A Correct and Useful Incremental Copying Garbage Collector

Martin Kero, Johan Nordlander, and Per Lindgren

Dept. of Computer Science and Electrical Engineering
Luleå University of Technology
MOTIVATION:

Build a Real-Time Garbage Collector!
Hasn’t that been done already???

It depends on what you mean by Real-Time Garbage Collection.
How To Get There
(redefining Real-Time GC)

• Design and understand the garbage collector algorithm.
• Understand the system to garbage collect.
• Schedule garbage collection in the system.
• Analyze schedulability of the system, including garbage collection.

In this paper we take the first step by formalizing a garbage collector algorithm; and reason about its basic properties.
Our Contribution

• We design a formal model of an incremental copying garbage collector.

• We reason about its basic properties:
  1. Correctness
  2. Usefulness.
The Garbage Collector Algorithm

- Cheney’s in-place breadth first traversal of gray objects.
- Read barrier similar to that of Baker (forward accesses to tospace).
- Write barrier in style of Steele (reverting black to gray).

- General objective:
  Defer as much work as possible to GC time!
The Model

• A heap $H$ is a finite mapping from addresses to nodes:

$$H ::= \{x_1 \mapsto o_1, \ldots, x_n \mapsto o_n\}$$

• A node can be either a sequence $V$ of values (enclosed by angle brackets) or a forwarding node:

$$o ::= \langle V \rangle \mid \bullet x$$
The Model

- A value is either an address or an integer:

\[ v ::= x \mid n \]

- We reserve one address name for the root of the heap (r).
$H = \{ x \mapsto \langle 1, 2, 3, z, 4, y \rangle, y \mapsto \langle 2, 5, 7, z \rangle, z \mapsto \langle 1, 2 \rangle, r \mapsto \langle x, y, z \rangle \}$
Bookkeeping

- We divide $H$ into three subheaps (white, gray, and black):

$$H = W \mid G \mid B$$

- The heap border (|) is just a special syntax for heap concatenation.

- Along the same line we use a symbol (up/down arrow) to keep track of the current position of the scan pointer.
The Algorithm

- Labeled Transition System (LTS).
- Internal transitions are garbage collector increments.
  \texttt{START, DONE, SCANSTART, SCANINT, SCANAddr, COPYSTART} ...
- Labeled transitions are mutator actions (read, write, and allocate).
  \texttt{r(p=n), w(p=n), w(p=q), a(p)}.
The Algorithm

- A garbage collection cycle begins with a START transition and ends with a DONE transition:

\[ H_0 \xrightarrow{\text{START}} H_1 \xrightarrow{} \ldots \xrightarrow{} H_{n-1} \xrightarrow{\text{DONE}} H_n \]
Garbage Collector Transitions – Examples

\[
\begin{align*}
0 &\mid G, r \mapsto \langle V \rangle \mid 0 & \text{START} & G &\mid r \mapsto \langle V \rangle \mid 0 \\
W &\mid G, x \mapsto \langle V \rangle \mid B & \text{SCANSTART} & W &\mid G, x \mapsto \langle \downarrow V \rangle \mid B \\
W &\mid G, x \mapsto \langle V \downarrow \rangle \mid B & \text{SCANDONE} & W &\mid G \mid x \mapsto \langle V \rangle, B \\
W &\mid 0 \mid B & \text{DONE} & 0 &\mid B \mid 0
\end{align*}
\]
Garbage Collector Transitions – Examples 2

\[
\begin{align*}
W, y & \mapsto \langle U \rangle, W' \mid G, x \mapsto \langle V \downarrow y, V' \rangle \mid B \\
\text{COPYSTART} \\
W, y & \mapsto \langle U \rangle, W' \mid z \mapsto \langle \rangle, G, x \mapsto \langle V \uparrow z y, V' \rangle \mid B \\
\text{FORWARD} \\
W, y & \mapsto \bullet z, W' \mid G, x \mapsto \langle V \downarrow y, V' \rangle \mid B \\
\end{align*}
\]
Garbage Collector Transitions – Examples 3

\[ W, y \mapsto \langle U, u, U' \rangle, W' \mid G, z \mapsto \langle U \rangle, G', x \mapsto \langle V \uparrow_z y, V' \rangle \mid B \]

COPY WORD

\[ W, y \mapsto \langle U, u, U' \rangle, W' \mid G, z \mapsto \langle U, u \rangle, G', x \mapsto \langle V \uparrow_z y, V' \rangle \mid B \]

equal

COPY DONE

\[ W, y \mapsto \langle U \rangle, W' \mid G, z \mapsto \langle U \rangle, G', x \mapsto \langle V \uparrow_z y, V' \rangle \mid B \]

\[ W, y \mapsto \langle U \rangle, W' \mid G, z \mapsto \langle U \rangle, G', x \mapsto \langle V \uparrow_z y, V' \rangle \mid B \]
Mutations

• Since we use a copying scheme – we would like to abstract away actual locations.

• We do this by defining mutator transitions based on the notion of paths.

