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 $\mathbf{cons}$
  - no pointer arithmetic (Pointer $\cap$ Int $= \emptyset$)
- Impl. Language (think: C / Assm / . . .)
  - lang. in which runtime system, in particular G.C., is impl.
  - with pointer arithmetic (Pointer $\subseteq$ Location $\subseteq$ Int $= \emptyset$)
  - 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 $\varphi : prune(s, h) \cong prune(s', h')$.
- A *weak heap isomorphism* $\varphi : (s', h') \cong (s, h)$ is a bijection $\varphi : dom(h') \cong dom(h)$ such that for all $p \in dom(h')$,

$$h(\varphi(p)) = \varphi^*(h'(p)),$$

  where φ^* is the extension of φ to all integers with the identity on nonpointers. If also $\varphi(s'(\mathsf{root})) = s(\mathsf{root})$, we call φ a *heap isomorphism*.

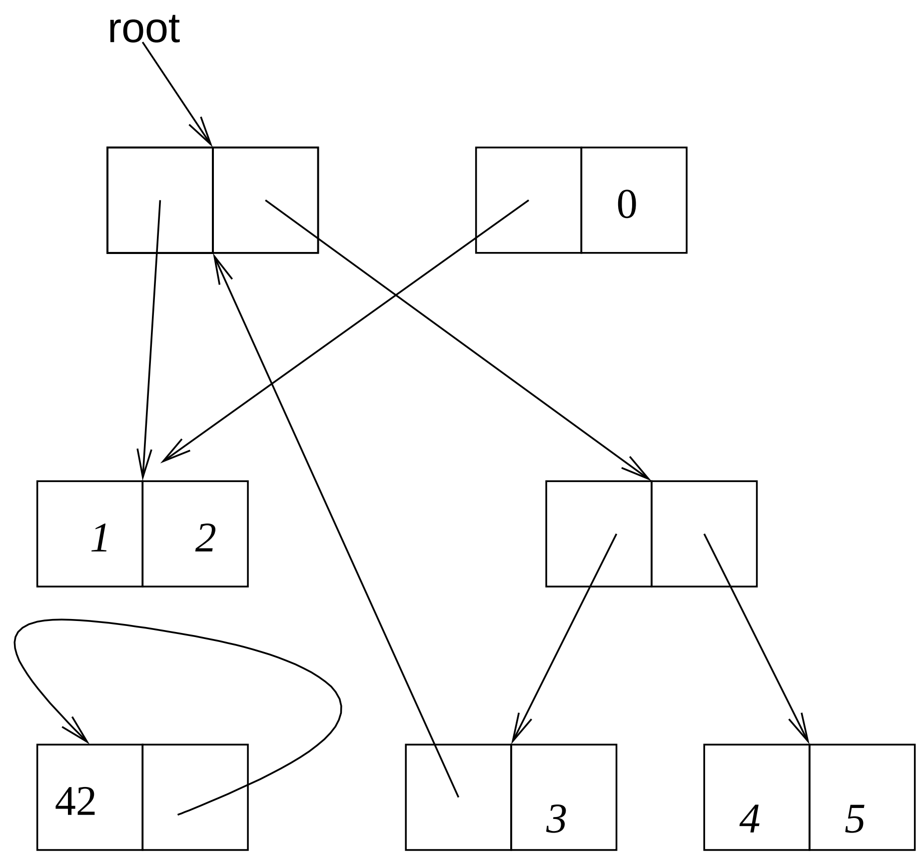
Cheney's Copying Collector

- Assumes 2 contiguous "semi-spaces" of equal size,

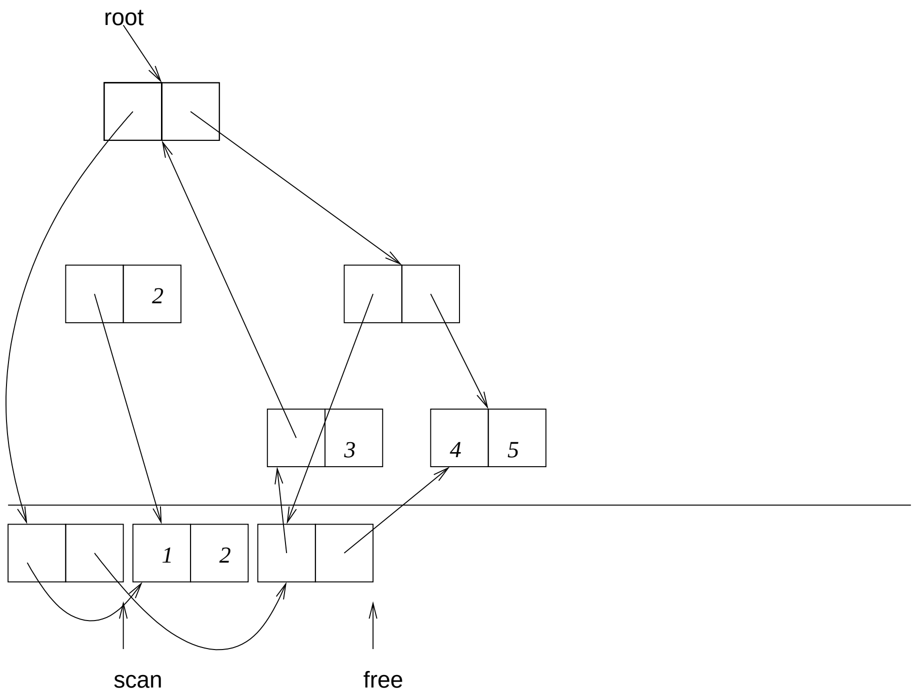
  $\mathsf{OLD} = [\mathsf{startOld}, \mathsf{endOld}[$ and $\mathsf{NEW} = [\mathsf{startNew}, \mathsf{endNew}[$

- $s(\mathsf{root}) \in \mathsf{OLD}$
- $\mathsf{ALIVE} = \{p \mid p \text{ is reachable}\}$
- Copies ALIVE to NEW and resumes allocation there

Example



Example (2)



Sets of cells

- $\mathsf{FORW} = $ forwarded locations in alive
- $\mathsf{UNFORW} = $ non-forwarded locations in alive
- $\mathsf{FIN} = [\mathsf{startNew}, \mathsf{scan}[$
- $\mathsf{UNFIN} = [\mathsf{scan}, \mathsf{free}[$
- $\mathsf{FREE} = [\mathsf{free}, \mathsf{endNew}[$

