain $t = \text{root}(t)$.

Path 1-2: vc: $x \neq 0 \wedge A$ is a forest $\wedge\ A[x] \neq 0 \supset A[x] \neq 0 \wedge \text{root}(A[x]) = \text{root}(x)$.

True by (A1).

Path 2-2: vc: $t \neq 0 \wedge \text{root}(t) = \text{root}(x) \wedge A[t] \neq 0 \supset A[t] \neq 0 \wedge \text{root}(A[t]) = \text{root}(x)$.

Use (A1) to obtain $\text{root}(t) = \text{root}(A[t])$.

Since the array is defined only for A[1:k], one should show all array references are in bounds. Since $t \neq 0$ and t is a non-negative integer, one need only show $t \leq k$. If we augment assertion 1 (i.e. the assumptions of lemma 1) with "$x \leq k \wedge A[i] \leq k$ for $1 \leq i \leq k$" and augment both assertions 2 and 3 (i.e. the conclusion of lemma 1) with "$t \leq k$," these additional assertions are easily verified by

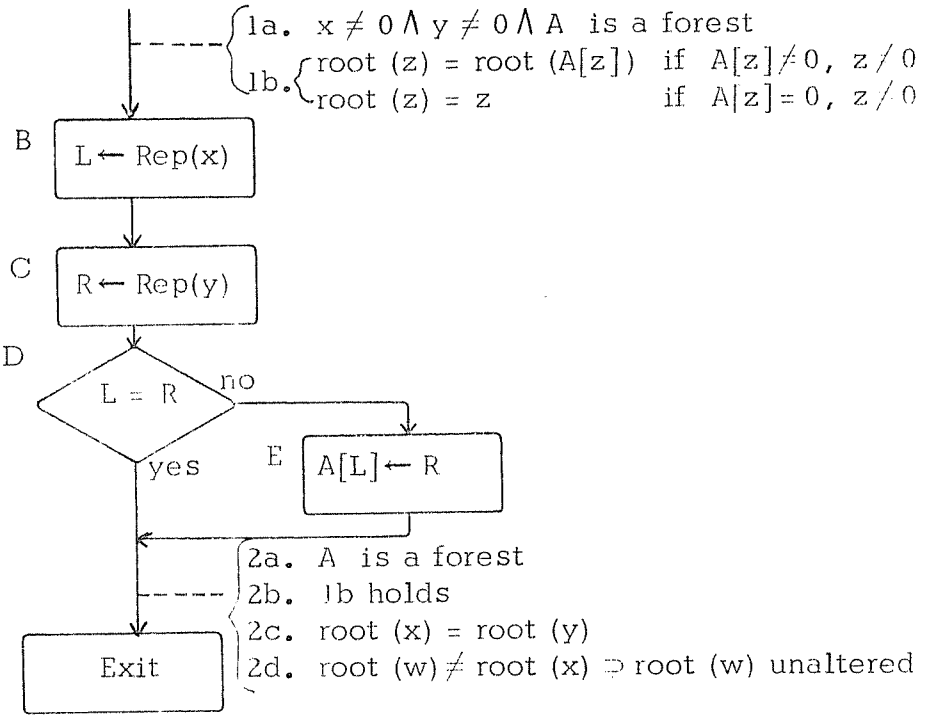
backward substitution using $A[i] \leq k$ for $1 \leq i \leq k$; the latter holds everywhere in Rep(x) since A is unaltered by Rep(x). ∎

One reason Morris said he failed using inductive assertions was that he wrote A[A[t]]. I avoided this by using (A1) and (A2). Going backward is not important since the proof for Rep(x) is easily done by working forward.

Lemma 2: The subroutine Enter(x,y) is defined by the following flowchart. If assertions 1a and 1b on the flowchart hold on entry, then assertions 2a - 2d hold at the exit.

Proof:

Enter(x,y)



The assertions 2a-2d are verified as follows: First note that 1a and 1b show the assumptions for calling Rep(x) and Rep(y) are satisfied. Hence at box D (using lemma 1) $L \neq 0 \bigwedge L = \text{root}(x) \bigwedge A[L] = 0$ and $R \neq 0 \bigwedge R = \text{root}(y) \bigwedge A[R] = 0$.

