Copyright Notice

The following manuscript

> EWD 630: On-the-fly garbage collection: an exercise in cooperation

was published in

> *Commun. ACM 21* (1978), 11: 966–975.

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, 5671 AL 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 cooperation between sequential processes with very little mutual interference despite frequent manipulations of a large shared data space, 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 doing so are 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, 5.24 .

(For reference purposes a glossary of names has been added at the end of the article.)

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. One of the problems that have to be dealt with is that of organizing the cooperation of the concurrent processes in such a way as to keep exclusion and synchronization constraints extremely weak, in spite of very frequent manipulations (by all processes involved) of a large shared data space. The problem of garbage collection was selected as one of the most challenging problems in this respect (and hopefully a very instructive one). 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 presentation 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 processes as weak as possible, and how to deal with the complexities engendered by doing so is the main topic of this paper.

It has hardly been our purpose to contribute specifically to the art of garbage collection, and consequently no practical significance is claimed for our solution. For that reason we felt justified in tackling a specific form of the garbage collection problem as it presents itself in the traditional implementation environment of pure LISP. We are aware of the fact that we have left out of consideration several aspects of the garbage collection problem that are important from other points of view (see, for instance, [2]).

In our abstract form of the problem, we consider a directed graph of varying structure but with a fixed number of nodes, in which each node has at most two outgoing edges. More precisely, each node may have a left-hand outgoing edge and may have a right-hand outgoing edge, but either of them or both may be missing. In this graph a fixed set of nodes exists, called "the roots". A node is called "reachable" if it is reachable from at least one root via

a directed path along the edges. The subgraph consisting of all reachable nodes and their interconnections is called "the data structure"; non-reachable nodes, i.e. nodes that do not belong to the data structure, are called "garbage nodes". The data structure can be modified by actions of the following types:

(1) redirecting an outgoing edge of a reachable node towards an already reachable one

(2) redirecting an outgoing edge of a reachable node towards a not yet reachable one without outgoing edges

(3) adding --where an outgoing edge was missing-- an edge pointing from a reachable node towards an already reachable one

(4) adding --where an outgoing edge was missing-- an edge pointing from a reachable node towards a not yet reachable one without outgoing edges

(5) removing an outgoing edge of a reachable node.

In actions (1), (2), and (5) nodes may be disconnected from the data structure and thus become garbage. In actions (2) and (4) a garbage node is "recycled", i.e. made reachable again.

The representation of the graph is such that each node can be identified independently of the structure of the graph, and that finding the left- or right-hand successor of a node can be regarded as a primitive operation, whereas finding its predecessor nodes would imply a search through the complete collection of nodes. Because of this representation, finding garbage is a non-trivial task, which is delegated to a so-called "garbage collector". The garbage collector maintains a so-called "free list", i.e. a collection of nodes that have been identified as garbage and are available to be added to the data structure.

In classical LISP implementations the computation proper (i.e. the modifications of the data structure as described above) proceeds until the free list is exhausted (or nearly so). Then the computation proper comes to a grinding halt, after which garbage is collected: starting from the roots, all reachable nodes are marked; upon completion of this marking cycle 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 an exemplary problem, we have imposed upon our solution a number of constraints (compare [2]).

Firstly, we wanted the synchronization and exclusion constraints between the mutator and the collector to be 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 and/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.

Secondly, we wanted to keep the overhead on the activity of the mutator (as required for the cooperation with the collector) as small as possible.

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 <u>all</u> 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. Moreover, we have rejected solutions in which garbage created during a marking phase was guaranteed <u>not</u> to be appended during the next appending phase.

2. The grain of action.

The fact that we require concurrent operation of two or more processes raises the problem of defining the net effect of such concurrent operation. In order to explain the problem we introduce the terms "local variable" for those variables that are accessed by one process only, and "shared variable" for those that are accessed by at least two processes.

As long as our concurrent processes only operate on local variables, there is no problem: we suppose that no one will have any doubt as to the net result of the concurrent operation of the two programs S_0 and S_1 given by

S_0: x:= 0 and S_1: y:= 3

This, however, changes radically as soon as we consider shared variables. With shared "z" , some readers may assume that the concurrent operation of S_2 and S_3 , given by

S_2: z:= 0 and S_3: z:= 3

will yield either $z = 0$ or $z = 3$, but in that case we must destroy that illusion! We need only assume z to consist of two bits z_0 and z_1 $(z = 2z_1 + z_0)$, and S_2 and S_3 on closer scrutiny to be composed as follows:

S_2: z_0:= 0; z_1:= 0 and S_3: z_0:= 1; z_1:= 1

to reach the conclusion that $z = 1$ and $z = 2$ are also possible results.