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*:

$$\begin{aligned} m^{\mathsf{fs}} \quad ::= \quad & \varnothing^{\mathsf{fs}} \mid x^{\mathsf{fs}} \mid \{e^{\mathsf{int}}\} \mid m^{\mathsf{fs}} \oplus e^{\mathsf{int}} \mid m^{\mathsf{fs}} \ominus e^{\mathsf{int}} \\ & \mid Itv(t^{\mathsf{int}}, t^{\mathsf{int}}) \mid m^{\mathsf{fs}} \cup m^{\mathsf{fs}} \end{aligned}$$

Extension of Separation Logic

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

$$\begin{aligned} m^{\mathsf{fs}} \quad ::= & \quad \varnothing^{\mathsf{fs}} \mid x^{\mathsf{fs}} \mid \{e^{\mathsf{int}}\} \mid m^{\mathsf{fs}} \oplus e^{\mathsf{int}} \mid m^{\mathsf{fs}} \ominus e^{\mathsf{int}} \\ & \mid Itv(t^{\mathsf{int}}, t^{\mathsf{int}}) \mid m^{\mathsf{fs}} \cup m^{\mathsf{fs}} \end{aligned}$$

We will also need *finite relations*:

$$f^{\mathsf{frp}} \quad ::= \quad \varnothing^{\mathsf{frp}} \mid x^{\mathsf{frp}} \mid f^{\mathsf{frp}} \oplus (e^{\mathsf{int}}, e^{\mathsf{int}}) \mid f_1^{\mathsf{frp}} \circ f_2^{\mathsf{frp}} \mid f^{\mathsf{T}} \mid f \circledcirc g$$

Extension of Separation Logic

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

$$\begin{array}{rcl} m^{\mathsf{fs}} & ::= & \varnothing^{\mathsf{fs}} \mid x^{\mathsf{fs}} \mid \{e^{\mathsf{int}}\} \mid m^{\mathsf{fs}} \oplus e^{\mathsf{int}} \mid m^{\mathsf{fs}} \ominus e^{\mathsf{int}} \\ & & \mid Itv(t^{\mathsf{int}}, t^{\mathsf{int}}) \mid m^{\mathsf{fs}} \cup m^{\mathsf{fs}} \end{array}$$