• If the mutator can access a node, there exists at least one path from the root that leads to that node.
Mutator Transitions – Paths

- A path is a sequence of indexes.
- Example:

\[ H = \{ r \mapsto \langle x, y \rangle, x \mapsto \langle 3, z \rangle, y \mapsto \langle z, 7 \rangle, z \mapsto \langle 1, 2 \rangle \} \]
Correctness of the Algorithm

- Preserves well-formedness.
- Terminates.
- Preserves the meaning of the heap.
Well-formedness

1. Tri-color invariant holds (i.e. no black to white pointers).

2. Forwarding addresses never point to white nodes.

3. The root is never white.

4. Values to the left of (behind) the scan pointer are never addresses to white nodes.

5. While copying, the content of the new copy is always a prefix of (or equal to) the content of the old copy.

- LEMMA: If $H$ is well-formed and $H \rightarrow H'$ then $H'$ is also well-formed.
Termination

• THEOREM: For every well-formed heap within a garbage collection cycle (i.e. the START transition has been taken but not the DONE transition), a state where the DONE transition can be taken may be reached after a finite number of garbage collection transitions.

\[ H_0 \xrightarrow{\text{START}} H_1 \xrightarrow{} \ldots \xrightarrow{} H_{n-1} \xrightarrow{\text{DONE}} H_n \]

\[ H'_1 \quad \ldots \quad H'_{n-1} \]

\[ \forall H'_i, 1 \leq i \leq n-1 . H'_i \xrightarrow{\text{DONE}} H''_i \]
Preservation

• Equivalence: what makes two heaps equivalent?
• Read equivalence (i.e. a read has the same result in both heaps).
Example – Read Equivalence

H:

r:

x: 3

H’:

r:

y: 3

Read equivalent
Example – Read Equivalence

But they are different now. A mutation may cause them to not be read equivalent any more.
Preservation

• Equivalence: what makes two heaps equivalent?
  • Read equivalence (i.e. a read has the same result in both heaps).

  NOT enough!

• NEW ATTEMPT – Structural equivalence:
  1. Read equivalent.
  2. Contains the same joining paths.
Structural Equivalence

- Joining paths:
  - The predicate join:

    \[ \text{join}(H, r, p1, p2) \text{ is true if, starting at } r \text{ in } H, \text{ both } p1 \text{ and } p2 \text{ lead to the same address} \]

- Example:

  ![Diagram showing the joining of paths](image)

  \[ \text{join}(H, r, 1:2, 2:1) \text{ is True since both } 1:2 \text{ and } 2:1 \text{ end up at } z. \]
Example Revisited

H:  
\[ r: \]
\[ x: 3 \]

H':  
\[ r: \]
\[ y: 3 \]
\[ z: 3 \]

since \( \text{join}(H, r, 1, 1:2) \) is True but \( \text{join}(H', r, 1, 1:2) \) is False
Preservation

• LEMMA:
  If two heaps are structurally equivalent, and one of them can take a labeled transition, then the other one can take exactly the same transition, and the resulting heaps are also structurally equivalent.

\[
\begin{align*}
  H_1 & \xrightarrow{l} H_1' \\
  \| & \| \\
  H_2 & \xrightarrow{l} H_2'
\end{align*}
\]
Is Structural Equivalence Strong Enough?

- We can show that structural equivalence is preserved by each garbage collection transition.
- But does it hold for infinite sequences of transitions?
  - The $\pi$-calculus offers a proof technique for that:
    - Show that structural equivalence is a weak bisimulation.
Weak Bisimulation

• A relation $S$ on process terms is a weak simulation if:

$$
P \xrightarrow{\tau} P' \quad \quad P \xrightarrow{l} P'

S \quad \quad S

S \quad \quad S

Q \xrightarrow{\tau} Q' \quad \quad Q \xrightarrow{\tau} Q'

S \quad \quad S

S \quad \quad S

• $S$ is a weak bisimulation if both $S$ and its converse are weak simulations.

• $P$ and $Q$ are weakly bisimilar ($P \approx Q$) if there exists a weak bisimulation $S$ such that $P \mathrel{S} Q$. 
Bisimulating Garbage Collection

- We define two process terms; one with garbage collection, and one without.

\[ P_{GC}(H) \overset{\text{def}}{=} \sum_{H \xrightarrow{l} H'} l.P_{GC}(H') + \sum_{H \xrightarrow{\tau} H''} \tau.P_{GC}(H'') \]

\[ P(H) \overset{\text{def}}{=} \sum_{H \xrightarrow{l} H'} l.P(H') \]

- **THEOREM:**
  If \( H \) and \( H' \) are well-formed and structurally equivalent, then \( P_{GC}(H) \approx P(H') \).
Usefulness

- **LEMMA:**
  A dead node can never become live again.

- **THEOREM:**
  If a node has been copied, the address of its corresponding forwarding node is not dead in the original heap (i.e. when the garbage collector started).

- **COROLLARY:**
  If $H \rightarrow^* H'$ is a GC cycle then,
  $\text{size}(H') \leq \text{size}(\text{live}(H) \cup \text{new allocations})$
Our Contribution

• A formal model of an incremental copying GC.

• We have shown:

1. Correctness:
   ★ Well-formedness and Termination
   ★ We defined structural equivalence and have shown that it is indeed a weak bisimulation.

2. Usefulness:
   The size of the heap after a GC cycle is bounded by the size of the live nodes when the GC started + new allocations made during the GC cycle.
Questions and Comments