Separation Logic in the Presence of Garbage Collection

Chung-Kil Hur Derek Dreyer Viktor Vafeiadis

Max Planck Institute for Software Systems (MPI-SWS)
Kaiserslautern and Saarbrücken, Germany

LICS 2011
Toronto, Canada

Separation Logic $=$

Hoare Logic

$$
\begin{aligned}
& \{P\}\ C\ \{Q\} \\
\iff\ & \forall s, h \text{ such that } s, h \models P, \\
& \quad 1.\ C, s, h \text{ does not get stuck} \\
& \quad 2.\ \text{if } C, s, h \leadsto^{*} \mathsf{skip}, s', h' \\
& \quad\quad \text{then } s', h' \models Q
\end{aligned}
$$

$+$ Separating Conjunction “$*$”

$$
\begin{aligned}
& s, h \models P * Q \\
\iff\ & \exists h_1, h_2.\ h = h_1 \uplus h_2 \ \wedge\ s, h_1 \models P \ \wedge\ s, h_2 \models Q
\end{aligned}
$$

Frame rule

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}\ \mathrm{FV}(R) \cap \mathrm{Mod}(C) = \emptyset$$

Two main settings of separation logic

Low-level languages with manual memory management:

- *e.g.*, C with malloc(), free()

High-level languages with automatic memory management:

- *e.g.*, Java, ML
- Garbage collection not observable in operational semantics

Our focus: Low-level languages with garbage collection

Want to support local reasoning about low-level programs that interface to a garbage collector (GC)

- *e.g.*, the output of a compiler for a garbage-collected language, linked with some hand-coded assembly

Want to allow programs to violate the GC's invariants in between calls to the memory allocator

- *e.g.*, creating dangling pointers, performing address arithmetic

Informal local reasoning principles clearly exist,
so we should be able to codify them in separation logic!

- Only work on the topic: [Calcagno, O'Hearn, & Bornat 2003] and [McCreight, Shao, Lin & Li 2007]

Motivating example: Array initialization

$\mathtt{x} := \mathsf{ALLOC}(n);$

$\mathtt{t} := \mathtt{x} + 4\,n;$

while $\mathtt{x} < \mathtt{t}$ do

$\quad [\mathtt{x}] := 0;$

$\quad \mathtt{x} := \mathtt{x} + 4$

od;

$\mathtt{x} := \mathtt{x} - 4\,n;$

$\mathtt{t} := 0$

Motivating example: Array initialization

GC safe →

$\mathtt{x} := \mathsf{ALLOC}(n);$

$\mathtt{t} := \mathtt{x} + 4\,n;$

while $\mathtt{x} < \mathtt{t}$ do

$\quad [\mathtt{x}] := 0;$

$\quad \mathtt{x} := \mathtt{x} + 4$

GC unsafe →

od;

$\mathtt{x} := \mathtt{x} - 4\,n;$

$\mathtt{t} := 0$

GC safe →

Key Challenges

$\{P\}$ GC() $\{P\}$

- Want to give a clean specification for the GC, essentially viewing it as equivalent to skip

The frame rule

- Soundness somewhat subtle due to lack of "heap locality"

High-level ideas

Problem 1: Unreachable blocks may be reclaimed

Conundrum due to [Reynolds 2000]:

$$\{\mathsf{true}\}$$
$$\mathtt{x} := \mathsf{new}();\ [\mathtt{x}] := 5;\ \mathtt{x} := \mathsf{null};$$
$$\{\mathtt{x} = \mathsf{null} \wedge \exists \ell.\ \ell \hookrightarrow 5\}$$

Problem 1: Unreachable blocks may be reclaimed

Conundrum due to [Reynolds 2000]:

$$\{\mathsf{true}\}$$
$$\mathtt{x} := \mathsf{new}();\ [\mathtt{x}] := 5;\ \mathtt{x} := \mathsf{null};$$
$$\{\mathtt{x} = \mathsf{null} \wedge \exists \ell.\ \ell \hookrightarrow 5\}$$
$$\mathsf{GC}()$$
$$\{\mathtt{x} = \mathsf{null} \wedge \exists \ell.\ \ell \hookrightarrow 5\}$$

