Proving Correctness of a Garbage Collector

via Local Reasoning

Lars Birkedal birkedal@itu.dk, Noah Torp-Smith noah@itu.dk

The IT University of Copenhagen

Joint work with John C. Reynolds, Carnegie Mellon University
Overview and Motivation

- Proved garbage collector correct, using (slight extension of) spatial logic [O’Hearn, Reynolds, et. al.]

- Has been considered a challenging problem, both in realm of spatial logics and in related realm of typed assembly language & proof carrying code

- Test of local reasoning / spatial logics: a non-toy example
Setup

- User Language (think: Java / ML / Scheme / . . . )
  - stack-allocated mutable variables
  - memory-allocation only through calls to cons
  - no pointer arithmetic (Pointer ∩ Int = ∅)

- Impl. Language (think: C / Assm / . . . )
  - lang. in which runtime system, in particular G.C., is impl.
    - with pointer arithmetic (Pointer ⊆ Location ⊆ Int = ∅)
  - heaps map locations to integers
  - all user level values represented as integers (encoding to allow G.C. to distinguish between pointers and non-pointers)
Correctness of Tracing G.C.

- If $GC', s, h \rightsquigarrow s', h'$ the requirement is that $(s', h')$ is a garbage collected version of $(s, h)$:
  - there is an isomorphism between the locations of the reachable cells in the two states (the iso because cells can be moved around during garbage collection)
- more precisely…
Correctness of Tracing G.C. (2)

- $(s, h)$ is a garbage collected version of $(s', h')$, if there is a heap isomorphism $\varphi : \text{prune}(s, h) \cong \text{prune}(s', h')$.

- A weak heap isomorphism $\varphi : (s', h') \cong (s, h)$ is a bijection $\varphi : \text{dom}(h') \cong \text{dom}(h)$ such that for all $p \in \text{dom}(h')$,
  \[
  h(\varphi(p)) = \varphi^*(h'(p)),
  \]
  where $\varphi^*$ is the extension of $\varphi$ to all integers with the identity on nonpointers. If also $\varphi(s'(\text{root})) = s(\text{root})$, we call $\varphi$ a heap isomorphism.
Cheney’s Copying Collector

- Assumes 2 contiguous “semi-spaces” of equal size,

\[ \text{OLD} = [\text{startOld}, \text{endOld}] \quad \text{and} \quad \text{NEW} = [\text{startNew}, \text{endNew}] \]

- \( s(\text{root}) \in \text{OLD} \)

- \( \text{ALIVE} = \{ p \mid p \text{ is reachable} \} \)

- Copies ALIVE to NEW and resumes allocation there
Example
Example (2)
Sets of cells

- $\text{FORW} = \text{forwarded locations in alive}$
- $\text{UNFORW} = \text{non-forwarded locations in alive}$
- $\text{FIN} = [\text{startNew, scan}]$
- $\text{UNFIN} = [\text{scan, free}]$
- $\text{FREE} = [\text{free, endNew}]$
Extension of Separation Logic

To formalize the partitioning of cells, we extend the term language with *finite sets of pointers*:
Extension of Separation Logic

To formalize the partitioning of cells, we extend the term language with *finite sets of pointers*:

\[ m^{fs} ::= \emptyset^{fs} | x^{fs} | \{ e^{\text{int}} \} | m^{fs} \oplus e^{\text{int}} | m^{fs} \ominus e^{\text{int}} | Itv(t^{\text{int}}, t^{\text{int}}) | m^{fs} \cup m^{fs} \]
Extension of Separation Logic

To formalize the partitioning of cells, we extend the term language with *finite sets of pointers*:

\[
m^{fs} ::= \emptyset^{fs} | x^{fs} | \{e^{int}\} | m^{fs} \oplus e^{int} | m^{fs} \ominus e^{int} \\
| Itv(t^{int}, t^{int}) | m^{fs} \cup m^{fs}
\]

We will also need *finite relations*:

\[
f^{frp} ::= \emptyset^{frp} | x^{frp} | f^{frp} \oplus (e^{int}, e^{int}) | f_1^{frp} \circ f_2^{frp} | f^T | f \odot g
\]
Extension of Separation Logic

To formalize the partitioning of cells, we extend the term language with *finite sets of pointers*:

\[
 m^{fs} ::= \emptyset^{fs} \mid x^{fs} \mid \{e^{int}\} \mid m^{fs} \oplus e^{int} \mid m^{fs} \ominus e^{int} \\
 \quad \mid \text{Itv}(t^{int}, t^{int}) \mid m^{fs} \cup m^{fs}
\]

