

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

Separation Logic =
Hoare Logic

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

$$\iff \forall s, h \text{ such that } s, h \models P,$$

1. C, s, h does not get stuck
2. if $C, s, h \rightsquigarrow^* \text{skip}, s', h'$
then $s', h' \models Q$

+ Separating Conjunction “*”

$$s, h \models P * Q$$

$$\iff \exists h_1, h_2. h = h_1 \uplus h_2 \wedge s, h_1 \models P \wedge s, h_2 \models Q$$

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

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

```
x := ALLOC(n);  
t := x + 4n;  
while x < t do  
  [x] := 0;  
  x := x + 4  
od;  
x := x - 4n;  
t := 0
```

Motivating example: Array initialization

GC safe \rightarrow

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

$t := x + 4n;$

while $x < t$ do

$[x] := 0;$

$x := x + 4$

GC unsafe \rightarrow

od;

$x := x - 4n;$

$t := 0$

GC safe \rightarrow

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

- 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]:

$\{\text{true}\}$

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

$\{x = \text{null} \wedge \exists l. l \leftrightarrow 5\}$

Problem 1: Unreachable blocks may be reclaimed

Conundrum due to [Reynolds 2000]:

$\{\text{true}\}$

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

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

$\text{GC}()$

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

Problem 1: Unreachable blocks may be reclaimed

Conundrum due to [Reynolds 2000]:

$\{\text{true}\}$

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

$\{x = \text{null} \wedge \exists l. l \leftrightarrow 5\}$

$\text{GC}()$

$\{x = \text{null} \wedge \exists l. l \leftrightarrow 5\}$

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 x :

$$\{x \hookrightarrow 7\} \quad \text{GC}() \quad \{x \hookrightarrow 7\}$$

Problem 2: Pointers can be relocated

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

$$\{x \hookrightarrow l * l \hookrightarrow 7\} \quad \text{GC}() \quad \{x \hookrightarrow l * l \hookrightarrow 7\}$$

Problem 2: Pointers can be relocated

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

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

Problem 2: Pointers can be relocated

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

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

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 :

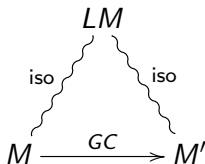
$$\{x \hookrightarrow l * l \hookrightarrow 7\} \quad \text{GC}() \quad \{x \hookrightarrow l' * l' \hookrightarrow 7\}$$

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])

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

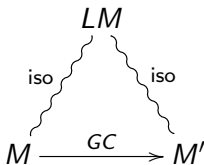


Logical memory (adapted from [McCreight et al. 2007])

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

$l \leftrightarrow 5$

$0x80 \leftrightarrow 5$



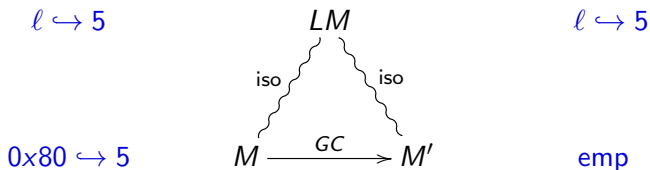
$\{\text{true}\}$

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

$\{x = \text{null} \wedge \exists l. l \leftrightarrow 5\}$

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

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



$\{\text{true}\}$

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

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

$\text{GC}()$

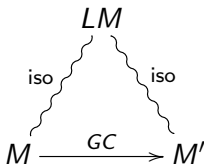
$\{x = \text{null} \wedge \exists l. l \hookrightarrow 5\}$

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

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

$$x \hookrightarrow l * l \hookrightarrow 7$$

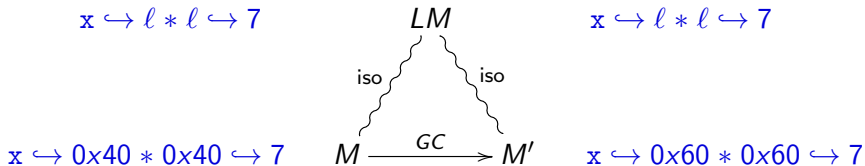
$$x \hookrightarrow 0x40 * 0x40 \hookrightarrow 7$$



$$\{x \hookrightarrow l * l \hookrightarrow 7\}$$

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

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



$$\{x \mapsto l * l \mapsto 7\} \quad GC() \quad \{x \mapsto l * l \mapsto 7\}$$

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

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

Semantics of Hoare triples with logical memories

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

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

1. C, M does not get stuck

2. if $C, M \rightsquigarrow^* \text{skip}, M'$

then $\exists LM'. LM' \models Q \wedge LM' \overset{\text{iso}}{\sim} M'$

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

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

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

1. C, M does not get stuck

2. if $C, M \rightsquigarrow^* \text{skip}, M'$

then $\exists LM'. LM' \models Q \wedge LM' \overset{\text{iso}}{\sim} M' \wedge LM' \text{ safe}$