Problem 1: Unreachable blocks may be reclaimed

Conundrum due to [Reynolds 2000]:

$$\{\mathsf{true}\}$$
$$\mathtt{x} := \mathsf{new}();\ [\mathtt{x}] := 5;\ \mathtt{x} := \mathsf{null};$$
$$\{\mathtt{x} = \mathsf{null} \wedge \exists \ell.\ \ell \hookrightarrow 5\}$$
$$\mathsf{GC}()$$
$$\{\mathtt{x} = \mathsf{null} \wedge \exists \ell.\ \ell \hookrightarrow 5\}$$

Approach by [Calcagno et al. 2003]:
Impose "monster-barring" syntactic restriction on assertions P.

Problem 2: Pointers can be relocated

This triple is easy to validate, even if the GC relocates $\mathtt{x}$:

$$\{\mathtt{x} \hookrightarrow 7\} \quad \mathsf{GC()} \quad \{\mathtt{x} \hookrightarrow 7\}$$

Problem 2: Pointers can be relocated

This triple is hard to validate, because the GC could move ℓ:

$$\{\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7\} \quad \mathsf{GC}() \quad \{\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7\}$$

Problem 2: Pointers can be relocated

This triple is hard to validate, because the GC could move ℓ:

$$\{\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7\} \quad \text{GC()} \quad \{\mathtt{x} \hookrightarrow \ell' * \ell' \hookrightarrow 7\}$$

Problem 2: Pointers can be relocated

This triple is hard to validate, because the GC could move ℓ:

$$\{\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7\} \quad \mathsf{GC}() \quad \{\mathtt{x} \hookrightarrow \ell' * \ell' \hookrightarrow 7\}$$

One approach: Avoid logical variables like ℓ, and use auxiliary program variables instead

Problem 2: Pointers can be relocated

This triple is hard to validate, because the GC could move ℓ:

$$\{\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7\} \quad \mathsf{GC}() \quad \{\mathtt{x} \hookrightarrow \ell' * \ell' \hookrightarrow 7\}$$

One approach: Avoid logical variables like ℓ, and use auxiliary program variables instead

- But we would prefer to use logical variables
- Worse, auxiliary variables may affect the reachability of data

Logical memory (adapted from [McCreight et al. 2007])

$LM \overset{\text{iso}}{\sim} M$: isomorphism between reachable blocks of LM and M

$$\begin{tikzcd}
 & LM \arrow[dl, squiggly, no head, "\text{iso}"'] \arrow[dr, squiggly, no head, "\text{iso}"] & \\
M \arrow[rr, "GC"] & & M'
\end{tikzcd}$$

Logical memory (adapted from [McCreight et al. 2007])

$LM \overset{\text{iso}}{\sim} M$: isomorphism between reachable blocks of LM and M

$\ell \hookrightarrow 5$

$0x80 \hookrightarrow 5$

$$\begin{tikzcd}
 & LM \arrow[dl, squiggly, dash, "\text{iso}"'] \arrow[dr, squiggly, dash, "\text{iso}"] & \\
M \arrow[rr, "GC"] & & M'
\end{tikzcd}$$

$\{\text{true}\}$

$\mathtt{x} := \text{new}();\ [\mathtt{x}] := 5;\ \mathtt{x} := \text{null};$

$\{\mathtt{x} = \text{null} \wedge \exists \ell.\ \ell \hookrightarrow 5\}$

Logical memory (adapted from [McCreight et al. 2007])

$LM \overset{\text{iso}}{\sim} M$: isomorphism between reachable blocks of LM and M

$$\begin{tikzcd}
\ell \hookrightarrow 5 & & LM \arrow[dl, squiggly, no head, "\text{iso}"'] \arrow[dr, squiggly, no head, "\text{iso}"] & & \ell \hookrightarrow 5 \\
0x80 \hookrightarrow 5 & M \arrow[rr, "GC"] & & M' & \mathsf{emp}
\end{tikzcd}$$

