ESD-TR-72-309

A SPACE-EFFICIENT LIST STRUCTURE TRACING ALGORITHM

Ben Wegbreit

June 1972

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

Sponsored by: Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

ARPA Order 952

(Prepared under Contract No. F19628-71-C-0174 by Harvard University, Cambridge, Massachusetts.)

LEGAL NOTICE

When U. S. Government drawings, specifications or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

OTHER NOTICES

Do not return this copy. Retain or destroy

ESD-TR-72-309

A SPACE-EFFICIENT LIST STRUCTURE TRACING ALGORITHM

Ben Wegbreit

June 1972

DEPUTY FOR COMMAND AND MANAGEMENT SYSTEMS
HQ ELECTRONIC SYSTEMS DIVISION (AFSC)
L. G. Hanscom Field, Bedford, Massachusetts 01730

Sponsored by: Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, Virginia 22209

ARPA Order 952

(Prepared under Contract No. F19628-71-C-0174 by Harvard University, Cambridge, Massachusetts.)

FOREWORD

This report was prepared in support of ARPA Order No. 952 by Harvard University, Cambridge, Massachusetts under Contract F19628-71-C-0174, monitored by Capt T. J. Rosenberger, MCIT, and was submitted September 1972.

This technical report has been reviewed and is approved.

MELVIN B. EMMONS, Colonel, USAF
Director, Information Systems Technology
Deputy for Command & Management Systems

ABSTRACT

This note presents an algorithm for tracing during garbage collection of list structure. It requires only one bit for each level of doubly branching structure traced. Compared to existing trace algorithms, it generally requires less storage -- often, substantially less.

A SPACE-EFFICIENT TRACE ALGORITHM

Introduction

This note treats the problem of minimizing the temporary storage required by the trace phase of a garbage collector for list structure. We are concerned here with list structure such as that used in LISP [1] in which the nodes have two fields. These fields, called car and cdr, may point either to another node or to an atom.

Garbage collection entails tracing through all structure in use and marking each node encountered. Subsequent to tracing, a linear sweep over all nodes collects those which are unmarked and puts them onto a free list. There are basically two existing techniques for tracing through list structure. Neither is entirely satisfactory.

The first [1] uses a stack of pointers to those nodes whose tracing has been postponed. When tracing the car pointer of a node, the cdr pointer is stacked if it is non-atomic. Whenever an atom or marked node is encountered, the top stack pointer is unstacked and traced. The maximum stack depth is the maximum number of car pointers (whose corresponding cdr pointer is non-atomic) in any path followed during tracing. Since this is potentially as large as the number of nodes in the system, reserving a stack of this extent is impractical. Implementations generally reserve a stack large enough to cover "reasonable" cases and give a system error if this is exceeded.

The second technique [2] records the addresses of nodes to be revisited in the structure itself. While tracing the car (respectively, cdr) pointer from a node, the car (cdr) field is used to contain the address of the preceding node on the trace path. When the algorithm encounters an atom or marked node, it returns to the preceding node. To determine whether tracing from that node was via the car pointer or via the cdr pointer, an extra flag bit is associated with each node. The bit is turned on when tracing from the car pointer and is turned off when tracing from the cdr pointer. Often there is no room for the flag bit in the node,[1] or the room must be used for other purposes. Hence, a bit map must be used. If there are W bits per word and N nodes in the system, this requires N/W additional words of storage.

Both trace techniques then require fairly large amounts of storage. The second technique needs one bit for each node in the system. The first technique needs a pointer stack equal in depth to the maximum anticipated car path traced; this too may be large.

The Algorithm

The proposed algorithm unites certain desirable properties of the above two. Observe that in the second algorithm it is not necessary to have a flag bit for every node – rather, only for those nodes which must be revisited. At any point in the trace, flag bits are needed only for those nodes on the trace path between the base pointer and the node under consideration. Since the trace path grows and shrinks in last-in-first-out order, a stack of flag bits can be used. Also, observe that nodes with one or more atomic fields require no trace bits. A node with two atomic fields is not traced and so never becomes part of the trace path. When revisiting a node with one atomic field, it is known that the traced field must have been the non-atomic one. Hence, only those nodes on the trace path having two non-atomic fields require a flag bit. The flag bit for the i^{th} such node is kept in the i^{th} position of the bit stack. As the trace path grows and shrinks, the bit stack is pushed and popped correspondingly.

Let push and pop be defined as the stack operations on a bit stack. Let marked(X) be a routine which marks the node X and returns true if and only if the node was previously marked. Let atom(X) be a predicate true only of atoms. For simplicity, we assume that there is a single base pointer B which is the root of all nodes in use.[2] Let Y be initialized to B, and let X be initialized to NIL, the null pointer. The trace algorithm is:

```
NEWNODE: if atom(Y) ∨ marked(Y) ∨ (atom(car(Y))∧atom(cdr(Y)))
           then goto BACKUP;
         if atom(car(Y))
           then begin TEMP ← cdr(Y); cdr(Y) ← X; X ← Y;
                      Y ← TEMP; goto NEWNODE
                end
           else begin TEMP ← car(Y); car(Y) ← X; X ← Y;
                      if not(atom(cdr(Y))) then push(1);
                      Y ← TEMP; goto NEWNODE
                end;
BACKUP:  if X=NIL then goto TRACEDONE;
         if atom(cdr(X))
           then begin TEMP ← car(X); car(X) ← Y; Y ← X;
                      X ← TEMP; goto BACKUP
                end;
         if atom(car(X)) then goto CLIMB;
         FLAG ← pop( );
         if FLAG=1 then begin push(0); TEMP ← cdr(X);
                              cdr(X) ← car(X); car(X) ← Y;
                              Y ← TEMP; goto NEWNODE
                        end;
CLIMB:   begin TEMP ← cdr(X); cdr(X) ← Y; Y ← X;
               X ← TEMP; goto BACKUP
         end;
```

The depth of the bit stack is an issue. The worst case is a single chain in which each node has a non-atomic car and cdr, using all nodes in the system – resulting in a path of length N. Providing for this would require a stack of N bits, i.e., an amount of reserved storage identical to that required by the pointer reversal technique. However, this is clearly a pathological case. For almost all cases of interest, the maximum number of nodes on the trace path having two non-atomic fields will be some small fraction of N and this fraction is all that need be reserved.