In order to express our intentions unambiguously, we introduce the notion of "atomic operations", denoted in our texts by a piece of program placed between a pair of angle brackets (we don't allow nested use of such pairs). We further require all accesses to shared variables to be part of an atomic operation and postulate that the net effect of our concurrently operating processes is as if atomic operations are mutually exclusive, i.e. the execution periods of atomic operations don't overlap. (We note in passing that it is pointless to introduce atomic operations accessing local variables only.) As a result it is now clear that concurrent operation of

S_2: $\langle z:= 0 \rangle$ and S_3: $\langle z:= 3 \rangle$

will, indeed, yield either $z = 0$ or $z = 3$ (even if, upon closer scrutiny, the assignments to z turn out to be composed of successive operations on the individual bits).

Having introduced atomic operations, we are now in a position to define a (partial) ordering between programs based on the notions "coarser-grained" and "finer-grained" ("A is coarser-grained than B" is equivalent with "B is finer-grained than A"). We say that A is coarser-grained than B (or alternatively, "has a coarser grain of action") if B is the result of replacing an atomic operation of A by a piece of program containing at least two atomic operations, and having all by itself the same net effect as the original operation.

Since a possible sequencing of the atomic operations in a coarse-grained solution of a problem can always be regarded as a possible sequencing of the atomic operations in a finer-grained solution, the proof that the finer-grained solution is correct implies the same for the coarse-grained solution. Hence, the advantage of coarser-grained solutions is that their correctness proofs are easier than those for finer-grained ones; their disadvantage, however, is that their implementation usually requires more severe mutual exclusion measures, which tend to defeat the aim of concurrency.

3. A reformulation of the problem.

Our first step was to restate the problem in as simple a form as we could. We found two important simplifications.

First, we followed the not unusual practice of introducing a special root node, called "NIL", whose two outgoing edges point to itself, and representing a formerly missing edge now by an edge with the node NIL as its target. (In order to shorten our discussions we use the terms "source" and "target" of an edge: if an edge points from node A to node B, then A is said to be the source and B is said to be the target of that edge.) For us, the introduction of the node NIL was definitely much more than just a coding trick. It allowed us to view data structure modifications of types (3) and (5) as special cases of type (1), and those of type (4) as special cases of type (2), so that we were left with two types of modification only. In the sequel it will become clear that the reduced diversity thus achieved has

been absolutely essential for our purposes.

A second simplification was obtained by viewing the nodes of the free list no longer as garbage, but as part of the data structure. This was achieved by introducing one or more special root nodes, and by linking the free nodes in such a way that NIL and all free nodes, but no others, are reachable from these special root nodes. This implies that from now on the nodes on the free list are reachable, and thus considered to be part of the data structure. A modification of type (2) is now replaced by a sequence of modifications of type (1): first redirecting an edge towards a node in the free list, then redirecting edges of free list nodes so as to remove that node from the free list. (Note that the operations must be performed in such an order that the node in question remains permanently reachable.) Making the free list part of the data structure is again no mere coding trick. It allowed us to eliminate modifications of type (2): now only one type of modification of the data structure is left to the mutator, namely type (1) "redirecting an outgoing edge of a reachable node towards an already reachable one". (Even the actions of the collector, required for appending an identified garbage node to the free list, are very close to the one operation available to the mutator. The only difference is that we have to allow the collector to redirect the outgoing edge of a reachable node towards a not yet reachable one.)

The activities of mutator and collector can now be described as repeated executions of:

mutator: "redirect an outgoing edge of a reachable node towards an already reachable one"

collector: marking phase:
"mark all reachable nodes";
appending phase:
"append all unmarked nodes to the free list and remove the markings from all marked nodes" .

The mutator and the collector must cooperate in such a fashion that the following two correctness criteria are satisfied.

CC1: Every garbage node is eventually appended to the free list. More precisely, every garbage node present at the beginning of an appending phase will have been appended by the end of the next appending phase.

