

---

# 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 ( $\text{Pointer} \cap \text{Int} = \emptyset$ )
- ◆ Impl. Language (think: C / Assm / ...)
  - lang. in which runtime system, in particular G.C., is impl.
  - with pointer arithmetic  
( $\text{Pointer} \subseteq \text{Location} \subseteq \text{Int} = \emptyset$ )
  - 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  $\varphi : \text{prune}(s, h) \cong \text{prune}(s', h')$ .
- ◆ 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,

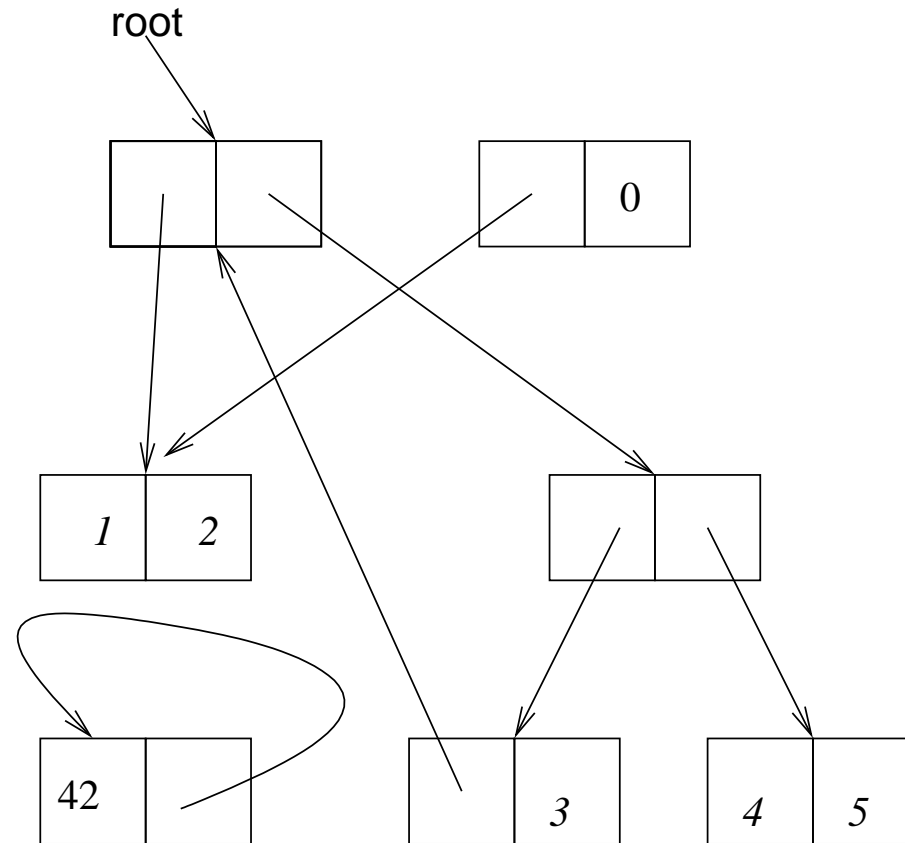
OLD = [startOld, endOld[    and    NEW = [startNew, endNew[