We will also need *finite relations*:

$$f^{\mathsf{frp}} \quad ::= \quad \varnothing^{\mathsf{frp}} \mid x^{\mathsf{frp}} \mid f^{\mathsf{frp}} \oplus (e^{\mathsf{int}}, e^{\mathsf{int}}) \mid f_1^{\mathsf{frp}} \circ f_2^{\mathsf{frp}} \mid f^{\mathsf{T}} \mid f \circledcirc g$$

Semantics for $\circledcirc$: extension with identity on non-pointers:

$$\begin{array}{rcl} [\![f^{\mathsf{frp}} \circledcirc g^{\mathsf{frp}}]\!] & = & \{(p,n) \mid ((p,n) \in [\![g]\!]s \wedge n \notin Ptr) \vee \\ & & (\exists p' \in Ptr.(p,p') \in [\![g]\!]s \wedge (p',n) \in [\![f]\!]s)\} \end{array}$$

Extension of Separation Logic (2)

Some new assertion forms:

- *Iterated Separating Conjunction*:

$$\forall_* p \in m.\ A(p)$$

Extension of Separation Logic (2)

Some new assertion forms:

- *Iterated Separating Conjunction*:

$$\forall_* p \in m.\ A(p)$$

- Semantics:

$$s, h \Vdash \forall_* p \in m.\ A(p) \text{ iff}$$
$$[\![m]\!]s = \varnothing \text{ implies } s, h \Vdash \mathsf{emp}, \text{ and}$$
$$[\![m]\!]s = \{p_1, \ldots, p_k\} \text{ implies}$$
$$s, h \Vdash A(p_1) * \cdots * A(p_k)$$

The Proof

We have

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

$$\varphi : \mathsf{FORW} \to \mathsf{BUSY} = \mathsf{FIN} \cup \mathsf{UNFIN} = [\mathsf{startNew}, \mathsf{free}[$$

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 $\mathsf{head}, \mathsf{tail}$.

$$A_{\mathsf{Uf}} \equiv \forall_* p \in \mathsf{UNFORW}.\ ((\exists q.(p, q) \in \mathsf{head} \wedge p \mapsto q) * (\exists q'.(p, q') \in \mathsf{tail} \wedge p + 4 \mapsto q'))$$

The Proof (2)

Analysis of each set:

- UNFORW is not modified, so we can use $\mathsf{head}, \mathsf{tail}$.

$$
\begin{aligned}
A_{\mathsf{Uf}} \quad \equiv \quad & \forall_* p \in \mathsf{UNFORW}.\ ((\exists q.(p,q) \in \mathsf{head} \wedge p \mapsto q) * \\
& (\exists q'.(p,q') \in \mathsf{tail} \wedge p+4 \mapsto q'))
\end{aligned}
$$

- FORW: First component points to cell determined by φ:

$$
A_{\mathsf{Fw}} \quad \equiv \quad \forall_* p \in \mathsf{FORW}.\ (\exists q.(p,q) \in \varphi \wedge p \mapsto q, -)
$$

The Proof (2)

Analysis of each set:

- UNFORW is not modified, so we can use $\mathsf{head}, \mathsf{tail}$.

$$A_{\mathsf{Uf}} \equiv \forall_* p \in \mathsf{UNFORW}.\ ((\exists q.(p,q) \in \mathsf{head} \wedge p \mapsto q) * (\exists q'.(p,q') \in \mathsf{tail} \wedge p+4 \mapsto q'))$$

- FORW: First component points to cell determined by φ:

$$A_{\mathsf{Fw}} \equiv \forall_* p \in \mathsf{FORW}.\ (\exists q.(p,q) \in \varphi \wedge p \mapsto q, -)$$

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

$$A_{\mathsf{Fr}} \equiv \forall_* p \in \mathsf{FREE}.\ p \mapsto -, -$$

The Proof (3)

Analysis of each set, ct'd:

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

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

The Proof (3)

Analysis of each set, ct'd:

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

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

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

$$A_{\mathsf{Fn}} \equiv \forall_* p \in \mathsf{UNFIN}.\ ((\exists q.(p,q) \in \varphi \circledcirc (\mathsf{head} \circ \varphi^{\mathsf{T}}) \wedge p \mapsto q) * (\exists q'.(p,q') \in \varphi \circledcirc (\mathsf{tail} \circ \varphi^{\mathsf{T}}) \wedge p+4 \mapsto q'))$$

The Proof (4)