CC2: Appending a garbage node to the free list is the collector's only modification of (the shape of) the data structure.

Our final goal was a fine-grained solution in which each atomic operation would be something like a single read or write of a variable. More precisely, we wanted in our final solution to accept the following atomic operations: "redirecting an edge", "finding the (left- or right-hand) successor of a node", and "testing and/or setting certain attributes of a node". (The latter class of operations is obviously needed for marking the nodes.) The implementation of these atomic operations falls outside the scope of this paper.

Moreover, we allow ourselves the (not essential but convenient) luxury of considering "append node nr. i to the free list" to be an atomic operation available to the collector. We felt entitled to do so because its finer-grained implementation in terms of a succession of redirection of edges is simple provided the free list remains long enough, for then the nodes involved are not touched by the mutator. Nor do we describe how to prevent the free list from getting too short, i.e. how to delay the mutator, if necessary, when it is about to take a node from the free list. (The latter is the familiar consumer/producer coupling, a fine-grained solution of which has been given in [3].)

We have taken the course of first finding a coarse-grained solution, and then transforming it into a finer-grained one. This two-stage approach has been of great heuristic value; it was, however, not without pitfalls.

4. The first coarse-grained solution.

A counterexample taught us, that the goal "no extra 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 initially reachable only via an edge from A to C. Suppose furthermore that, from then on, the mutator performs repeatedly a sequence of redirections with the following results:

1) making an outgoing edge from B point to C
2) making the edge from A to C disappear

3) making an outgoing edge from A point to C

4) making the edge from B to C disappear.

Since the collector observes nodes one at a time, it may never discover that C is reachable: while the collector is observing A for its successors, C may be reachable via B only, and the other way round. We therefore expect that the mutator may have to mark in some way the target nodes of edges it redirects.

Marking will be described in terms of colours. We start with all nodes white, and will design the algorithm so that the combined activity of the collector's marking phase and the mutator will make all reachable nodes black. All nodes that are still white after the marking phase will thus be garbage. For any repetitive process --and the marking phase 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, and secondly, we must find an invariant relation which is initially true and not destroyed during the repetition, so that it still holds upon termination. For the monotonicity argument we choose (fairly obviously) that during the marking phase no node will go back from black to white. Since we will soon introduce the colour "grey", we restate this more generally as: "during the marking phase no node will become lighter". (Grey is darker than white and lighter than black.) For the invariant relation --which must be satisfied both before and after the collector's marking cycle-- we must generalize the initial and final states of the marking cycle. Our first choice (perhaps less obvious, but not unnatural) was:

P1: "no edge points from a black node to a white one".

Additional action is now required from the mutator when it is about to introduce an edge from a black node to a white one, since just placing it would cause a violation of P1. The monotonicity argument requires that the black source node of the new edge has to remain black, so P1 tells us that the target node of the new edge cannot be allowed to remain white. But the mutator cannot just make it black, because that could cause a violation of P1 between the new target node and its immediate successors. We therefore introduce the intermediate colour "grey", and let the mutator change the new target's colour from white to grey; for reasons of simplicity, the mutator shall do so independently of the colour of the new edge's source. Our choice was a coarse-

grained mutator that repeatedly performs the following atomic operation, in which "shading a node" means making it grey if it is white, and leaving it unchanged if it is grey or black:

M1: < redirect an outgoing edge of a reachable node towards an already reachable one, and shade the new target > .

Note 1. Disregarding P1 , the problem of node C from the counterexample at the beginning of this section could also have been solved by having the mutator shade the old target instead of the new one. This, however, would lead to a solution in which garbage created during a marking phase is guaranteed not to be collected during the next appending phase. Hence, we rejected this solution in accordance with the last sentence of section 1. (End of note 1.)

