html transcription

On-the-fly garbage collection: an exercise in cooperation.

by

Edsger W.Dijkstra *)

Leslie Lamport **)

A.J.Martin ***)

C.S.Scholten ****)

E.F.M.Steffens ***)

*) Burroughs, Plataanstraat 5, NL-4565 NUENEN, The Netherlands

**) Massachusetts Computer Associates Inc., 26 Princess Street, WAKEFIELD, Mass. 01880, U.S.A.

***) Philips Research Laboratories, EINDHOVEN, The Netherlands

****) Philips-Electrologica B.V., APELDOORN, The Netherlands

Abstract. As an example of intimate interference between sequential processes, a technique is developed, which allows nearly all of the activity, needed for garbage detection and collection, to be performed by an additional processor operating concurrently with the processor devoted to the computation proper. Exclusion and synchronization constraints have been kept as weak as could be achieved; the severe complexities engendered by this goal are amply illustrated.

Key Words and Phrases: multiprocessing, fine-grained interleaving, cooperation between sequential processes with minimized mutual exclusion, program correctness for multiprogramming tasks, garbage collection.

CR Categories: 4.32, 4.34, 4.35, 4.39, 5.23 .

On-the-fly garbage collection: an exercise in cooperation.

1. Introduction.

In any large-scale computer installation today, a considerable amount of time of the (general purpose) processor is spent on "operating the system". With the advent of multiprocessor installations the question arises to what extent such "housekeeping activities" can be carried out concurrently with the computation(s) proper. Because the more intimate the interference, the harder the organization of the cooperation between the concurrent processes, the problem of garbage collection was selected as one of the most challenging --and, hopefully, most instructive!-- problems. (Our exercise has not only been very instructive, but at times even humiliating, as we have fallen into nearly every logical trap that we could possibly fall into.) In our treatment we have tried to blend a condensed design history --in order not to hide the heuristics too much-- with a rather detailed justification of our final solution. We have tried to keep exclusion and synchronization constraints between the processors as weak as possible, and how to deal with the complexities engendered by that goal is the main topic of this paper. It has hardly been our purpose to contribute specifically to the art of garbage collection, and, consequently, from that point of view no practical significance is claimed for our solution.

We tackled --in the form we knew it-- the garbage collection problem as it presents itself in the traditional implementation environment for pure LISP. There the data structure to be stored consists of a directed graph in which each node has at most two outgoing edges, more precisely: may have a left-hand outgoing edge and may have a right-hand outgoing edge, but either of them or both may be missing. Our discussions were simplified considerably as soon as we followed the --not unusual-- practice of introducing a special purpose node, called "NIL" --with its two outgoing edges pointing to itself-- , and representing a formerly missing edge now by an edge with the node NIL as its target. For us, the introduction of the node called NIL was definitely much more than just a coding trick, for it allowed us to treat "adding an outgoing edge to a node", "removing an outgoing edge from a node" and "redirecting an outgoing edge of a node", which were originally regarded as three different operations, on the same footing, viz. as "redirecting an outgoing edge of a node". In the sequel it will become clear that the homogeneity thus achieved has been absolutely essential for our purposes.

Furthermore all the nodes of the date structure to be stored are "reachable" (via a directed path along the edges) from one or more fixed nodes, called "the roots", and modifications of the data structure are confined to redirecting an outgoing edge of an already reachable node. The roots are given a constant place in memory. The storage allocated to each node is constant in time and equal in size, to be more precise: sufficient for a few bits marking information (see later) and two addresses --for each of the outgoing edges the address of its target node-- . Given (the address of) a node, finding (the address of) its left- or right-hand successor node can be regarded as an atomic, primitive action; finding its predecessor nodes, however, would imply a search through memory.