$\{\mathsf{true}\}$

$\mathtt{x} := \mathsf{new}();\ [\mathtt{x}] := 5;\ \mathtt{x} := \mathsf{null};$

$\{\mathtt{x} = \mathsf{null} \wedge \exists \ell.\ \ell \hookrightarrow 5\}$

GC()

$\{\mathtt{x} = \mathsf{null} \wedge \exists \ell.\ \ell \hookrightarrow 5\}$

$LM \overset{\text{iso}}{\sim} M$: isomorphism between reachable blocks of LM and M

$\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7$

$\mathtt{x} \hookrightarrow 0x40 * 0x40 \hookrightarrow 7$

$$
\begin{tikzcd}
 & LM \arrow[dl, squiggly, "\text{iso}"'] \arrow[dr, squiggly, "\text{iso}"] & \\
M \arrow[rr, "GC"] & & M'
\end{tikzcd}
$$

$\{\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7\}$

$LM \overset{\text{iso}}{\sim} M$: isomorphism between reachable blocks of LM and M

$\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7$ $\qquad\qquad$ $\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7$

$\mathtt{x} \hookrightarrow 0x40 * 0x40 \hookrightarrow 7$ $\qquad\qquad$ $\mathtt{x} \hookrightarrow 0x60 * 0x60 \hookrightarrow 7$

$$\begin{tikzcd} & LM \arrow[dl, squiggly, "\text{iso}"'] \arrow[dr, squiggly, "\text{iso}"] & \\ M \arrow[rr, "GC"] & & M' \end{tikzcd}$$

$\{\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7\}$ GC() $\{\mathtt{x} \hookrightarrow \ell * \ell \hookrightarrow 7\}$

Semantics of Hoare triples with logical memories

$\{\{P\}\}\ C\ \{\{Q\}\}$

$\iff \forall M, LM$ such that $LM \models P \ \wedge\ LM \overset{\text{iso}}{\sim} M$

1. C, M does not get stuck
2. if $C, M \rightsquigarrow^* \text{skip}, M'$
   then $\exists LM'.\ LM' \models Q \ \wedge\ LM' \overset{\text{iso}}{\sim} M'$

Semantics of Hoare triples with logical memories

$\{\{P\}\}\ C\ \{\{Q\}\}$

$\iff$ $\forall M, LM$ such that $LM \models P \ \wedge\ LM \overset{\text{iso}}{\sim} M$

1. C, M does not get stuck
2. if $C, M \leadsto^* \text{skip}, M'$
   then $\exists LM'.\ LM' \models Q \ \wedge\ LM' \overset{\text{iso}}{\sim} M'$

But in order to guarantee $\{\{P\}\}\ \text{GC}()\ \{\{P\}\}$, we need to ensure that we only invoke the GC under GC-safe memories

Semantics of Hoare triples with logical memories

$$\{\{P\}\}\ C\ \{\{Q\}\}$$

$\iff$ $\forall M, LM$ such that $LM \models P \ \wedge\ LM \overset{\text{iso}}{\sim} M \wedge LM$ safe

1. C, M does not get stuck
2. if $C, M \rightsquigarrow^* \text{skip}, M'$
   then $\exists LM'.\ LM' \models Q \ \wedge\ LM' \overset{\text{iso}}{\sim} M' \wedge LM'$ safe

But in order to guarantee $\{\{P\}\}\ \text{GC}()\ \{\{P\}\}$, we need to ensure that we only invoke the GC under GC-safe memories

GC safety

$LM = (s, h)$

v safe : v is either a non-pointer word
or a pointer to the head of an allocated block.

s safe : all program variables in s contain safe values.

h safe : all reachable blocks in h contain safe values.

LM safe : $LM.s$ safe $\wedge$ $LM.h$ safe.