We have decided that the collector's marking phase should make all reachable nodes black while keeping P1 invariant. This decision leads fairly directly to a coarse-grained collector. Like the mutator, the collector's marking phase uses the intermediate colour grey to preserve P1 . Grey nodes are then ones which must be made black, but which might still have white successors. Hence, whenever it encounters a grey node, the marking phase must make it black and shade its successors. For our coarse-grained collector, let the entire operation of blackening a grey node and shading its successors be a single atomic operation. Since the marking phase must make all grey nodes black, it can terminate only when there are no more grey nodes. The obvious way of trying to guarantee the absence of grey nodes is to let the marking phase terminate when the collector had observed all nodes in some order without finding any grey ones. This produces the collector given below; it is described with the "if...fi" and "do...od" constructs introduced in [1]. (The idea of "B → S" , a so-called "guarded command", is that the statement list S is only eligible for execution in those initial states for which B is true. Below we need only a few simple cases. The repetitive construct "do B → S od" is semantically equivalent to the now traditional "while B do S od". The alternative construct "if B1 → S1 ▯ B2 → S2 fi" requires B1 or B2 to hold to start with; if B2 were non B1 , it would be semantically equivalent to the now traditional "if B1 then S1 else S2 fi" .) As before, angle brackets are used to enclose atomic operations. Comments have been inserted between braces, and labels have been inserted for future reference.

The collector has two local integer variables i and k , and a local variable c of type colour; the nodes are assumed to be numbered from 0 through M-1 . Our coarse-grained collector then repeatedly executes the following program:

```
marking phase:
begin {there are no black nodes}
      "shade all roots" {P1 and there are no white roots};
      i:= 0;  k:= M;
   marking cycle:
      do k > 0 → {P1 and there are no white roots}
            < c:= colour of node nr.i > ;
            if c = grey → k:= M;
                      C1: < shade the successors of node
                            nr.i and make node nr.i black >
             [] c ≠ grey → k:= k - 1
            fi;
            i:= (i+1) mod M
      od
end {P1 and there are no white roots and no grey nodes,
     hence --as is easily seen-- all white nodes are garbage};
appending phase:
begin i:= 0;
   appending cycle:
      do i < M → { all nodes with a number < i are nonblack;
                   all nodes with a number ≥ i are nongrey,
                   and are garbage, if white}
            < c:= colour of node nr.i > ;
            if c = white → < append node nr.i to the free list >
             [] c = black → < make node nr.i white >
            fi;
            i:= i+1
      od {there are no black nodes}
end
```

Note 2. Appending node nr.i to the free list includes redirecting its outgoing edges so that no other nodes than NIL or free nodes can be reached from it (see our second simplification as described in section 3). (End of note 2.)

Note 3. It does not matter in which order nodes are examined during the marking phase, so we could have written a more general algorithm that does not specify any fixed order. Such an algorithm would allow more efficient implementations of the marking phase, in which the collector maintains a list of grey or probably grey nodes. For the sake of simplicity, we have not done so. (End of note 3.)

We shall now demonstrate that the correctness criteria CC1 and CC2 are met.

Proof. In order to prove that CC2 is met, we observe that, because in the marking phase the collector does not change (the shape of) the data structure, it suffices to show that during the appending phase it appends only garbage nodes to the free list. Because node nr.i is appended after having been observed to be white, it suffices to show that the relation

"a white node with a number $\geq$ i is garbage"

1) is an invariant of the appending cycle's repeatable action
2) holds when the collector enters its appending cycle.

1) We shall demonstrate the invariance first, and shall do so by first proving it for the appending cycle's repeatable action in isolation, and then showing that the mutator leaves that proof's assertions invariant.

Because in the appending cycle's repeatable action i is increased, the collector could only violate the relation by making a non-garbage node white or by making a (white) garbage node into non-garbage. By the alternative construct either violation is possible, but only with respect to node nr.i; we can still guarantee "a white node with a number $>$ i is garbage", from which it follows that the subsequent increase i:= i+1 restores the original relation.

Because i is a local variable of the collector, also the mutator could only violate the assertions either by making a non-garbage node white --which it doesn't, because M1 only shades-- or by making a (white) garbage node into

non-garbage --which it doesn't either, because M1 only redirects edges towards already reachable nodes and, hence, leaves garbage garbage-- . Therefore, the mutator's actions do not invalidate the demonstration that the relation is an invariant of the appending cycle's repeatable action.

2) To show, next, that the relation holds at the beginning of the appending cycle, we have to demonstrate (because i = 0) that the marking phase has established that "all white nodes are garbage", which shall be shown under the assumption that, at the beginning of the marking phase, there were no black nodes.