Besides redirecting an outgoing edge of a node towards an already reachable node, it should be possible to direct it towards a new node, that has to be added to the data structure; such a new node --which upon creation has only NIL as successor node-- is taken from the so-called "free list", i.e. a linearly linked list of nodes that are currently not used for storing a node of the data structure. The difference between those two types of changes of the data structure has been made to disappear, by linking the "free" nodes linearly --via their left-hand outgoing edge, say-- and introducing a special root pointing to the begin node of the free list. (See Appendix 1.) Again this is more than a mere coding trick, for now also the nodes of the free list can be regarded as reachable. By furthermore declaring that also the special purpose node NIL is a root, we achieved our next homogenizing simplification: only <u>one</u> type of change of the data structure is left, viz. redirecting for a reachable node one of its outgoing edges to a reachable node.

Note that redirecting an outgoing edge of a reachable node may turn a number of other, formerly reachable nodes into unreachable ones: they then disappear from the data structure and become what is called "garbage". Garbage may arise anywhere in store, and it is the purpose of the so-called "garbage collector" to detect such disconnected and therefore obsolete nodes and to append them to the free list. In classical LISP implementations the computation proper proceeds until the free list is exhausted (or nearly so). Then the computation proper comes to a grinding halt, during which the processor is devoted to garbage collection. Starting from the roots, all reachable nodes are marked. Because we have made the nodes of the free list reachable from a special root, nodes of the free list (if any) will in our case be marked as well. Upon completion of this marking phase, all unmarked nodes can be concluded to be garbage and are appended to the free list, after which the computation proper is resumed.

The minor disadvantage of this arrangement is the delay of the computation proper; its major disadvantage is the unpredictability of these garbage collecting interludes, which makes it hard to design such systems so as to meet real time requirements as well. It was therefore tempting to investigate whether a second processor -- called "the collector"-- could collect garbage concurrently with the activity of the other processor --for the purpose of this discussion called "the mutator"-- which would be dedicated to the computation proper. In order to investigate a difficult problem, we have imposed upon our solution a number of constraints (compare [2]).

Firstly we wanted to reduce, as much as possible, the overhead on the activity of the mutator (as required for the cooperation with the collector.)

Secondly, we wanted the synchronization and exclusion constraints between the mutator and the collector as weak as possible. (The classical implementation presents in this respect the other extreme: a garbage collecting interlude can in its entirety be regarded as a single critical section that excludes all mutator activity!) We wanted in particular to avoid highly frequent mutual exclusion of "elaborate" activities, as this would defy our aim of concurrent activity: our ultimate aim was something like no more interference than the mutual exclusion of a single read or write of the same single variable. (One synchronization measure is evidently unavoidable: when needing a new node from the free list, the mutator may have to be delayed until the collector has appended some nodes to the free list. This is the now traditional producer/consumer coupling; in the context of this article it must suffice to mention that this form of synchronization can be achieved without any need for mutual exclusion (see [3]). In our very first solution the overhead on the mutator was dependent on the question whether the collector was engaged on an odd or an even major cycle, a dependency which required a synchronization between mutator and collector; in our later solutions the need for this synchronization has been eliminated.)

Thirdly we did not want the mutator's ongoing activity to impair the collector's ability to identify garbage more than we could avoid. With a major cycle of the collector consisting of a marking phase followed by an appending phase, it is impossible to guarantee that the appending phase will append all the garbage existing at its beginning: new garbage could have been created between an appending phase and the preceding marking phase. (We do require, however, that such garbage, existing at the beginning of an appending phase but not identified as such by the collector, will be appended in the next major cycle of the collector.) Solutions we found, in which garbage created during a marking phase was guaranteed <u>not</u> to be appended during the next appending phase, have been rejected for that reason.

<u>2. Preliminary investigations</u>.

A counterexample taught us that the goal "no overhead for the mutator" is unattainable. Suppose that the nodes A and B are permanently reachable via a constant set of edges, while node C is reachable only via an edge from A to C . Suppose furthermore that from then on the mutator performs repeatedly the following sequence of operations

1) making an outgoing edge from B point to C
2) deleting the edge from A to C
3) making an outgoing edge from A point to C
4) deleting the edge from B to C .

