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 ***)

Note: This copy has been made before the last four authors had had the opportunity to express their agreement.

*) 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. A technique is presented which allows nearly all of the garbage detection and collection activity to be performed by an additional processor, operating concurrently with the processor(s) carrying out the computations proper. Exclusion and synchronization contraints between the processors have been minimized.

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

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

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

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 emerging 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. Whether the following solution is of any economic significance, is a question beyond the scope of this article.

We tackled the problem as it presents itself in the traditional implementation environment for pure LISP (and shall describe our solution in the for that environment usual terminology, leaving the natural generalizations to the reader). 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. (Either of them or both may be missing.) At any moment in time all the nodes of the graph must be "reachable" (via a directed path along the directed edges) from one or more fixed nodes --called "the roots"-- with a constant place in memory. The storage allocated to each node is constant in time and equal in size, viz. sufficient to accommodate two pointers --one for each possible outgoing edge-- pointing to the node's immediate successors (if any: a missing edge is coded by means of a special pointer-value that has been reserved for that purpose). Given (the address of) a node, finding (the address of) its left- or right-hand successor node can be regarded in this representation as an atomic, primitive action; finding its predecessor nodes, however, would imply a search through memory.

For a reachable node an outgoing edge may be deleted, changed or added. Note that deletion and change of an outgoing edge may turn a number of formerly reachable nodes into unreachable ones: they become what is called "garbage". Changing or adding an edge may direct the new edge towards a target node that was already reachable or towards a new node that has to be added to the data structure; such a new node --which upon creation has no outgoing edges-- is taken from the so-called "free list", i.e. a linked list of node (locations) that are currently not used for storing a node from the data structure. 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 central processor is devoted to garbage collection. Starting from the roots all reachable nodes are marked; upon completion of this marking phase all unmarked nodes that are not in the free list are garbage and are appended to the free list, after which operation the computation proper is resumed. The minor disadvantage of this arrangement is the central processor time spent on the collection of the garbage; its major disadvantage is the unpredictability of these garbage collecting interludes, which makes it hard to design such a system 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 on a more continuous basis, 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. We have imposed upon our solution a few constraints. The (microscopic) interference between collector and mutator should be minimal --i.e. no highly frequent mutual exclusion of elaborate activities, as this would defy our aim of concurrent activity--, the overhead on the activity of the mutator (as required for the cooperation) should be kept as small as possible, and, finally, the ongoing activity of the mutator should not impair the collector's ability to identify garbage as such as soon as logically possible. (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 collected some garbage. 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.)

A counterexample taught us that the goal "no overhead for the mutator" is unattainable. Suppose that 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 with respect to C repeatedly the following sequence of operations:

1) making an outgoing edge from B to point to C
2) deleting the edge from A to C
3) making an outgoing edge from A to 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 never needs to discover that C is reachable as well: while A is observed by the collector, C may only be reachable via B and the other way round. Therefore, when changing or adding an edge, the mutator may have to mark in some way the target node of the new edge.

When we start with all nodes white, and, furthermore, the combined activity of collector and mutator can ensure that eventually all reachable nodes become black, then all white nodes that are not already on the free list can be identified as garbage. For each repetitive process --and the marking process certainly is one-- we have always two concerns: 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 suggest (fairly obviously)

A) each node will only darken monotonically.

For the invariant relation during the marking phase --a relation which must be satisfied both before and after marking-- we propose (perhaps less obviously, but not unnaturally)

B) no edge will ever point from a black node to a white one.

Additional action is required from the mutator when it is about to place an edge from a black node to a white node: just placing it would cause a violation of requirement B . Requirement A tells us that the black source node of the new edge has to remain black, and, therefore, requirement B implies 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 requirement B between the target node and its immediate successors. For that reason grey is introduced as an intermediate colour and the only overhead on the activity of the mutator will be --irrespective of the colour of the source node of the new edge--

"before placing a new edge, the mutator makes its new target at least grey".

Note 1. Making a node "at least grey" turns a white node into a grey one and leaves the colour of an initially grey or black node unchanged. It is assumed to be available as an atomic, primitive action. (End of Note 1.)

When all the roots have been darkened, requirement B is satisfied and there are no grey nodes, we can conclude that all reachable nodes are black and that, therefore, all white nodes are either garbage or on the free list. Two problems, however, present themselves: the collector has to distinguish between those two types of white nodes and, more seriously, as long as the nodes on the free list remain white, the mutator can create new grey nodes, and for the collector it will in general be very hard to reach the state without grey nodes and to detect that it has done so. The most elegant solution is to arrange matters in such a way that (due to collector activity) the nodes on the free list will eventually become black as well, as this solves both problems: when all reachable nodes are black, and all nodes on the free list are also black, further creation of grey nodes is excluded and all white nodes can be concluded to be garbage.