Because the absence of black nodes implies P1 , and because M1 and C1 have been carefully designed so as to leave P1 invariant and not to introduce white roots, "P1 and there are no white roots" is clearly established before and kept invariant during the marking cycle. When furthermore all grey nodes have disappeared, our target state, in which all reachable nodes are black and all white nodes are garbage, has been reached.

The marking cycle terminates with a scan past all nodes, during which no grey nodes are encountered. If we had only the collector to consider, the conclusion that at the end of such a scan grey nodes are absent --and hence the target state has been reached-- would be trivial. Due to the ongoing activity of the mutator --the shading activity of which can introduce grey nodes!-- a more subtle argument, which now follows, is required.

Firstly, we observe that the target state, characterized by the absence of grey nodes, is stable: 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.

Secondly, we show that a collector scan past all nodes, during which no grey nodes are encountered, implies that the stable target state has already been reached at the beginning of that scan: because the mutator leaves grey nodes grey and the collector did not colour any nodes during that scan, a grey node existing at its beginning would, in contradiction to the assumption, have been encountered during that scan. Hence we can conlude that upon termination

of the marking phase all white nodes are, indeed, garbage.

Because the appending phase makes all black nodes white, and the mutator does not introduce black nodes, there are no black nodes at the end of the appending phase; this justifies the assumption made above, that there would be no black nodes at the beginning of the marking phase. Thus we have completed the proof that starting the collector in the absence of black nodes ensures that CC2 is met.

* . * *

To prove that CC1 is met, we must first show that the collector's two phases terminate properly.

Proper termination of the appending phase is obvious, except for one thing: node nr.i must be black or white, because the alternative construct does not cater for the case "c = grey". But we have already proved that at the end of the marking phase, there are no grey nodes and every white node is garbage. Since the mutator cannot shade a garbage node, and shading a black node has no effect, it is clear that every node is either black or white when it is examined during the appending phase.

Termination of the marking phase follows from the fact that the integral quantity

k + M *(the number of non-black nodes)

--which, by definition, is non-negative-- is left invariant by the activity of the mutator, and is decreased by at least one in each iteration of the marking cycle.

Consider now the situation at the beginning of an appending phase. At that moment, the nodes are partitioned into three sets:

-- the set of reachable nodes (they are black)
-- the set of white garbage nodes (during the first appending phase to come, they will be appended to the free list)
-- the set of black garbage nodes (during the first appending phase to come, they will not be appended to the free list, but they will be made white).

Calling the last set the set of "D-nodes", we have to show that all D-nodes will be appended during the second appending phase to come.

We call an edge "leading into D" when its target is a D-node, but its source is not. Because D-nodes are garbage, we can state that at the beginning of the first appending phase, sources of edges leading into D are white garbage nodes.

Since the D-nodes are garbage, the mutator will not redirect edges so as to make them point to a D-node, and since they are black to start with, during the first appending phase the collector won't do so either. The collector, however, will append all white garbage nodes, which includes --see Note 2-- redirecting outgoing edges of the nodes appended, so that, as a result, we can state that at the end of the first appending phase
-- all D-nodes are white garbage nodes
-- there are no edges leading into D .

The absence of edges leading into D is an invariant for the subsequent marking phase: the mutator does not introduce them, because D-nodes are garbage, and the collector does not redirect edges during its marking phase. The continued absence of edges leading into D, plus the fact that all D-nodes are white garbage to start with, implies that the D-nodes remain white garbage nodes during the subsequent marking phase: because they are garbage, the mutator leaves them as they are, and because they are all white, the collector is prevented from shading them. (Shading the first D-node by the collector would require the existence of an edge pointing to it from a grey node; in view of the absence of edges leading into D, this grey node would have to be a D-node, which is impossible.) Consequently, at the end of the marking phase, all D-nodes are still white garbage and will be appended to the free list during the subsequent appending phase. Hence, CC1 is also met. (End of proof.)

By keeping P1 invariant during the marking cycle, we obtained our coarse-grained solution. Encouraged by this success, we tried to keep P1 also invariant in a finer-grained solution --a solution in which the mutator's action M1 was split into two atomic operations: one for redirecting the edge and one for shading the new target-- . In order to keep P1 invariant, the mutator had to shade the future target first, and then redirect the edge towards the node just shaded. This finer-grained solution --although presented in

a way sufficiently convincing to fool ourselves-- contained the following bug, discovered by N.Stenning and M.Woodger [5].