The collector, which observes nodes one at a time, will discover that A and B are reachable from the roots, but may never discover that C is reachable as well: while A is observed by the collector, C may be reachable via B only, and the other way round. We may therefore expect that the mutator may have to mark in some way target nodes of changed edges.

Marking will be described in terms of colours. When we start with all nodes white, and, furthermore, can ensure that the combined activity of collector and mutator will make all reachable nodes black, then all white nodes can be identified as garbage. For each repetitive process --and the marking process certainly is one-- we have always two concerns (see[1]): firstly we must have a monotonicity argument on which to base our proof of termination, secondly we must find an invariant relation which, initially true and not being destroyed, will still hold upon termination. For the monotonicity argument we have chosen (fairly obviously) that during marking no node will go back from black to white, or, stated in slightly more general terms:

"at no moment during marking nodes will become lighter".

For the invariant relation --a relation which must be satisfied both before and after the marking cycle-- we must generalize the initial and final states of the marking process, and our first guess was (perhaps less obvious, but not unnatural)

P1: "during marking there will be no edge pointing from a black node to a white one".

Additional action is then required from the mutator when it is about to introduce an edge from a black node to a white one: just placing it would cause a violation of P1 . The monotonicity argument tells us that the black source node of the new edge has to remain black, and, therefore, P1 tells us that the target node of the new edge cannot be allowed to remain white. But the mutator cannot make it just black, because that could cause a violation of P1 between that new target node and its immediate successors. For that reason grey has been introduced as intermediate colour, and for the mutator we have considered the action --of which the "shading" part can be regarded as the overhead--

M1: "the mutator changes an edge and shades its new target" .

Note 0. The "shading" of a node is defined to make a white node grey and to leave the colour of a grey or a black node unchanged. (End of note 0.)

Note 1. Disregarding P1 , the problem of node C , described at the beginning of this section, could also be solved by changing M1 into shading the old target instead of the new one; this would, however, lead to a solution of the type that we have rejected at the end of section 1. (End of note 1.)

As long as the whole of M1 , i.e. the combined changing of an edge and shading its new target, was taken as a single, indivisible "grain of action", the invariant relation P1 was sufficient for a coarse-grained solution (as a matter of fact with the same collector as described in section 3). Encouraged by this success we tried to split action M1 up into two grains of action, one for changing the edge and one for shading the new target. Wanting to keep P1, which had been so successful, invariant, we had to choose as first indivisible grain of action the shading of the future target, and as second grain redirecting the edge towards the node just shaded, because doing it the other way round would have caused a temporary violation of P1 . Our solution, although presented in a manner sufficiently convincing to fool ourselves, contained a bug found by N.Stenning and M.Woodger [6]. If, after shading the future target, the mutator went to sleep, the collector could in its appending phase make that future target white again, in the next marking phase it could make the source black, whereafter the mutator would wake up again and could introduce a black-to-white edge; the new target node could then erroneously be considered as garbage.

In short: in the finer-grained solution we were heading for, total absence of an edge from a black node to a white one was a stronger relation than we could maintain. We could, however, retain the notion "grey" as "semi-marked", more precisely, as representing the unfulfilled marking obligation: as before, the marking activity of the collector remains localized at grey nodes and their possible white successors. Again we tried to find and justify our fine-grained solution by using a --this time different!-- coarse-grained solution as stepping stone.

3. A coarse-grained solution.

Relation P1 has to be replaced by a weaker one. (It will be replaced by P2 and P3 , as defined below.) In our previous effort we had made essential use of the fact that, after the collector had initialized the marking phase by shading all roots, the validity of P1 allowed us to conclude that the existence of a white reachable node implied the existence of a grey node (even of a grey reachable node, but the reachability of such an existing grey node was not essential). A weaker relation, from which the same conclusion can be drawn, is

P2: "during the marking cycle (which the collector has initialized by shading all roots) there exists for each white reachable node a so-called "propagation path", leading to it from a (not necessarily reachable) grey node, and consisting solely of edges with white targets (and, as a consequence, without black sources)."