We will also need *finite relations*:

\[
 f^{frp} ::= \emptyset^{frp} \mid x^{frp} \mid f^{frp} \oplus (e^{int}, e^{int}) \mid f^{frp} \circ f^{frp} \mid f^T \mid f \odot g
\]

Semantics for \( \odot \): extension with identity on non-pointers:

\[
\llbracket f^{frp} \odot g^{frp} \rrbracket = \{(p, n) \mid ((p, n) \in \llbracket g \rrbracket s \land n \notin \text{Ptr}) \lor \\
(\exists p' \in \text{Ptr}. (p, p') \in \llbracket g \rrbracket s \land (p', n) \in \llbracket f \rrbracket s)\}
\]
Extension of Separation Logic (2)

Some new assertion forms:

- *Iterated Separating Conjunction*:

\[ \forall_* p \in m. A(p) \]
Some new assertion forms:

- **Iterated Separating Conjunction:**

  \[ \forall_* p \in m. A(p) \]

- **Semantics:**

  \[ s, h \models \forall_* p \in m. A(p) \text{ iff } \]

  [\!m\!] s = \emptyset \text{ implies } s, h \models \text{emp}, \text{ and } \]

  [\!m\!] s = \{p_1, \ldots, p_k\} \text{ implies } \]

  s, h \models A(p_1) \ast \cdots \ast A(p_k) \]
The Proof

We have

- The sets mentioned before
- Relations head and tail that record the initial heap
- $\varphi$, a bijection,

$$\varphi : \text{FORW} \rightarrow \text{BUSY} = \text{FIN} \cup \text{UNFIN} = [\text{startNew}, \text{free}]$$

These are added to the program as auxiliary variables [Owicki, Gries], and will be part of the proof.
The Proof (2)

Analysis of each set:
The Proof (2)

Analysis of each set:

- UNFORW is not modified, so we can use head, tail.

\[
A_{Uf} \equiv \forall \_p \in \text{UNFORW}. \ ((\exists q. (p, q) \in \text{head} \land p \leftrightarrow q)^* \\
(\exists q'. (p, q') \in \text{tail} \land p + 4 \leftrightarrow q'))
\]
The Proof (2)

Analysis of each set:

- UNFORW is not modified, so we can use head, tail.

\[ A_{Uf} \equiv \forall \_p \in \text{UNFORW}. ((\exists q. (p, q) \in \text{head} \land p \leftrightarrow q)^\ast \\
(\exists q'. (p, q') \in \text{tail} \land p + 4 \leftrightarrow q')) \]

- FORW: First component points to cell determined by \( \varphi \):

\[ A_{FW} \equiv \forall \_p \in \text{FORW}. (\exists q. (p, q) \in \varphi \land p \leftrightarrow q, -) \]
The Proof (2)

Analysis of each set:

◆ UNFORW is not modified, so we can use head, tail.

\[ A_{Uf} \equiv \forall \*_p \in \text{UNFORW.} \ ((\exists q. (p, q) \in \text{head} \land p \mapsto q) * \ (\exists q'. (p, q') \in \text{tail} \land p + 4 \mapsto q')) \]

◆ FORW: First component points to cell determined by \( \varphi \):

\[ A_{Fw} \equiv \forall \*_p \in \text{FORW.} \ (\exists q. (p, q) \in \varphi \land p \mapsto q, -) \]

◆ FREE. Pointers here are in the domain of the heap:

\[ A_{Fr} \equiv \forall \*_p \in \text{FREE.} \ p \mapsto -, - \]
Analysis of each set, ct’d:

- **UNFIN**: Each cell is a copy of the cell in FORW that points to it:

\[
A_{Un} \equiv \forall p \in \text{UNFIN}. \left( (\exists q. (p, q) \in \text{head} \circ \varphi^T \land p \mapsto q) \ast \\
(\exists q'. (p, q') \in \text{tail} \circ \varphi^T \land p + 4 \mapsto q') \right)
\]
The Proof (3)

Analysis of each set, ct’d:

◆ **UNFIN**: Each cell is a copy of the cell in FORW that points to it:

\[
A_{Un} \equiv \forall_* p \in \text{UNFIN. } ((\exists q. (p, q) \in \text{head} \circ \varphi^T \wedge p \mapsto q)^* \\
(\exists q'. (p, q') \in \text{tail} \circ \varphi^T \wedge p + 4 \mapsto q'))
\]