Semantics of Hoare triples with logical memories

$\{\{P\}\}\ C\ \{\{Q\}\}$

$\iff$ $\forall M, LM$ such that $LM \models P \ \wedge\ LM \overset{\text{iso}}{\sim} M \wedge LM$ safe

1. C, M does not get stuck
2. if $C, M \leadsto^* \mathsf{skip}, M'$
   then $\exists LM'.\ LM' \models Q \ \wedge\ LM' \overset{\text{iso}}{\sim} M' \wedge LM'$ safe

But in order to guarantee $\{\{P\}\}$ GC() $\{\{P\}\}$, we need to ensure that we only invoke the GC under GC-safe memories

Motivating example: Array initialization

GC safe $\rightarrow$

$\mathtt{x} := \mathsf{ALLOC}(n);$

$\mathtt{t} := \mathtt{x} + 4\,n;$

while $\mathtt{x} < \mathtt{t}$ do

  $[\mathtt{x}] := 0;$

  $\mathtt{x} := \mathtt{x} + 4$

GC unsafe $\rightarrow$

od;

$\mathtt{x} := \mathtt{x} - 4\,n;$

$\mathtt{t} := 0$

GC safe $\rightarrow$

Two-level logic

- Outer-level logic

$$\{\{P\}\}\ C\ \{\{Q\}\}$$

$$\iff \forall M, LM \text{ such that } LM \models P \ \wedge\ LM \overset{\text{iso}}{\sim} M \wedge LM \text{ safe}$$

1. C, M does not get stuck
2. if $C, M \rightsquigarrow^{*} \text{skip}, M'$ then $\exists LM'.\ LM' \models Q \ \wedge\ LM' \overset{\text{iso}}{\sim} M' \wedge LM' \text{ safe}$

- Inner-level logic

$$\{P\}\ C\ \{Q\}$$

$$\iff \forall M, LM \text{ such that } LM \models P \ \wedge\ LM \overset{\text{iso}}{\sim} M$$

1. C, M does not get stuck
2. if $C, M \rightsquigarrow^{*} \text{skip}, M'$ then $\exists LM'.\ LM' \models Q \ \wedge\ LM' \overset{\text{iso}}{\sim} M'$

Towards an "inclusion" rule

Obviously unsound:

$$\frac{\{P\}\ C\ \{Q\}}{\{\{P\}\}\ C\ \{\{Q\}\}}$$

Towards an “inclusion” rule

We want something like this . . .

$$\frac{\{P \wedge \text{mem is GC-safe}\}\ C\ \{Q \wedge \text{mem is GC-safe}\}}{\{\{P\}\}\ C\ \{\{Q\}\}}$$

We want something like this ...

$$\frac{\{P \wedge \text{mem is GC-safe}\}\ C\ \{Q \wedge \text{mem is GC-safe}\}}{\{\!\{P\}\!\}\ C\ \{\!\{Q\}\!\}}$$

... but how do we characterize mem is GC-safe?

Towards an "inclusion" rule

We want something like this ...

$$\frac{\{P \wedge \text{mem is GC-safe}\}\ C\ \{Q \wedge \text{mem is GC-safe}\}}{\{\{P\}\}\ C\ \{\{Q\}\}}$$

... but how do we characterize mem is GC-safe?

Solution: We make a simplifying assumption.

- In the inner-level logic, the store may contain unsafe values, but the heap may not.
- This is OK, given how interior pointers are typically used.

Towards an "inclusion" rule

We want something like this . . .

$$\frac{\{P \wedge \boxed{\text{store}} \text{ is GC-safe}\}\ C\ \{Q \wedge \boxed{\text{store}} \text{ is GC-safe}\}}{\{\{P\}\}\ C\ \{\{Q\}\}}$$

. . . but how do we characterize $\boxed{\text{store}}$ is GC-safe?

Solution: We make a simplifying assumption.

- In the inner-level logic, the store may contain unsafe values, but the heap may not.
- This is OK, given how interior pointers are typically used.

Inclusion rule

$$\frac{\{P \wedge \mathsf{safe}(V)\}\ C\ \{Q \wedge \mathsf{safe}(\mathrm{Mod}(C))\}}{\{\{P\}\}\ C\ \{\{Q\}\}}\ V \subseteq \mathrm{ProgVars}$$

- safe is a new primitive predicate in our inner-level logic.