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

$LM = (s, h)$

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

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

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

1. C, M does not get stuck

2. if $C, M \rightsquigarrow^* \text{skip}, M'$

then $\exists LM'. LM' \models Q \wedge LM' \overset{\text{iso}}{\sim} M' \wedge LM' \text{ safe}$

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

Motivating example: Array initialization

GC safe \rightarrow

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

$t := x + 4n;$

while $x < t$ do

$[x] := 0;$

$x := x + 4$

GC unsafe \rightarrow

od;

$x := x - 4n;$

$t := 0$

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 $\exists LM'. LM' \models Q \wedge LM' \overset{\text{iso}}{\sim} M'$

Obviously unsound:

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

Towards an “inclusion” rule

We want something like this ...

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

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**?

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.

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

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

Two-level logic (revisited)

- Outer-level logic

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

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

1. C, M does not get stuck

2. if $C, M \rightsquigarrow^* \text{skip}, M'$

then $\exists LM'. LM' \models Q \wedge LM' \stackrel{\text{iso}}{\sim} M' \wedge LM' \text{ safe}$

- Inner-level logic

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

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

1. C, M does not get stuck

2. if $C, M \rightsquigarrow^* \text{skip}, M'$

then $\exists LM'. LM' \models Q \wedge LM' \stackrel{\text{iso}}{\sim} M'$

Two-level logic (revisited)

- Outer-level logic

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

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

1. C, M does not get stuck

2. if $C, M \rightsquigarrow^* \text{skip}, M'$

$$\text{then } \exists LM'. LM' \models Q \wedge LM' \stackrel{\text{iso}}{\sim} M' \wedge LM' \text{ safe}$$

- Inner-level logic

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

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

1. C, M does not get stuck

2. if $C, M \rightsquigarrow^* \text{skip}, M'$

$$\text{then } \exists LM'. LM' \models Q \wedge LM' \stackrel{\text{iso}}{\sim} M' \wedge LM'.h \text{ safe}$$

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

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

Baking the frame rule in

- 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

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

1. C, M does not get stuck

2. if $C, M \rightsquigarrow^* \text{skip}, M'$

then $\exists LM'. LM' \models Q \wedge LM' \overset{\text{iso}}{\sim} M' \wedge LM'.h \text{ safe}$

Baking the frame rule in

- Outer-level logic

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

$\Leftrightarrow \forall M, LM, LM_f$ such that $LM \models P \wedge LM \uplus LM_f \stackrel{\text{iso}}{\sim} M \wedge LM \uplus LM_f$ safe

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

- Inner-level logic

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

$\Leftrightarrow \forall M, LM, LM_f$ such that $LM \models P \wedge LM \uplus LM_f \stackrel{\text{iso}}{\sim} M \wedge (LM \uplus LM_f).h$ safe

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

Proof rules & Examples

$$\begin{aligned}\text{Words} &\stackrel{\text{def}}{=} \{ w \in \mathbb{Z} \} \\ \text{Locs} &\stackrel{\text{def}}{=} \{ \ell_1, \ell_2, \dots \} \\ \text{LogPtrs} &\stackrel{\text{def}}{=} \{ \ell \hat{+} i \mid \ell \in \text{Locs} \wedge i \in \mathbb{Z} \} \\ \text{LogVals} &\stackrel{\text{def}}{=} \{ \mathbf{v} \in \text{Words} \uplus \text{LogPtrs} \} \\ \text{LStores} &\stackrel{\text{def}}{=} \{ \mathbf{s} \in \text{ProgVars} \rightarrow \text{LogVals} \} \\ \text{LHeaps} &\stackrel{\text{def}}{=} \{ \mathbf{h} \in \text{Locs} \rightarrow_{\text{fin}} \mathbb{N} \rightarrow_{\text{fin}} \text{LogVals} \}\end{aligned}$$