◆ **FIN**: scanned cells in BUSY. Scanning means updating component to $\varphi$-value (but only if the component is a pointer). This is captured by $\odot$:

\[
A_{Fn} \equiv \forall_* p \in \text{UNFIN. } ((\exists q. (p, q) \in \varphi \odot (\text{head} \circ \varphi^T) \wedge p \mapsto q)^* \\
(\exists q'. (p, q') \in \varphi \odot (\text{tail} \circ \varphi^T) \wedge p + 4 \mapsto q'))
\]
The Proof (4)

The Precondition:

\[ \text{InitAss} \equiv \]
\[ \text{Ptr}(\text{startNew}) \land \text{Ptr}(\text{endNew}) \land \text{Ptr}(\text{root}) \land \text{Disjoint}(\text{OLD}, \text{NEW}) \land \]
\[ \text{SbSet}(\text{ALIVE}, \text{OLD}) \land \text{Reachable}(\text{head}, \text{tail}, \text{ALIVE}, \text{root}) \land \]
\[ \#\text{NEW} = \#\text{OLD} \land \text{PtrRg}(\text{head}, \text{ALIVE}) \land \text{PtrRg}(\text{tail}, \text{ALIVE}) \land \]
\[ \text{Tfun}(\text{head}, \text{ALIVE}) \land \text{Tfun}(\text{tail}, \text{ALIVE}) \land \]
\[ ((\forall_* p \in \text{ALIVE}. ((\exists q. (p, q) \in \text{head} \land p \leftrightarrow q)) \ast \]
\[ (\exists q. (p, q') \in \text{tail} \land p + 4 \leftrightarrow q') \ast) \ast \]
\[ (\forall_* p \in \text{NEW}. p \leftrightarrow -, -) \ast \text{T}) \]

The \text{T} deals with “unreachable” cells (they are framed out).
The Proof (4)

The Invariant:

\[ I \equiv \]
iso(\(\varphi\), FORW, BUSY) \& isUnion(FORW, UNFORW, ALIVE)\&
#ALIVE \leq #NEW \& root \in FORW \& scan \leq free \&
Disjoint(ALIVE, NEW) \& Ptr(free) \& Ptr(scan) \& Ptr(offset)\&
Ptr(maxFree) \& Reachable(head, tail, ALIVE, root)\&
PtrRg(head, ALIVE) \& PtrRg(tail, ALIVE)\&
Tfun(head, ALIVE) \& Tfun(tail, ALIVE)\&
(A_Uf \ast A_{Fw} \ast A_{Fn} \ast A_{Un} \ast A_{Fn}) \]
The most interesting part of the proof is when we copy a cell. We prove a local specification and use the Frame Rule: The local specification only mentions the “footprint” of the program fragment ($x$ is cell pointed to by $\text{scan}$):

$$\{(\exists q. (x, q) \in \text{head} \land x \mapsto q) \ast (\exists q'. (x, q') \in \text{tail} \land x + 4 \mapsto q')\ast (\text{scan} \mapsto -) \ast (\text{free} \mapsto -, -)\}$$

CopyCell

$$\{((x \mapsto \text{free}, -) \ast (\text{scan} \mapsto \text{free}) \ast (\text{free} \mapsto t_1, t_2)) \land (x, t_1) \in \text{head} \land (x, t_2) \in \text{tail}\}$$
Remarks about the Proof:

The proof of the specification is entirely "logical": always uses proof-rules, not "semantical arguments".

The proof that the invariant is strong enough to conclude that there is a heap isomorphism, is almost logical: We prove logically that, Recall equation for heap isomorphism.
The Proof (6)

Remarks about the Proof:

- The proof of the specification is entirely “logical”: always uses proof-rules, not “semantical arguments”.

The Proof (6)

Remarks about the Proof:

- The proof of the specification is entirely “logical”: always uses proof-rules, not “semantical arguments”.
- The proof that the invariant is strong enough to conclude that there is a heap isomorphism, is almost logical: We prove logically that,

\[
I \land \text{scan = free } \rightarrow (p \in \text{ALIVE} \land (p, q) \in \varphi \rightarrow (q \leftrightarrow r \leftrightarrow (p, r) \in \varphi \odot \text{head}))
\]

Recall equation for heap isos:

\[
h'(\varphi(p)) = \varphi^*(h(p))
\]
Conclusion and Future Work

- Formal proof of an algorithm that is used in practice
- Local reasoning “passed the test”
- Method of sets and relations is believed to be widely applicable, so further study is needed... also of higher-order separation logic
- A more precise formulation of the interface issues is needed.
- A technical report will be available soon.