Under this policy, the new algorithm is similar in its storage requirements to the first technique. The improvement, a significant one, is that the elements of the bit stack are more than an order of magnitude smaller than the pointers on the stack of resumption points. For a stack consisting of a fixed number of machine words, the bit stack holds far more elements. Hence, deeper structure

can generally be traced. Let P be the number of bits in a pointer. With the stack of flag bits each car and cdr pointer on the trace path takes either one or no bits, depending on whether or not the node containing that pointer has two non-atomic fields. With the stack of resumption points, each cdr pointer takes no bits, while each car pointer takes either P or zero – P if the node contains two non-atomic fields and zero otherwise. Let A_{max} and D_{max} be the maximum number of car and cdr pointers (belonging to nodes with two non-atomic fields) on the trace path followed by the garbage collector in some configuration. The stack of flag bits uses at most $A_{max} + D_{max}$ bits and may use less. The resumption point stack uses $P \cdot A_{max}$. Hence, the proposed algorithm is an improvement whenever

$$P > \frac{A_{max} + D_{max}}{A_{max}}$$

Suppose the nodes are used primarily to represent binary trees of varying depth. The average number of car and cdr pointers, $\overline{A}$ and $\overline{D}$, followed during tracing will be roughly equal. The condition $P > (A_{max} + D_{max})/A_{max}$ will surely be satisfied. In fact, the stack of flag bits will be a considerable improvement. Any configuration which can be traced with a resumption point stack of S words can be traced with a bit stack of 2S/P words.

It must be pointed out that LISP tends to favor a skew in the cdr direction. Hence, the relative performance of the bit stack will not be as great. However, since atomic list elements give rise to nodes with one atomic field (so that no flag bit is required), the bit stack algorithm still tends to work quite well. For example, the list structure (A(BC)(H(IJ(KLM)N))(O(PQ)R)) forms a tree 10 levels deep; yet because many list elements are atoms, only 3 flag bits are required. In contrast, the resumption point technique needs 2 pointers; but, since each pointer is P bits long, this requires a total of 2P bits.

A second advantage of the bit stack arises from its symmetric treatment of car and cdr chains. Successful use of the resumption point stack is rather dependent on the particular organization of the list structure being traced, while the bit stack technique is comparatively robust.

Acknowledgment

The author wishes to thank Mr. J. Spitzen for his helpful comments.

References

[1] J. McCarthy, "Recursive functions of symbolic expressions and their computation by machine," CACM, vol. 3, pp. 184-195, April 1960.

[2] H. Schorr and W. Waite, "An efficient machine-independent procedure for garbage collection in various list structures," CACM, vol. 10, pp. 501-506, August 1967.

Footnotes

This research was supported in part by the U.S. Air Force, Electronics Systems Division, under Contract F19628-71-C-0173 and by the Advanced Research Projects Agency under Contract F19628-68-C-0379.

The author is with the Center for Research in Computer Technology, Harvard University, Cambridge, Massachusetts.

1. There is room in the IBM 7090-7094 since one 36-bit word holds two 15-bit pointers, each spanning the address space (leaving 6 spare bits). There is no room in the PDP 10 since one 36-bit word holds two 18-bit pointers, each spanning the address space (leaving no spare bits). Depending on the representation chosen, there may or may not be room on the IBM 360-370. One representation uses two 32-bit words per node, each word containing a 24-bit pointer which spans the address space (16 bits left over per node). Another representation uses a single 32-bit word per node, containing two 16-bit pointers, each of which can address 64 K words (leaving no spare bits).

2. Allowing a set of base pointers entails only a trivial addition to the algorithm.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)*
Harvard University
Cambridge, Massachusetts 02138

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
N/A

3. REPORT TITLE
A SPACE-EFFICIENT LIST STRUCTURE TRACING ALGORITHM

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*
None

5. AUTHOR(S) *(First name, middle initial, last name)*
Ben Wegbreit

6. REPORT DATE
June 1972

7a. TOTAL NO. OF PAGES
9

7b. NO. OF REFS
2

8a. CONTRACT OR GRANT NO.
F19628-71-C-0174

b. PROJECT NO.

c. ARPA Order No. 952

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
ESD-TR-72-309

9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*

10. DISTRIBUTION STATEMENT
Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY
Deputy for Command and Management Systems
Hq Electronic Systems Division (AFSC)
L G Hanscom Field, Bedford, Mass. 01730

13. ABSTRACT

This note presents an algorithm for tracing during garbage collection of list structure. It requires only one bit for each level of doubly branching structure traced. Compared to existing trace algorithms, it generally requires less storage -- often, substantially less.

DD FORM 1 NOV 65 1473

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
List tracing Garbage collection Storage reclamation Storage regeneration Free storage management List processing						