Consider the following sequence of events:

1) prior to introducing an edge from node A to node B , the mutator shades node B (and goes to sleep)

2) the collector goes through a complete marking phase, followed by an appending phase (node B is now white, i.e. the mutator's shading has been undone! We further note that there is no garbage)

3) the collector goes through part of the next marking phase (and then goes to sleep), during which it so happens that node A is made black and node B is left white

4) the mutator (wakes up and) introduces without making garbage the edge from A to B (P1 is now violated)

5) the mutator removes all other ingoing edges of B --the absence of garbage makes this possible-- and goes to sleep again (node B is now only reachable via the edge from A)

6) the collector completes its marking phase (node B has remained white)

7) the collector goes through its appending phase, during which the reachable node B is erroneously appended to the free list.

This ill-fated effort convinced us that 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. However, it still seemed reasonable to retain the notion of "grey" as "semi-marked", more precisely, as representing an unfulfilled marking obligation. This meant that we could use the same collector. However, we had to find a different coarse-grained mutator that we could use as a stepping stone to our ultimate fine-grained solution.

5. A new coarse-grained solution.

For our new coarse-grained solution, we had to replace P1 by a weaker relation. (It was replaced by P2 and P3 , defined below.) In our first solution, we had made essential use of the fact that during the marking cycle, the validity of P1 allowed us to conclude that the existence of a white reachable node implied the existence of a grey node. (It even implied the

existence of a grey reachable node, but the reachability of the grey node was not essential.) For our new solution we needed a weaker relation P2 , from which the same conclusion could be drawn. We defined a "propagation path" to be one consisting solely of edges with white targets, and starting from a grey node, and chose the following relation

P2: "for each white reachable node, there exists a propagation path leading to it" .

Note 4. The grey node of the propagetion path is not necessarily reachable. (End of note 4.)

Corollary 1. If each root is grey or black, the absence of edges from black to white clearly implies that relation P2 is satisfied. In particular, P2 is true at the beginning of the marking cycle, because all roots have been shaded and there are no black nodes. (End of corollary 1.)

In proving the correctness of our first solution, the invariance of P1 was only needed to prove that, during the marking cycle, the absence of grey nodes implies that all white nodes are garbage. We can clearly use P2 instead of P1 to draw the same conclusion. Therefore, to prove the correctness of our new solution, we need only prove that both the mutator and the marking collector leave P2 invariant. However, P2 turned out to be too weak a relation from which to conclude that its truth will not be destroyed. To keep P2 invariant, we had to restrict the existence of black-to-white edges by the following further relation --analogous to P1 , but weaker--

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

Corollary 2. In the absence of black nodes, P3 is trivially satisfied. Hence it holds at the beginning of the marking cycle. (End of corollary 2.)

By corollaries 1 and 2 , P2 and P3 is true at the beginning of the marking cycle. To show that the marking cycle of our coarse-grained collector leaves P2 and P3 invariant, we must show that the atomic operation C1 cannot destroy its truth. Shading a node can cause neither P2 nor P3 to become

false; shading the successors of a node implies that its outgoing edges are no longer part of any propagation path, so making that node black immediately afterwards does not make P2 false either. Moreover, since its successors have just been shaded, making the node black does not introduce a black-to-white edge, and, hence, cannot make P3 false either. Combining these results we conclude that in its marking cycle the collector leaves P2 and P3 invariant.

All that remains to be done now in order to construct a correct coarse-grained solution is to define a mutator operation that leaves P2 and P3 invariant. When the mutator redefines an outgoing edge of a black node, it may direct it towards a white node. This new black-to-white edge is the one permitted by P3 . We must prevent, however, the previously redirected edge from also being a black-to-white edge, and we therefore consider for our coarse-grained mutator the following atomic operation:

M2: < shade the target of the previously redirected edge, and redirect an outgoing edge of a reachable node towards a reachable node > .

Note 5. For the very first time that the mutator redirects an edge, we can assume that (for lack of a previously redirected edge) either the shading will be suppressed or else an arbitrary reachable node will be shaded. The choice does not matter for the sequel. (End of note 5.)

Action M2 has been carefully chosen in such a way that it leaves P3 invariant. We now prove that it leaves the stronger relation P2 and P3 invariant as well, thereby showing that the new mutator with our original collector gives a correct solution.