Note 2. In the absence of edges from a black node to a white one, relation P2 is clearly satisfied. (End of note 2.)

Relation P2 is strong enough to draw at the end of the marking cycle the desired conclusion that all white nodes are garbage; all by itself, however, it is too weak to be kept invariant. It can be kept invariant when we restrict the existence of black-to-white edges by P3 --analogous to P1 , but weaker-- given by

P3: "during the marking cycle only the last edge placed by the mutator may lead from a black node to a white one".

Note 3. In the absence of black nodes, P3 is trivially satisfied. (End of note 3.)

When the mutator redefines an outgoing edge of a black node, it may direct it towards a white node: this new edge from a black node to a white one is permitted by P3 , but because the previously placed one could still exist and be of the same type, we consider for the mutator (instead of the earlier M1) , for the time being as a single grain of action:

M2: "the mutator shades the target of the edge previously placed by it and changes an edge".

Note 4. For the very first time that the mutator changes an edge, we can assume that, for lack of a previously placed edge, the shading will be suppressed or an arbitrary reachable node will be shaded; the choice does not matter for the sequel. (End of note 4.)

Action M2 has been carefully chosen in such a way that it leaves P3 invariant; it leaves, however, the stronger relation P2 and P3 invariant as well.

Proof. The action M2 cannot introduce new reachable nodes; it, therefore, does not introduce new white ones, for which extra propagation paths must exist. If the node whose successor is redefined is black, its outgoing edge that may have disappeared as a result of the change, was not part of any propagation path, and the edges of the old propagation paths will be sufficient to provide the new propagation paths. (Possibly we don't need all of them as a result of the shading and/or white reachable nodes having become unreachable.) If the node whose successor is redefined was white or grey to start with, the net result of action M2 will be a graph without edges from a black node to a white one: if one existed, its target has now been shaded, and no new one has been introduced as the source of the new edge is not black. (It may have been shaded!) By Note 2, P2 still holds. (End of proof.)

We have now reached the stage where we can describe our first collector, which repeatedly performs the following program. (Our bracket pairs "if...fi" and "do...od" delineate our alternative and repetitive constructs respectively (see [1]), comments have been inserted between braces, and labels have been inserted for the discussion.) The program has two local integer variables i and k ; the nodes in memory are assumed to be numbered from 0 through M-1 .

```
marking phase:
begin {there are no black nodes; hence (see Note 3)  P3  holds}
  C1: "shade all the roots" {P2 and P3};
      i:= 0; k:= M;
  marking cycle:
      do k > 0 → {P2 and P3}
            if C2: "node nr. i  is grey" →
                        k:= M;
                     C3: "shade the successors of node nr. i  and make node
                        nr. i  black" {P2 and P3}
             ▯ non C2: "node nr. i  is not grey" →
                        k:= k - 1  {P2 and P3}
            fi {P2 and P3};
            i:= (i + 1)mod M
      od {P2 and P3  and there are no grey nodes, hence all white nodes are
           garbage}
end;
```

```
appending phase:
begin i:= 0;
      do i < M → {a node with a number < i cannot be black;
                  a node with a number ≥ i  cannot be grey,
                  and is garbage, if white}
            if C2: "node nr. i  is white" →
                   C4: "append node nr. i  to the free list"
             ▯ C2: "node nr. i  is black" →
                   C5: "make node nr. i  white"
            fi;
            i:= i + 1
      od {there are no black nodes}
end
```

The indivisible actions of the collector --between the executions of which actions M2 of the mutator may occur-- are

1) "shading of a single root" (from which C1 is composed: the order in which the roots are shaded is irrelevant)

2) establishing the current colour of node nr. i (labeled "C2")

3) the total actions C3 , C4 (see, however, the Appendix) and C5 .

Remark 1. With a more elaborate administration local to the collector --a list of grey or possibly grey nodes-- a probably much more efficient marking phase could have been designed. For the sake of simplicity we have not done so. (End of remark 1.)