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



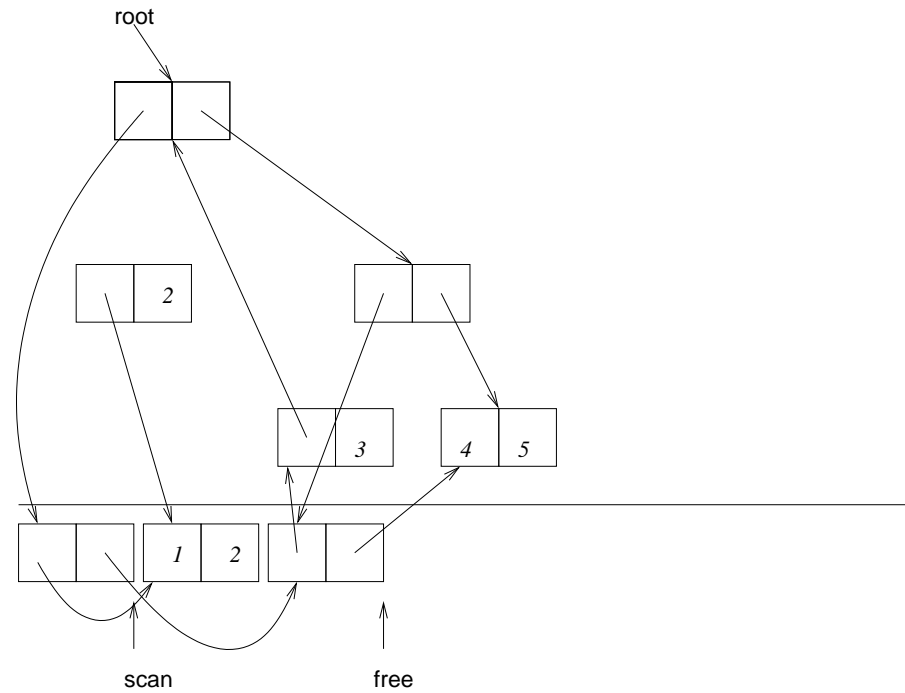
# Example

---



# Example (2)

---



# Sets of cells

---

- ◆ FORW = forwarded locations in alive
- ◆ UNFORW = non-forwarded locations in alive
- ◆ FIN = [startNew, scan[
- ◆ UNFIN = [scan, free[
- ◆ FREE = [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*:

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



# Extension of Separation Logic

---

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

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

We will also need *finite relations*:

$$f^{\text{frp}} ::= \emptyset^{\text{frp}} \mid x^{\text{frp}} \mid f^{\text{frp}} \oplus (e^{\text{int}}, e^{\text{int}}) \mid f_1^{\text{frp}} \circ f_2^{\text{frp}} \mid f^{\text{T}} \mid f \odot g$$



# Extension of Separation Logic

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

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

We will also need *finite relations*:

$$f^{\text{frp}} ::= \emptyset^{\text{frp}} \mid x^{\text{frp}} \mid f^{\text{frp}} \oplus (e^{\text{int}}, e^{\text{int}}) \mid f_1^{\text{frp}} \circ f_2^{\text{frp}} \mid f^{\text{T}} \mid f \odot g$$

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

$$\llbracket f^{\text{frp}} \odot g^{\text{frp}} \rrbracket = \{(p, n) \mid ((p, n) \in \llbracket g \rrbracket s \wedge n \notin \text{Ptr}) \vee \\ (\exists p' \in \text{Ptr}. (p, p') \in \llbracket g \rrbracket s \wedge (p', n) \in \llbracket f \rrbracket s)\}$$



# Extension of Separation Logic (2)

---

Some new assertion forms:

- ◆ *Iterated Separating Conjunction:*

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



# Extension of Separation Logic (2)

---

Some new assertion forms:

- ◆ *Iterated Separating Conjunction:*

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

- ◆ **Semantics:**

$s, h \Vdash \forall_* p \in m. A(p)$  iff

$\llbracket m \rrbracket s = \emptyset$  implies  $s, h \Vdash \text{emp}$ , and

$\llbracket m \rrbracket s = \{p_1, \dots, p_k\}$  implies

$s, h \Vdash A(p_1) * \dots * A(p_k)$



# 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.

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



# The Proof (2)

---

## Analysis of each set:

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

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

- ◆ FORW: First component points to cell determined by  $\varphi$ :

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



# The Proof (2)

---

## Analysis of each set:

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

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

- ◆ FORW: First component points to cell determined by  $\varphi$ :

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

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

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



# The Proof (3)

---

## Analysis of each set, ct'd:

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

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



# The Proof (3)

---

## Analysis of each set, ct'd:

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

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

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

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



# The Proof (4)

---

## The Precondition:

InitAss  $\equiv$

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

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



# The Proof (4)

---

## The Invariant:

$I \equiv$

$\text{iso}(\varphi, \text{FORW}, \text{BUSY}) \wedge \text{isUnion}(\text{FORW}, \text{UNFORW}, \text{ALIVE}) \wedge$   
 $\# \text{ALIVE} \leq \# \text{NEW} \wedge \text{root} \in \text{FORW} \wedge \text{scan} \leq \text{free} \wedge$   
 $\text{Disjoint}(\text{ALIVE}, \text{NEW}) \wedge \text{Ptr}(\text{free}) \wedge \text{Ptr}(\text{scan}) \wedge \text{Ptr}(\text{offset}) \wedge$   
 $\text{Ptr}(\text{maxFree}) \wedge \text{Reachable}(\text{head}, \text{tail}, \text{ALIVE}, \text{root}) \wedge$   
 $\text{PtrRg}(\text{head}, \text{ALIVE}) \wedge \text{PtrRg}(\text{tail}, \text{ALIVE}) \wedge$   
 $\text{Tfun}(\text{head}, \text{ALIVE}) \wedge \text{Tfun}(\text{tail}, \text{ALIVE}) \wedge$   
 $(A_{Uf} * A_{Fw} * A_{Fn} * A_{Un} * A_{Fn})$



# 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):

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

CopyCell

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



# 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,

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

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.