Two-level logic (revisited)

- Outer-level logic

$$\{\{P\}\}\ C\ \{\{Q\}\}$$

$\iff$ $\forall M, LM$ such that $LM \models P \ \wedge\ LM \overset{\text{iso}}{\sim} M \wedge LM$ safe

1. C, M does not get stuck
2. if $C, M \rightsquigarrow^* \text{skip}, M'$
   then $\exists LM'.\ LM' \models Q \ \wedge\ LM' \overset{\text{iso}}{\sim} M' \wedge LM'$ safe

- Inner-level logic

$$\{P\}\ C\ \{Q\}$$

$\iff$ $\forall M, LM$ such that $LM \models P \ \wedge\ LM \overset{\text{iso}}{\sim} M$

1. C, M does not get stuck
2. if $C, M \rightsquigarrow^* \text{skip}, M'$
   then $\exists LM'.\ LM' \models Q \ \wedge\ LM' \overset{\text{iso}}{\sim} M'$

Two-level logic (revisited)

- Outer-level logic

$$\{\{P\}\}\ C\ \{\{Q\}\}$$

$$\iff \forall M, LM \text{ such that } LM \models P \ \land\ LM \overset{\text{iso}}{\sim} M \land LM \text{ safe}$$

1. C, M does not get stuck
2. if $C, M \rightsquigarrow^* \text{skip}, M'$
   then $\exists LM'.\ LM' \models Q \ \land\ LM' \overset{\text{iso}}{\sim} M' \land LM' \text{ safe}$

- Inner-level logic

$$\{P\}\ C\ \{Q\}$$

$$\iff \forall M, LM \text{ such that } LM \models P \ \land\ LM \overset{\text{iso}}{\sim} M \land LM.h \text{ safe}$$

1. C, M does not get stuck
2. if $C, M \rightsquigarrow^* \text{skip}, M'$
   then $\exists LM'.\ LM' \models Q \ \land\ LM' \overset{\text{iso}}{\sim} M' \land LM'.h \text{ safe}$

Frame rule

$$\frac{\{P\}\ C\ \{Q\}}{\{P * R\}\ C\ \{Q * R\}}\ \mathrm{FV}(R) \cap \mathrm{Mod}(C) = \emptyset$$

Our semantics so far doesn't support frame,
because the presence of a GC violates "heap locality"

- Solution: Following [Birkedal et al. 2006],
  we bake the frame rule into the semantics of triples

- Outer-level logic

$\{\{P\}\}\ C\ \{\{Q\}\}$

$\iff$ $\forall M, LM$ such that $LM \models P \;\wedge\; LM \stackrel{\text{iso}}{\sim} M \;\wedge\; LM$ safe

1. C, M does not get stuck
2. if $C, M \leadsto^* \text{skip}, M'$
   then $\exists LM'.\ LM' \models Q \;\wedge\; LM' \stackrel{\text{iso}}{\sim} M' \;\wedge\; LM'$ safe

- Inner-level logic

$\{P\}\ C\ \{Q\}$

$\iff$ $\forall M, LM$ such that $LM \models P \;\wedge\; LM \stackrel{\text{iso}}{\sim} M \;\wedge\; LM.h$ safe

1. C, M does not get stuck
2. if $C, M \leadsto^* \text{skip}, M'$
   then $\exists LM'.\ LM' \models Q \;\wedge\; LM' \stackrel{\text{iso}}{\sim} M' \;\wedge\; LM'.h$ safe

Baking the frame rule in

- Outer-level logic

$\{\{P\}\}\ C\ \{\{Q\}\}$

$\iff$ $\forall M, LM, LM_{\mathsf{f}}$ such that $LM \models P \ \wedge\ LM \uplus LM_{\mathsf{f}} \overset{\text{iso}}{\sim} M \ \wedge\ LM \uplus LM_{\mathsf{f}}$ safe

1. C, M does not get stuck
2. if $C, M \leadsto^* \mathsf{skip}, M'$
   then $\exists LM'.\ LM' \models Q \ \wedge\ LM' \uplus LM_{\mathsf{f}} \overset{\text{iso}}{\sim} M' \ \wedge\ LM' \uplus LM_{\mathsf{f}}$ safe