We observe that (even independent of the colour of node nr. i !) action C3 : "shade the successors of node nr. i and make node nr. i black" can never cause a violation of P2 and P3 : the shading of the successors can never do any harm; as a result of the shading the outgoing edges of node nr. i are no longer needed for a propagation path, and making node nr. i black maintains the existence of the propagation paths needed, without introducing an edge from a black node to a white one.

Marking has been completed when all grey nodes have disappeared. Its detection by the collector, however, presents some problems. In the marking cycle of the collector as given above, k is reset to M each time the collector encounters a grey node, and the marking cycle terminates with a scan past all nodes, during which no grey nodes have been encountered. If we had only the collector to consider, the conclusion that upon termination of the marking cycle grey nodes are absent would be trivial; due to the on-going activity of the mutator --the shading activity of which can introduce grey nodes!-- a more subtle argument is required to justify the conclusion that after a collector scan past all nodes, during which no grey nodes have been encountered, there are, indeed, no grey nodes. The argument is as follows.

Absence of grey nodes implies on account of P2 that all white nodes are garbage and that all reachable nodes are black. Hence, the state characterized by the absence of grey nodes is stable during marking: the absence of white reachable nodes prevents the mutator from introducing grey ones and the absence of grey nodes prevents the collector from doing so. Consider now the collector scan past all nodes during which no grey nodes have been encountered. Because the mutator leaves a grey node grey, no grey node can have existed at the beginning of that scan, i.e. the stable state must already have been reached at the beginning of that scan.

Finally, termination of the marking cycle is guaranteed because of the monotonicity of the colouring history of each node, and because of the fact that resetting k to M is always accompanied by effective darkening of at least one node (nr. i to be precise).

When the appending phase starts, all reachable nodes are black and all white nodes are garbage. Note that the existence of black garbage is not excluded. The appending phase deals with each node in turn: as long as it has not been dealt with (i.e. has a number $\geq i$) it cannot change colour: if black, it remains black because the mutator can only shade it, and if it is white, it is garbage and, by definition, the mutator won't touch it. As soon as it has been dealt with (i.e. has a number $< i$), it has been white and can at most have been shaded by the mutator. Black garbage at the beginning of the appending phase will not be appended during that appending phase, it will only be made white; during the next marking phase it will remain white, and the next appending phase will indeed append it. Therefore, no garbage, once created, will escape being collected.

4. A solution with a fine-grained collector.

In this section we show how the course-grained solution of section 3 can be used as a stepping stone for a solution which admits, as far as the collector is concerned, a finer grain of interleaving. In particular we shall show how C3 can be broken open as a succession of five indivisible subactions, say (m1 and m2 being local variables of the collector):

```
C3.1:  m1:= number of the left-hand successor of node nr. i ;
C3.2:  shade node nr. m1 ;
C3.3:  m2:= number of the right-hand successor of node nr. i ;
C3.4:  shade node nr. m2 ;  |
C3.5:  make node nr. i  black  |      .
```

None of the actions C3.1 , C3.2 , C3.3 , and C3.4 can cause a violation of P2 <u>and</u> P3 . The actions C3.1 and C3.3 cannot do so because they leave no trace in common memory, and the actions C3.2 and C3.4 cannot do so, because shading cannot do so. Besides that, because shading of a node commutes with any number of actions M2 of the mutator, we have, by the time that the collector starts with C3.5 , a state as if the shading of node nr. m1 had been part of C3.1 and the shading of node nr. m2 had occurred simultaneously with C3.3 . Without loss of generality we can, therefore, continue our discussion as if "shade right-hand successor" and "shade left-hand successor" are available as indivisible actions. The problem, however, lies with C3.5 : can we safely make node nr. i black? Note that neither m1 , nor m2 needs still to be one of its successors: m1 and m2 even never need to have been its left- and right-hand successor simultaneously! A more thorough study of the mutator, however, reveals that it is safe.

<u>Proof</u>. During the marking phase we define a changing set of edges to which we given the meaningless term "Q-edges".