The simplest way to arrange that the nodes of the free list will be coloured as well is to introduce an additional root from which all the nodes on the free list (linked via edges!) are reachable. During the marking phase the collector makes no distinction between the free list and the data structure proper, and in the initial absence of black nodes it performs repeatedly the following program --assertion names being added within braces-- :

```
marking phase: {P0}
      make all roots at least grey {P1};
      repeat for a node  i  observed by the collector to be grey:
         make the left-hand successor of node  i  (if any) at least grey;
         make the right-hand successor of node  i  (if any) at least grey;
```

```
        make node i black
    until in a single scan in some order all nodes have been observed to
          be non-grey {P2};
collecting phase:
    process all nodes in some order, where "to process a node" means that
    a white node is appended at the free list and a black node is made
    white {P0}
```

Note 2. To make a node black is assumed to be an atomic, primitive action available to the collector. The final colour of a node made black by the collector and "simultaneously" made at least grey by the mutator will be black, i.e. as if the two operations had been performed in some order. (End of Note 2.)

Note 3. To make a node white is assumed to be an atomic, primitive action available to the collector. The final colour of a node made white by the collector and "simultaneously" made at least grey by the mutator will be either white or grey, i.e. as if the two operations had been performed in some order. (End of Note 3.)

Note 4. The mutator takes nodes away from one end of the free list, the collector appends nodes at the other end. The prevention of undesired interference between those two actions does not require mutual exclusion in time. (End of Note 4.)

The assertions and conclusions justifying the design are the following.

P0: There are no black nodes, and, therefore, requirement B (no edges from a black node to a white node) is satisfied.

P1: There are no edges from a black node to a white node and all roots are at least grey; as a result the existence of a white reachable node (those on the free list included) implies the existence of at least one grey node.

P2: There are no grey nodes --see Note 5-- and as P1 still holds, all white nodes are garbage and will remain white as far as mutator activity is concerned. This state is reached by the collector thanks to the monotonicity of the colour history of each node during the marking phase and the fact that the collector effectively darkens (at least node i !) as long as it finds a grey node.

P0: During the collecting phase requirements A and B may be violated. Requirement A is here inessential because the collecting phase does not depend on it for termination and requirement B is restored at completion of the collecting phase, because the mutator can't create a black node and all nodes that were black have been made white by the collector.

Note 5. The conclusion that there are no grey nodes when in a single scan all nodes have been observed by the collector to be non-grey requires some further justification, as one of the nodes observed to be non-grey --white, to be precise-- in the meantime could have been made grey by the mutator. Thanks to the continuing validity of P1 and the fact that the mutator leaves a grey node grey, however, the conclusion is justified. Suppose that during the scan the mutator has introduced a number of grey nodes. Consider the moment of the first time that the mutator did so; at that moment that node was a white reachable node (those on the free list included) and, therefore, it must have been so when the scan started. On account of P1 we can now conclude that when the scan started a grey node must have existed. As the mutator leaves grey nodes grey, the collector must have encountered that grey node during the scan. Therefore, if during the scan the collector has not encountered a grey node, the mutator cannot have created one either and the conclusion "there are no grey nodes" is therefore justified. (End of Note 5.)

Note 6. It is assumed that when the collector establishes (the address of) the left-hand successor of node i "simultaneously" with a redefinition of the left-hand successor of node i by the mutator, the result will be as if inspection and modification are done in some order: the collector will be directed either towards the old or towards the new left-hand successor. It is not difficult to verify that under all circumstances requirement B remains fulfilled. (End of Note 6.)

Finally: at the beginning of a collecting phase some garbage may be white and some may be black. At the end of that collecting phase all initially white garbage has been collected, all initially black garbage has been made white and will remain so during the next marking phase until it will be collected during the next collecting phase. As a result we can guarantee that no garbage, once created, will escape being collected.

History and acknowledgements. (As in this combination this is our first exercise in international and inter-company cooperation, some internal credit should be given as well.) After a 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 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 it as well was his suggestion-- and, on a suggestion made by Jack Mazola, by the first and the third author. The first and the fourth merged these embellishments and produced the version published above.

30th May 1975