Proof. The action M2 cannot introduce new reachable nodes. Hence, every white node which is reachable after the operation had a propagation path leading to it before the operation. If the node whose successor is redefined is black, then its outgoing edge was not part of any propagation path, so the edges of the old propagation paths will be sufficient to provide the propagation paths needed to maintain P2 . (We may not need all of them because of the shading operation, and because some white reachable nodes may have been made unreachable.) If the node whose successor is redefined was white or grey to start with, then the net result of action M2 will be a graph without edges from a black node to

a white one: if one existed, then its target has now been shaded, and no new one has been introduced since the source of the new edge is not black. The roots must still be grey or black, so by corollary 1 , P2 still holds. (End of proof.)

6. A fine-grained solution.

To complete our task, we now use the coarse-grained solution of section 5 as a stepping stone to a fine-grained one. For our fine-grained mutator, M2 is split up into the following succession of atomic operations:

M2.1: < shade the target of the previously redirected edge > ;

M2.2: < redirect an outgoing edge of a reachable node towards a reachable node >

In the collector, we break open C1 as the sequence of five atomic operations (m1 and m2 being local variables of the collector):

C1.1: < m1:= number of the left-hand successor of node nr.i > ;

C1.2: < shade node nr.m1 > ;

C1.3: < m2:= number of the right-hand successor of node nr.i > ;

C1.4: < shade node nr.m2 > ;

C1.5: < make node nr.i black >

We first observe that the collector's action of shading a node commutes with any number of mutator actions M2.1 and M2.2 ; without loss of generality we can, therefore, continue our discussion as if the four atomic operations C1.1 through C1.4 were replaced by a succession of the following two atomic operations:

C1.1a: < shade the left-hand successor of node nr.i > ;

C1.3a: < shade the right-hand successor of node nr.i >

Examining the proof for our coarse-grained solution, it is clear that in order to prove the correctness of this fine-grained one, it suffices to prove that P2 and P3 is still invariant during the (fine-grained) marking cycle. We shall prove the invariance of P2 and P3 by proving the invariance of a stronger relation.

For the purpose of its definition, we first introduce the notion of so-called "C-edges". Loosely formulated, C-edges are the edges the sources of which have been detected as grey by the collector's marking cycle. More precisely, the set of C-edges is empty at the beginning of the marking cycle, and the actions C1.1a and C1.3a add to it the left-hand and right-hand outgoing edge of node nr.i, respectively. Note that the formulation has been chosen so as to make it clear that, from C1.1a onwards, being a C-edge is a property of the left-hand outgoing edge of node nr.i, independent of the node it points to. In particular, being a C-edge is invariant with respect to redirection of that edge by the mutator.

Note 6. We only define the set of C-edges for our benefit. The set is not explicitly updated, although the collector could easily do so. In the jargon, the term "ghost variable" is sometimes used for such an entity. (End of note 6.)

The strengthened versions of P2 and P3 can now be formulated as follows:

P2a: "every root is grey or black, and for each white reachable node there exists a propagation path leading to it, containing no C-edges"

P3a: "there exists at most one edge "E" satisfying
pr: "(E is a black-to-white edge) or
(E is a C-edge with a white target)" ;
the existence of such an E satisfying pr implies that the mutator is between action M2.2 and the subsequent action M2.1, and that E is identical with the edge most recently redirected by the mutator" .

We now prove that P2a and P3a holds when the collector executes its marking cycle.

Proof. We first observe that P2a holds at the beginning of the marking cycle, thanks to corollary 1 and the fact that the set of C-edges is then empty. As there are neither C-edges nor black nodes at the beginning of the marking cycle, there is no edge E satisfying pr , so at the beginning of the marking cycle P3a holds as well.

We further make the general remark that none of the operations M2.1, M2.2, C1.1a, C1.3a, and C1.5 introduces new white reachable nodes. Consequently,

when proving the invariance of P2a under these operations, it suffices to show that the existence, before the operation, of a propagation path without C-edges, leading to a reachable node that is white before and after the operation, implies the existence afterwards of such a propagation path leading to that node.