- Inner-level logic

$\{P\}\ C\ \{Q\}$

$\iff$ $\forall M, LM, LM_{\mathsf{f}}$ such that $LM \models P \ \wedge\ LM \uplus LM_{\mathsf{f}} \overset{\text{iso}}{\sim} M \ \wedge\ (LM \uplus LM_{\mathsf{f}}).h$ safe

1. C, M does not get stuck
2. if $C, M \leadsto^* \mathsf{skip}, M'$
   then $\exists LM'.\ LM' \models Q \ \wedge\ LM' \uplus LM_{\mathsf{f}} \overset{\text{iso}}{\sim} M' \ \wedge\ (LM' \uplus LM_{\mathsf{f}}).h$ safe

Proof rules & Examples

Logical entities

$$
\begin{aligned}
\mathrm{Words} &\stackrel{\mathsf{def}}{=} \{\, w \in \mathbb{Z} \,\} \\
\mathrm{Locs} &\stackrel{\mathsf{def}}{=} \{\, \boldsymbol{\ell}_1, \boldsymbol{\ell}_2, \ldots \,\} \\
\mathrm{LogPtrs} &\stackrel{\mathsf{def}}{=} \{\, \boldsymbol{\ell} \widehat{+} i \mid \boldsymbol{\ell} \in \mathrm{Locs} \wedge i \in \mathbb{Z} \,\} \\
\mathrm{LogVals} &\stackrel{\mathsf{def}}{=} \{\, \mathbf{v} \in \mathrm{Words} \uplus \mathrm{LogPtrs} \,\} \\
\mathrm{LStores} &\stackrel{\mathsf{def}}{=} \{\, \mathbf{s} \in \mathrm{ProgVars} \to \mathrm{LogVals} \,\} \\
\mathrm{LHeaps} &\stackrel{\mathsf{def}}{=} \{\, \mathbf{h} \in \mathrm{Locs} \rightharpoonup_{\mathrm{fin}} \mathbb{N} \rightharpoonup_{\mathrm{fin}} \mathrm{LogVals} \,\}
\end{aligned}
$$

Assertions

- Outer-level assertions

$$
\begin{aligned}
P \;:=\;& \mathbf{E} \mid \mathrm{logptr}(\mathbf{E}) \mid \mathrm{word}(\mathbf{E}) \\
\mid\;& \mathbf{E} \hookrightarrow \mathbf{E} \mid P * P \mid P \mathrel{-\!\!*} P \\
\mid\;& P \Rightarrow P \mid P \wedge P \mid P \vee P \mid \forall v.\, P \mid \exists v.\, P
\end{aligned}
$$

- Inner-level assertions

$$
\begin{aligned}
\mathbf{P} \;:=\;& \mathrm{safe}(\mathbf{E}) \\
\mid\;& \mathbf{E} \mid \mathrm{logptr}(\mathbf{E}) \mid \mathrm{word}(\mathbf{E}) \\
\mid\;& \mathbf{E} \hookrightarrow \mathbf{E} \mid \mathbf{P} * \mathbf{P} \mid \mathbf{P} \mathrel{-\!\!*} \mathbf{P} \\
\mid\;& \mathbf{P} \Rightarrow \mathbf{P} \mid \mathbf{P} \wedge \mathbf{P} \mid \mathbf{P} \vee \mathbf{P} \mid \forall v.\, \mathbf{P} \mid \exists v.\, \mathbf{P}
\end{aligned}
$$

Selected proof rules

$$\frac{}{\{\mathtt{x} = v \wedge E = E)\}\ \mathtt{x} := E\ \{\mathtt{x} = E[v/\mathtt{x}]\}}\quad \text{(Assign)}$$

$$\frac{}{\{\mathtt{x} = u \wedge E \hookrightarrow v\}\ \mathtt{x} := [E]\ \{\mathtt{x} = v \wedge E[u/\mathtt{x}] \hookrightarrow v\}}\quad \text{(Read)}$$

$$\frac{}{\{E \hookrightarrow - \wedge \mathsf{safe}(E')\}\ [E] := E'\ \{E \hookrightarrow E'\}}\quad \text{(Write)}$$

$$\frac{n \geq 0}{\{\{\mathsf{true}\}\}\ \mathtt{x} := \mathsf{ALLOC}(n)\ \{\{\mathtt{x} \hookrightarrow_n -, \ldots, -\}\}}\quad \text{(Alloc)}$$