<u>Remark 2</u>. We could only get the argument straight after the introduction of a meaningless term such as "Q-edges". At first we used terms like "inessential" or "dispensable", etc., but they all messed up our thinking. The problem is that an edge that is not "inessential" --in the above, special meaning of the word-- is not "essential" --in the normal sense of the word-- . It was very instructive to experience how totally misleading the choice of so-called

"meaningful identifiers" can be. (End of remark 2.)

Remark 3. Note that we only define the set of Q-edges for our benefit. The mutator and collector would have a hard time if they had to update it explicitly: in the jargon the term "ghost variable" is sometimes used for such an entity. (End of remark 3.)

The set of Q-edges is defined as follows as a function of the evolving computations:

1) at the beginning of the marking phase the set of Q-edges is initialized with all the edges with a grey target

2) each time a white node becomes grey, all its incoming edges (that were not already a Q-edges) are added to the set of Q-edges

3) when action M2 , seen as a replacement of an outgoing edge, replaces a Q-edge --or an edge that, according to the second rule, would have become one as a consequence of M2's shading act-- the new edge that replaces it, is also a Q-edge : it "inherits the Q-ness" from the edge it replaces.

The above rules imply that a Q-edge is never needed for a propagation path. The third rule, all by itself, implies that once the left-hand outgoing edge of a node is a Q-edge, it will remain so, no matter how often redirected by the mutator, and that the same holds for the right-hand outgoing edge. In short: when, since the beginning of the marking phase, a given node has had a grey left-hand successor and has had a grey right-hand successor, it has two outgoing Q-edges, and making it black will never cause violation of P2 . It won't violate P3 either: if it has a white successor, the corresponding edge must have been the last one placed by the mutator (it can therefore have at most one white successor) and that edge from a black node to a white one is the edge explicitly allowed by P3 . (End of proof.)

Note 5. In breaking up C3 we have placed C3.5 "make node nr. i black" at the end. As making a node black commutes with all other actions M2 and C3.1 through C3.4 , we could also have placed it at the beginning, before dealing (in some order) with the successors; P2 and P3 could then be violated temporarily. (End of note 5.)

5. A solution with a fine-grained mutator as well.

The treatment in the previous section was greatly simplified by the circumstance that the mutator could only interfere via a single type of action: as far as the collector was concerned, the mutator activity could be considered as a stream of actions M2, and it was therefore, so to speak, not necessary to take the mutator's "instruction counter" into account. It was the possibility of this simplification that motivated us to postpone the transition to a fine-grained mutator till the end; this postponement was the more seductive because we thought that the following very simple argument would justify that transition.

It is obvious that no harm is done if at random moments a daemon would shade a reachable node. We now assume a friendly daemon that, between any two successive actions M2 of the mutator, shades the target of the last placed edge, and the idea was that then it would make no difference if M2 did not do the shading anymore, and that therefore it was safe to replace M2 by the succession of the following two separate indivisible subactions:

"redirect for a reachable node an outgoing edge towards a reachable node";
"shade the target of the edge just placed" .

It was David Gries who pointed out to us that, although the conclusion was correct, the argument contained a flaw: the friendly daemon's activity does not exclude that, at the beginning of an action M2 , the node just shaded by the daemon in the meantime has been made white again by the collector, and the combination of daemon with shading M2 is therefore not necessarily semantically equivalent to the combination of daemon with M2 , from which the shading has been removed.

The full argument is that, with our friendly daemon, for the initial state of an action M2 during a marking cycle we can assert (besides P2 <u>and</u> P3) the absence of an edge from a black node to a white one, regardless of the question whether the last shading by our friendly daemon took place during the current marking phase or earlier: in the first case we observe that the only edge allowed by P3 to point from a black node towards a white one has a target that is at least grey, in the second case we observe that the marking phase started without edges pointing from a black node to a white one and that during marking the collector cannot introduce them all by itself. As a result of the guaranteed absence of an edge from a black node to a white one, the proofs that M2 leaves P2 <u>and</u> P3 invariant is now also valid if M2 does not shade at all.