The Precondition:

$$\begin{aligned}
&\mathsf{InitAss} \equiv\\
&\mathsf{Ptr}(\mathsf{startNew}) \wedge \mathsf{Ptr}(\mathsf{endNew}) \wedge \mathsf{Ptr}(\mathsf{root}) \wedge \mathsf{Disjoint}(\mathsf{OLD}, \mathsf{NEW})\wedge\\
&\mathsf{SbSet}(\mathsf{ALIVE}, \mathsf{OLD}) \wedge \mathsf{Reachable}(\mathsf{head}, \mathsf{tail}, \mathsf{ALIVE}, \mathsf{root})\wedge\\
&\#\mathsf{NEW} = \#\mathsf{OLD} \wedge \mathsf{PtrRg}(\mathsf{head}, \mathsf{ALIVE}) \wedge \mathsf{PtrRg}(\mathsf{tail}, \mathsf{ALIVE})\wedge\\
&\mathsf{Tfun}(\mathsf{head}, \mathsf{ALIVE}) \wedge \mathsf{Tfun}(\mathsf{tail}, \mathsf{ALIVE})\wedge\\
&((\forall_* p \in \mathsf{ALIVE}.\ ((\exists q.\ (p, q) \in \mathsf{head} \wedge p \mapsto q)*\\
&\quad (\exists q.\ (p, q') \in \mathsf{tail} \wedge p + 4 \mapsto q')))*\\
&(\forall_* p \in \mathsf{NEW}. p \mapsto -, -) * \mathsf{T})
\end{aligned}$$

The T deals with “unreachable” cells (they are framed out).

The Proof (4)

The Invariant:

$$
\begin{aligned}
&I \equiv \\
&\mathsf{iso}(\varphi, \mathsf{FORW}, \mathsf{BUSY}) \ \wedge \ \mathsf{isUnion}(\mathsf{FORW}, \mathsf{UNFORW}, \mathsf{ALIVE}) \wedge \\
&\#\mathsf{ALIVE} \leq \#\mathsf{NEW} \wedge \mathsf{root} \in \mathsf{FORW} \ \wedge \mathsf{scan} \leq \mathsf{free} \wedge \\
&\mathsf{Disjoint}(\mathsf{ALIVE}, \mathsf{NEW}) \wedge \mathsf{Ptr}(\mathsf{free}) \wedge \mathsf{Ptr}(\mathsf{scan}) \wedge \mathsf{Ptr}(\mathsf{offset}) \wedge \\
&\mathsf{Ptr}(\mathsf{maxFree}) \wedge \mathsf{Reachable}(\mathsf{head}, \mathsf{tail}, \mathsf{ALIVE}, \mathsf{root}) \wedge \\
&\mathsf{PtrRg}(\mathsf{head}, \mathsf{ALIVE}) \wedge \mathsf{PtrRg}(\mathsf{tail}, \mathsf{ALIVE}) \wedge \\
&\mathsf{Tfun}(\mathsf{head}, \mathsf{ALIVE}) \wedge \mathsf{Tfun}(\mathsf{tail}, \mathsf{ALIVE}) \wedge \\
&(\mathsf{A}_{\mathsf{Uf}} * \mathsf{A}_{\mathsf{Fw}} * \mathsf{A}_{\mathsf{Fn}} * \mathsf{A}_{\mathsf{Un}} * \mathsf{A}_{\mathsf{Fn}})
\end{aligned}
$$

The Proof (5)

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 scan):

$$\{(\exists q.\ (x, q) \in \mathsf{head} \wedge x \mapsto q) * (\exists q'.\ (x, q') \in \mathsf{tail} \wedge x + 4 \mapsto q') * (\mathsf{scan} \mapsto -) * (\mathsf{free} \mapsto -, -)\}$$

$$\mathsf{CopyCell}$$

$$\{((x \mapsto \mathsf{free}, -) * (\mathsf{scan} \mapsto \mathsf{free}) * (\mathsf{free} \mapsto t_1, t_2)) \wedge (x, t_1) \in \mathsf{head} \wedge (x, t_2) \in \mathsf{tail}\}$$

The Proof (6)

Remarks about the Proof:

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,

$$I \wedge \mathsf{scan} = \mathsf{free} \rightarrow (p \in \mathsf{ALIVE} \wedge (p, q) \in \varphi \rightarrow$$
$$(q \hookrightarrow r \leftrightarrow (p, r) \in \varphi \circledcirc \mathsf{head}))$$

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.