Assertions

- Outer-level assertions

$$\begin{aligned} P &::= \mathbf{E} \mid \text{logptr}(\mathbf{E}) \mid \text{word}(\mathbf{E}) \\ &\mid \mathbf{E} \hookrightarrow \mathbf{E} \mid P * P \mid P \multimap P \\ &\mid P \Rightarrow P \mid P \wedge P \mid P \vee P \mid \forall v. P \mid \exists v. P \end{aligned}$$

- Inner-level assertions

$$\begin{aligned} \mathbf{P} &::= \text{safe}(\mathbf{E}) \\ &\mid \mathbf{E} \mid \text{logptr}(\mathbf{E}) \mid \text{word}(\mathbf{E}) \\ &\mid \mathbf{E} \hookrightarrow \mathbf{E} \mid \mathbf{P} * \mathbf{P} \mid \mathbf{P} \multimap \mathbf{P} \\ &\mid \mathbf{P} \Rightarrow \mathbf{P} \mid \mathbf{P} \wedge \mathbf{P} \mid \mathbf{P} \vee \mathbf{P} \mid \forall v. \mathbf{P} \mid \exists v. \mathbf{P} \end{aligned}$$

Selected proof rules

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

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

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

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

Example 1: Array initialization

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

$t := x + 4n;$

while $x < t$ do

$[x] := 0;$

$x := x + 4$

od;

$x := x - 4n;$

$t := 0$

Example 1: Array initialization

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

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

$\mathbf{x} := \mathbf{x} - 4n$

Example 1: Array initialization

$\{\{ \text{true} \}\}$

$\mathbf{x} := \text{ALLOC}(n);$

$\{\{ \mathbf{x} \hookrightarrow_n -, \dots, - \}\}$

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

$\mathbf{x} := \mathbf{x} - 4n$

$\{\{ \mathbf{x} \hookrightarrow_n 0, \dots, 0 \}\}$

Example 1: Array initialization

$\{\{ \text{true} \}\}$

$$\frac{\{P \wedge \text{safe}(V)\} C \{Q \wedge \text{safe}(\text{Mod}(C))\}}{\{\{P\}\} C \{\{Q\}\}}$$

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

$\{\{x \hookrightarrow_n -, \dots, -\}\}$

$\{x \hookrightarrow_n -, \dots, - \wedge \text{safe}(x)\}$

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

$\mathbf{x} := \mathbf{x} - 4n$

$\{x \hookrightarrow_n 0, \dots, 0 \wedge \text{safe}(x)\}$

$\{\{x \hookrightarrow_n 0, \dots, 0\}\}$

Example 1: Array initialization

$\{\{\text{true}\}\}$

$$\frac{\{P \wedge \text{safe}(V)\} C \{Q \wedge \text{safe}(\text{Mod}(C))\}}{\{\{P\}\} C \{\{Q\}\}}$$

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

$\{\{x \hookrightarrow_n -, \dots, -\}\}$

$\{x \hookrightarrow_n -, \dots, - \wedge \text{safe}(x)\}$

n times

$\overbrace{([x] := 0; x := x + 4); \dots; ([x] := 0; x := x + 4)}$

$\{x - 4n \hookrightarrow_n 0, \dots, 0 \wedge \text{safe}(x - 4n)\}$

$x := x - 4n$

$\{x \hookrightarrow_n 0, \dots, 0 \wedge \text{safe}(x)\}$

$\{\{x \hookrightarrow_n 0, \dots, 0\}\}$

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.

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

$t := x + 4n;$

while $x < t$ do

$[x] := 0;$

$x := x + 4$

od;

$x := x - 4n;$

$t := 0$

Example 2: Add & Square

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

$i := i \times i; i := 2 \times i + 1$

Example 2: Add & Square

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

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

$$i := i \times i; i := 2 \times i + 1$$

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

Example 2: Add & Square

$$\begin{aligned} & \{\{i = 2n + 1 \wedge j = 2m + 1\}\} \\ & \quad \{i = 2n + 1 \wedge j = 2m + 1 \wedge \text{word}(n, m)\} \\ & \quad i := (i + j - 2) \div 2; \\ & \quad \{i = n + m \wedge j = 2m + 1 \wedge \text{word}(n, m)\} \\ & \quad i := i \times i; \quad i := 2 \times i + 1 \\ & \quad \{i = 2(n + m)^2 + 1 \wedge j = 2m + 1\} \\ & \quad \{i = 2(n + m)^2 + 1 \wedge j = 2m + 1 \wedge \text{safe}(i)\} \\ & \{\{i = 2(n + m)^2 + 1 \wedge j = 2m + 1\}\} \end{aligned}$$

- 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)

- 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:

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