Example 1: Array initialization

$$
\begin{array}{l}
\mathtt{x} := \mathsf{ALLOC}(n); \\
\mathtt{t} := \mathtt{x} + 4\,n; \\
\mathsf{while}\ \mathtt{x} < \mathtt{t}\ \mathsf{do} \\
\quad [\mathtt{x}] := 0; \\
\quad \mathtt{x} := \mathtt{x} + 4 \\
\mathsf{od}; \\
\mathtt{x} := \mathtt{x} - 4\,n; \\
\mathtt{t} := 0
\end{array}
$$

Example 1: Array initialization

$\mathtt{x} := \mathsf{ALLOC}(n);$

$$\overbrace{([\mathtt{x}] := 0;\ \mathtt{x} := \mathtt{x} + 4);\ \ \ldots;\ ([\mathtt{x}] := 0;\ \mathtt{x} := \mathtt{x} + 4)}^{n \text{ times}}$$

$\mathtt{x} := \mathtt{x} - 4\,n$

Example 1: Array initialization

$\{\{\mathsf{true}\}\}$

$\mathtt{x} := \mathsf{ALLOC}(n);$

$\{\{\mathtt{x} \hookrightarrow_n -, \ldots, -\}\}$

$$\overbrace{([\mathtt{x}] := 0;\ \mathtt{x} := \mathtt{x} + 4);\ \ \ldots;\ ([\mathtt{x}] := 0;\ \mathtt{x} := \mathtt{x} + 4)}^{n \text{ times}}$$

$\mathtt{x} := \mathtt{x} - 4\,n$

$\{\{\mathtt{x} \hookrightarrow_n 0, \ldots, 0\}\}$

Example 1: Array initialization

$$\boxed{\dfrac{\{P \wedge \mathsf{safe}(V)\}\ C\ \{Q \wedge \mathsf{safe}(\mathrm{Mod}(C))\}}{\{\!\{P\}\!\}\ C\ \{\!\{Q\}\!\}}}$$

$\{\!\{\mathsf{true}\}\!\}$

$\mathtt{x} := \mathsf{ALLOC}(n);$

$\{\!\{\mathtt{x} \hookrightarrow_n -, \ldots, -\}\!\}$

$\{\mathtt{x} \hookrightarrow_n -, \ldots, - \wedge \mathsf{safe}(\mathtt{x})\}$

$$\overbrace{([\mathtt{x}] := 0;\ \mathtt{x} := \mathtt{x} + 4);\ \ \ldots;\ ([\mathtt{x}] := 0;\ \mathtt{x} := \mathtt{x} + 4)}^{n \text{ times}}$$

$\mathtt{x} := \mathtt{x} - 4\,n$

$\{\mathtt{x} \hookrightarrow_n 0, \ldots, 0 \wedge \mathsf{safe}(\mathtt{x})\}$

$\{\!\{\mathtt{x} \hookrightarrow_n 0, \ldots, 0\}\!\}$

Example 1: Array initialization

$$\boxed{\dfrac{\{P \wedge \mathsf{safe}(V)\}\ C\ \{Q \wedge \mathsf{safe}(\mathrm{Mod}(C))\}}{\{\{P\}\}\ C\ \{\{Q\}\}}}$$

$\{\{\mathsf{true}\}\}$

$\mathtt{x} := \mathsf{ALLOC}(n);$

$\{\{\mathtt{x} \hookrightarrow_n -, \ldots, -\}\}$

$\{\mathtt{x} \hookrightarrow_n -, \ldots, - \wedge \mathsf{safe}(\mathtt{x})\}$

$$\overbrace{([\mathtt{x}] := 0;\ \mathtt{x} := \mathtt{x} + 4);\ \ \ldots;\ ([\mathtt{x}] := 0;\ \mathtt{x} := \mathtt{x} + 4)}^{n \text{ times}}$$