If $L = R$, then 2c holds. There is no change to the forest A so 2a, 2b, and 2d hold. Assume $L \neq R$. Then $L \neq 0$ implies $A[L]$ is defined; after box E, $A[L] = R \neq 0$. A is still a forest since the only change to A is at $A[L]$, now set to R. That is, the structure whose root is R is still a tree since the tree whose root was L ($\neq R$) is now a new son tree of R. In symbols $\text{root}(L) = \text{root}(R) = \text{root}(A[L])$ with $A[L] \neq 0$ and $L \neq 0$. But $\text{root}(R) = R$ since $A[R] = 0$; hence 1b still holds at the exit. Moreover, 2d holds since only root(x) is changed at box E.

It remains to show 2c, root(x) = root(y):

root(x) = root(L)		by definition of L and the change to A at box E,
	= R	by above (in showing 1b holds at exit),
	= root(y)	by definition of R.

To show that all array references remain in bounds after Enter, one can augment assertion 1a with "$x \leq k \bigwedge y \leq k \bigwedge A[i] \leq k$ for $1 \leq i \leq k$" and assertion 2a with "$A[i] \leq k$ for $1 \leq i \leq k$." The

new 2a holds by 1a and box E since $R \leq k$ by the augmented assertion 3 of Rep. Finally, note that the new 1a is precisely the augmentation to Rep's assertion 1, the assumption for calling Rep. ∎

We are now ready to prove the statement of correctness.

Theorem 1: Let $\equiv$ denote the equivalence relation. Let z and w be integers, $1 \leq z, w \leq k$. Then each call of Enter(x,y) preserves the relation

$$z \equiv w \quad \text{iff} \quad \text{Equiv}(z,w).$$

(I.e. this relation could appear as both assertion 1c and assertion 2e on the flowchart for Enter(x,y).)

Proof: Since Equiv(z,w) iff Rep(z) = Rep(w) by the definition of Equiv and since Rep(z) = root(z) by lemma 1, the relation to prove (preserve) is

$$z \equiv w \quad \text{iff} \quad \text{root}(z) = \text{root}(w).$$

It is first necessary to show that Lemma 2's assumptions, 1a and 1b, for calling Enter(x,y) are satisfied. By assumption $x \neq 0$ and $y \neq 0$. Prior to the first call to Enter, we have by the array initialization, A[z] = 0, i.e. root(z) = z for all z, so A is initially a forest (of all roots). Hence both 1a and 1b hold prior

to the first call. For subsequent calls to Enter, the assertions 2a and 2b guarantee 1a and 1b since A is changed only in Enter. (The augmented assertion 1a, $x \leq k \wedge y \leq k \wedge A[i] \leq k$ for $1 \leq i \leq k$, for showing valid array references is satisfied for similar reasons: by assumption on x and y, and first by the initialization $A[i] = 0$ and subsequently by the augmented 2a, $A[i] \leq k$ for $1 \leq i \leq k$.)

The proof proceeds by cases prior to the call of Enter(x,y) which by lemma 2, assertion 2d, changes only root(x).

1. $x \equiv y$. Thus root(x) = root(y) and Enter(x,y) makes no change to the forest A.
2. $x \not\equiv y$. Let s be an integer, $1 \leq s \leq k$, and consider which equivalences and roots are changed.
   a) $s \equiv x$, i.e. root(s) = root(x). After Enter(x,y) we have root(x) = root(y) by 2c. Then $s \equiv x$ and $x \equiv y$ means $s \equiv y$ iff root(s) = root(x) = root(y). For this case these are the only changes to $z \equiv w$ iff root(z) = root(w).
   b) $s \equiv y$, i.e. root(s) = root(y). After Enter(x,y) we have root(x) = root(y). Then $s \equiv y$ and $x \equiv y$ means $s \equiv x$ iff root(s) = root(y) = root(x).

c. $s \not\equiv x$, $s \not\equiv y$, i.e. $root(s) \neq root(x)$, $root(s) \neq root(y)$. Note that s, x, and y are all different else either $s \equiv x$, $s \equiv y$, or $x \equiv y$. After Enter(x,y), we have $root(x) = root(y)$. Then $s \not\equiv x \equiv y \not\equiv s$ iff $root(x) = root(y) \neq root(s)$.