<u>Remark 4</u>. The detailed implementation of what we have described as "a grain of interleaving" falls very definitely outside the scope of this paper: many techniques --even allowing concurrent access to the same unit of information-- are possible (see[3]). (End of remark 4.)

<u>In retrospect</u>.

It has been surprisingly hard to find the published solution and justification. It was only too easy to design what looked --sometimes even for weeks and to many people-- like a perfectly valid solution, until the effort to prove it to be correct revealed a (sometimes deep) bug. Work has been done on formal correctness proofs ([4], [5]), but a shape that would make them fit for print has, to our tastes, not yet been reached. Hence our informal justification (which we do <u>not</u> regard as an adequate substitue for a formal correctness proof!). Whether its stepwise approach --which this time seems to have been successful in reducing the case analyses-- is more generally applicable, is at the moment of writing still an open question.

When it is objected that we still needed rather subtle arguments, we can only agree whole-heartedly: all of us would have preferred a simpler argument! Perhaps we should conclude that constructions that give rise to such tricky problems are not to be recommended. One firm conclusion, however, can be drawn: to believe that such solutions can be found without a very careful justification is optimism on the verge of foolishness.

<u>History and acknowledgements</u>. (As in this combination this is our first exercise in international and inter-company cooperation, some internal credit is given as well.) After careful consideration of a wider class of problems the third and fifth authors selected and formulated this problem, and did most of the preliminary investigations; the first author found a first solution during a discussion with the latter, W.H.J.Feijen and M.Rem. It was independently improved by the second author --to give the free list a root and mark its nodes as well, was his suggestion-- and, on a suggestion made by John M.Mazola,

by the first and the third author. The first and the fourth merged these embellishments, but introduced the bug that was found by N.Stenning and M. Woodger. The final version and its justification are the result of the four authors in the Netherlands. The active and inspiring interest shown by David Gries is mentioned in gratitude.

References.

1. Dijkstra, Edsger W., Guarded Commands, Nondeterminacy and Formal Derivation of Programs. Comm. ACM 18, 8 (Aug. 1975), 453-457.
2. Steele Jr., Guy L., Multiprocessing Compactifying Garbage Collection. Comm. ACM 18, 9 (Sep. 1975), 495-508.
3. Lamport, Leslie, On Concurrent Reading and Writing. (Submitted to the Comm. ACM.)
4. Gries, David, An Exercise in Proving Parallel Programs Correct. (Submitted to the Comm. ACM.)
5. Lamport, Leslie, Report CA-7508-0111, Massachusetts Computer Associates, Inc.
6. Woodger, M., Private Communications.

Appendix

Here we give an example of how the free list and the operations such as taking a node from or appending a node to the free list can be implemented. We consider the nodes of the free list ordered according to "age". For each node in the free list, the right-hand successor is NIL , the left-hand successor is NIL for the youngest node and is the next-younger one for the others. We have a root called TAKE , its left-hand successor and its right-hand successor are both the oldest free node; we have a second root called APP , whose left-hand and right-hand successor are both the youngest free node.

Taking a free node --and making it the left-hand successor of some reachable node X , say-- can be done in the following steps (shown in a hopefully self-explanatory notation):

```
X.left:= TAKE.left;               (All four actions should follow
TAKE.left:= TAKE.right.left;       the shading convention chosen.)
TAKE.right.left:= NIL;
TAKE.right:= TAKE.left
```

Appending, say, node Y --in action C4-- could be done by:

```
Y.left:= NIL; Y.right:= NIL;
APP.left:= Y;
APP.right.left:= Y;
APP.right:= APP.left
```

When a minimum of two free nodes is maintained, the collector that appends is certain only to deal with nodes that are left alone by the mutator, and the action C4 need not to be regarded as a single indivisible action, but is trivially allowed to be broken up in the above subactions. The synchronization guaranteeing the lower bound for the length of the free list is here supposed to have been implemented by other, independent means.

5th of January 1977