The invariance of P2a and P3a with respect to the three shading operations M2.1 , C1.1a , and C1.3a can be dealt with simultaneously. Propagation paths leading to a reachable node that is white both before and after the shading operation are either left intact or are shortened by it. If these propagation paths did not contain C-edges before the shading operation, then they won't do so afterwards. The shading operations of the collector do create new C-edges, but these are C-edges with grey or black targets, and, therefore, cannot belong to any propagation path. This proves the invariance of P2a . Relation P3a is invariant as well, because the collector's shading acts introduce neither a black-to-white edge, nor a C-edge with a white target, and operation M2.1 only removes the edge E if it did exist.

Action C1.5 leaves P2a invariant: because the outgoing edges of the grey node nr.i are C-edges, they don't belong to existing propagation paths without C-edges, and hence making that node black leaves the existence of such paths unaffected. Action C1.5 also leaves P3a invariant, as it introduces no new solutions E of pr : it may introduce a black-to-white edge, but then that edge was already a C-edge with a white target.

Action M2.2 leaves P3a invariant because, if P3a held before M2.2 , no edge E satisfying pr existed, and the redirection can create at most one such edge. We finally prove the invariance of P2a under M2.2 . If the edge to be redirected is a C-edge or if its source is black, it does not belong to a propagation path without C-edges. Since, furthermore, M2.2 does not create C-edges, the existence of such paths remains in this case unaffected. In the other case --i.e. if the edge to be redirected is not a C-edge and has a white or grey source-- the already established invariance of P3a implies after M2.2 the absence of black-to-white edges and the absence of C-edges with a white target. In view of Corollary 1, these absences imply for all white reacheble nodes the existence of propagation paths without C-edges. (End of proof.)

In retrospect.

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. The reasoning we have used contains most of the ideas needed for a formal proof in the style of [3] or [4]. Because the inclusion of such a proof would result in a paper, possibly tedious, but in any case very different from what we intended to write this time, we have confined ourselves to our informal justification (which we do not regard as an adequate substitute for a formal correctness proof). Whether our stepwise approach 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.

History and acknowledgements. (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 the 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 fifth author, 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 several cross-Atlantic iterations. The active and inspiring interest shown by David Gries is mentioned

in gratitude. As with each new version of the manuscript the proofs became simpler, we also express our indebtedness to R.Stockton Gaines whose comments on an earlier version caused two further iterations.

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, Proving the Correctness of Multiprocess Programs. IEEE Trans. on Software Engineering SE-3, 2 (Mar. 1977), 125-143.

4. Gries, David, An Exercise in Proving Parallel Programs Correct. (Submitted to the Comm. ACM.)

5. Woodger, M., Private Communications.

Glossary of names.

Correctness criteria:

CC1: Every garbage node is eventually appended to the free list. More precisely, every garbage node present at the beginning of an appending phase will have been appended by the end of the next appending phase.

CC2: Appending a garbage node to the free list is the collector's only modification of (the shape of) the data structure.

Atomic operations of the mutator:

M1: < redirect an outgoing edge of a reachable node towards an already reachable one, and shade the new target > .

M2: < shade the target of the previously redirected edge, and redirect an outgoing edge of a reachable node towards a reachable node > .

M2.1: < shade the target of the previously redirected edge > .

M2.2: < redirect an outgoing edge of a reachable node towards a reachable node > .

Atomic operations of the collector:

C1: < shade the successors of node nr.i and make node nr.i black > .

C1.1: < m1:= number of the left-hand successor of node nr.i > .

C1.2: < shade node nr.m1 > .

C1.3: < m2:= number of the right-hand successor of node nr.i > .

C1.4: < shade node nr.m2 > .

C1.5: < make node nr.i black > .

C1.1a:< shade the left-hand successor of node nr.i > .

C1.3a:< shade the right-hand successor of node nr.i > .

Invariant relations:

P1: No edge points from a black node to a white one.

P2: For each white reachable node, there exists a propagation path leading to it.

P3: Only the last edge placed by the mutator may lead from a black node to a white one.

P2a: Every root is grey or black, and for each white reachable node there exists a propagation path leading to it, containing no C-edges.

P3a: There exists at most one edge "E" satisfying

pr: "(E is a black-to-white edge) or
(E is a C-edge with a white target)" ;

the existence of such an E satisfying pr implies that the mutator is between action M2.2 and the subsequent action M2.1, and that E is identical with the edge most recently redirected by the mutator.