$\{\mathtt{x} - 4\,n \hookrightarrow_n 0, \ldots, 0 \wedge \mathsf{safe}(\mathtt{x} - 4\,n)\}$

$\mathtt{x} := \mathtt{x} - 4\,n$

$\{\mathtt{x} \hookrightarrow_n 0, \ldots, 0 \wedge \mathsf{safe}(\mathtt{x})\}$

$\{\{\mathtt{x} \hookrightarrow_n 0, \ldots, 0\}\}$

Example 1: Array initialization

For the original example, note that the setting of `t` to a safe value is important, since `t` is modified by the program.

$$\mathtt{x} := \mathsf{ALLOC}(n);$$

$$\mathtt{t} := \mathtt{x} + 4\,n;$$

$$\text{while } \mathtt{x} < \mathtt{t} \text{ do}$$

$$\quad [\mathtt{x}] := 0;$$

$$\quad \mathtt{x} := \mathtt{x} + 4$$

$$\text{od};$$

$$\mathtt{x} := \mathtt{x} - 4\,n;$$

$$\boxed{\mathtt{t} := 0}$$

Example 2: Add & Square

$$\mathtt{i} := (\mathtt{i} + \mathtt{j} - 2) \div 2;$$

$$\mathtt{i} := \mathtt{i} \times \mathtt{i};\ \mathtt{i} := 2 \times \mathtt{i} + 1$$

Example 2: Add & Square

$$\{\{\mathtt{i} = 2n + 1 \wedge \mathtt{j} = 2m + 1\}\}$$

$$\mathtt{i} := (\mathtt{i} + \mathtt{j} - 2) \div 2;$$

$$\mathtt{i} := \mathtt{i} \times \mathtt{i};\ \mathtt{i} := 2 \times \mathtt{i} + 1$$

$$\{\{\mathtt{i} = 2(n + m)^2 + 1 \wedge \mathtt{j} = 2m + 1\}\}$$

Example 2: Add & Square

$$\{\{\mathtt{i} = 2n+1 \land \mathtt{j} = 2m+1\}\}$$
$$\{\mathtt{i} = 2n+1 \land \mathtt{j} = 2m+1 \land \mathsf{word}(n,m)\}$$
$$\mathtt{i} := (\mathtt{i} + \mathtt{j} - 2) \div 2;$$
$$\{\mathtt{i} = n+m \land \mathtt{j} = 2m+1 \land \mathsf{word}(n,m)\}$$
$$\mathtt{i} := \mathtt{i} \times \mathtt{i};\ \mathtt{i} := 2 \times \mathtt{i} + 1$$
$$\{\mathtt{i} = 2(n+m)^2+1 \land \mathtt{j} = 2m+1\}$$
$$\{\mathtt{i} = 2(n+m)^2+1 \land \mathtt{j} = 2m+1 \land \mathsf{safe}(\mathtt{i})\}$$
$$\{\{\mathtt{i} = 2(n+m)^2+1 \land \mathtt{j} = 2m+1\}\}$$

Conclusion

- Summary
  - Separation logic to reason about low-level programs that might violate GC safety in between calls to the GC
  - Key ideas:
    - Logical memory
    - Two-level logic with “inclusion” rule & safe predicate
  - Detailed soundness proof (in the technical appendix)

Conclusion

- Summary
  - Separation logic to reason about low-level programs that might violate GC safety in between calls to the GC
  - Key ideas:
    - Logical memory
    - Two-level logic with "inclusion" rule & safe predicate
  - Detailed soundness proof (in the technical appendix)

- Limitations
  - Only accounts for stop-the-world collectors
  - Conjunction rule is unsound
  - Example we should but can't prove in general:

$$\{\mathrm{x} = v \wedge \mathrm{y} = w\}$$
$$\mathrm{x} := \mathrm{x} \text{ xor } \mathrm{y}; \quad \mathrm{y} := \mathrm{x} \text{ xor } \mathrm{y}; \quad \mathrm{x} := \mathrm{x} \text{ xor } \mathrm{y}$$
$$\{\mathrm{x} = w \wedge \mathrm{y} = v\}$$