Note that $z \equiv w$ iff $root(z) = root(w)$ holds prior to the first call of Enter(x,y) since at that time $z \equiv w$ iff $z = w$ iff $root(z) = root(w)$ by $root(z) = z$ for all z initially. ∎

Theorem 2: Enter(x,y) and Equiv(x,y) each terminate.

Proof: Clear provided Rep(x) and Rep(y) both terminate. This follows in one of three ways: First, note that A is always a (finite) forest and hence $A[t] = 0$ eventually. Or second, note that only roots are altered by Enter(x,y), and then only to another root; specifically root(x) is changed to root(y) which, by box D of Enter(x,y), is different from root(x) and hence no cycles in the forest are generated. In other words, within each call to Rep(x), the variable t does not assume the same value twice; hence the loop in Rep(x) is executed at most k times. Or third, employ an induction proof based on the number of non-redundant calls to Enter(x,y). ∎

DISCUSSION

It seems appropriate to comment on the discussion (section G) in Morris (1970b). His proof and my proof are not fundamentally different. A rough indication of where his lemmas appear in my proof is:

Morris	London
Lemma 1	Lemma 1
Lemma 2	Theorem 2
Lemma 3 (keystone)	Lemma 2 (keystone?)
Lemma 4	Apply lemma 1 and (A2) to show root (root (x)) = root (x), i.e. Equiv (root (x), x)
Lemmas 5,6,7	Theorem 1

Morris calls his lemma 3 the "keystone" to his proof. Perhaps my lemma 2 is also a keystone.

My proof appears to avoid induction; however, the use of inductive assertions allows my inductions to appear only implicitly. Note, though, that there is an easy correspondence between his explicit inductions and my uses of assertions. Indeed, my proof of theorem 2 is almost explicit induction.

The notation in my assertions corresponds vaguely to his introduced functions. He notes, "Throughout [his] proof [he] was

able to think of expressions ... in terms of the objects they denote rather than something to be evaluated by an interpreter." In my proof my notation similarly allowed me to think of the effect or "evaluation" of a subroutine on the data (the forest) "rather than something to be evaluated." That my proof is not as "algebraic" [his term] as his is reduces the elegance and formal checkability of my proof, but I claim it does not reduce the rigor.

I think Morris is overly pessimistic about our ability to show objects are unaltered between subroutine calls. The use of inductive assertions can do this and has. In London (1970b), the proof involves a subroutine siftup which is called in the main program. The proof for the subroutine shows that only certain parts of the array being sorted are changed. In other words, there are no side effects (beyond the statement of what siftup does). The last example in London (1970c) involves showing the absence of side effects ("nothing else is changed"). The same methods can show no change over larger units as demonstrated, for example, in London (1968). Thus when Morris asks for "ways of combining proofs about sub-programs into proofs about larger programs without having to reprove everything," I note that this has been done in a few cases, albeit not routinely or mechanically.

Finally, I certainly agree with a point of view Morris expressed in the context of the design of programming languages. He added the

emphasized words to "the traditional question, 'How easy is it to write and prove correct a program to do X?' "

REFERENCES

Knuth, D. E. (1968). The Art of Computer Programming, Vol. 1 -- Fundamental Algorithms, Addison-Wesley, Reading, Mass.

London, R. L. (1968). Correctness of the Algol Procedure Askforhand, Computer Sciences Technical Report No. 50, University of Wisconsin.

London, R. L. (1970a). Experience with inductive assertions for proving programs correct, Symposium on the Semantics of Algorithmic Languages, E. Engeler (Ed.), Springer-Verlag Lecture Notes Series (to appear). Also Computer Sciences Technical Report No. 92, University of Wisconsin.

London, R. L. (1970b). Proof of algorithms: A new kind of certification, Comm. ACM, Vol. 13, June, 1970, 371-373.

London, R. L. (1970c). Proving programs correct: Some techniques and examples, BIT, Vol. 10, No. 2, 168-182.

Morris, J. H. (1970a). Another recursion induction principle, Computer Center Technical Report No. 38, University of California, Berkeley.

Morris, J. H. (1970b). A correctness proof using recursively defined functions, Proceedings of Symposium on Formal Semantics of Programming Languages, R. Rustin (Ed.), held at Courant Institute, New York University, (to appear). Also Computer Center Technical Report No. 39, University of California, Berkeley.