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

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

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

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

+ Separating Conjunction “∗”

s, h \models P \ast Q

\iff \exists h_1, h_2. h = h_1 \uplus h_2 \land s, h_1 \models P \land s, h_2 \models Q
\[
\frac{\{P\} \ C \ \{Q\}}{\{P \ast R\} \ C \ \{Q \ast R\}} \quad \text{FV}(R) \cap \text{Mod}(C) = \emptyset
\]
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
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

\[\begin{align*}
x &:= \text{ALLOC}(n); \\
t &:= x + 4n; \\
\text{while } x < t \text{ do} \\
& \quad [x] := 0; \\
& \quad x := x + 4 \\
\text{od;}
\end{align*}\]

\[\begin{align*}
x &:= x - 4n; \\
t &:= 0
\end{align*}\]
Motivating example: Array initialization

GC safe →

\[ x := \text{ALLOC}(n); \]
\[ t := x + 4n; \]

while \( x < t \) do

\[ [x] := 0; \]
\[ x := x + 4 \]

GC unsafe →

od;

\[ x := x - 4n; \]
\[ t := 0 \]

GC safe →
Key Challenges

\{P\} \text{GC()} \{P\}

- Want to give a clean specification for the GC, essentially viewing it as equivalent to \text{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]:

\[
\begin{align*}
&\{\text{true}\} \\
x &:= \text{new}(); \ [x] := 5; \ x := \text{null}; \\
&\{x = \text{null} \land \exists l. l \rightarrow 5\}
\end{align*}
\]
Conundrum due to [Reynolds 2000]:

\[
\{\text{true}\} \\
x := \text{new}(); \quad [x] := 5; \quad x := \text{null}; \\
\{x = \text{null} \land \exists \ell. \ell \rightarrow 5\} \\
\text{GC()} \\
\{x = \text{null} \land \exists \ell. \ell \rightarrow 5\}
\]
Conundrum due to [Reynolds 2000]:

\[
\{ \text{true} \} \\
x := \text{new()}; \ [x] := 5; \ x := \text{null}; \\
\{ x = \text{null} \land \exists \ell. \ \ell \rightarrow 5 \} \\
\text{GC()} \\
\{ x = \text{null} \land \exists \ell. \ \ell \rightarrow 5 \}
\]

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 \( x \):

\[
\{ x \leftarrow 7 \} \quad \text{GC()} \quad \{ x \leftarrow 7 \}
\]
Problem 2: Pointers can be relocated

This triple is hard to validate, because the GC could move $\ell$:

$$\{x \leftarrow \ell \land \ell \leftarrow 7\} \quad \text{GC(\quad)} \quad \{x \leftarrow \ell \land \ell \leftarrow 7\}$$
Problem 2: Pointers can be relocated

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

\[
\{x \leftrightarrow ℓ \ast ℓ \leftrightarrow 7\} \quad \text{GC()} \quad \{x \leftrightarrow ℓ' \ast ℓ' \leftrightarrow 7\}
\]
Problem 2: Pointers can be relocated

This triple is hard to validate, because the GC could move $\ell$:

$$\{x \leftrightarrow \ell \ast \ell \leftrightarrow 7\} \quad \text{GC()} \quad \{x \leftrightarrow \ell' \ast \ell' \leftrightarrow 7\}$$

One approach: Avoid logical variables like $\ell$, and use auxiliary program variables instead
Problem 2: Pointers can be relocated

This triple is hard to validate, because the GC could move $\ell$:

\[
\{ x \leftrightarrow \ell \ast \ell \leftarrow 7 \} \quad \text{GC()} \quad \{ x \leftrightarrow \ell' \ast \ell' \leftarrow 7 \}
\]

One approach: Avoid logical variables like $\ell$, and use auxiliary program variables instead

- But we would prefer to use logical variables
- Worse, auxiliary variables may affect the reachability of data
$LM \xrightarrow{iso} M$: isomorphism between reachable blocks of $LM$ and $M$
Logical memory (adapted from [McCreight et al. 2007])

$LM \xrightarrow{iso} M$: isomorphism between reachable blocks of $LM$ and $M$

\[
\begin{array}{c}
\ell \leftrightarrow 5 \\
0x80 \leftrightarrow 5 \\
\end{array}
\]

\[
\begin{array}{c}
LM \\
\xrightarrow{iso} M \xrightarrow{GC} M'
\end{array}
\]

\[
\{\text{true}\}
\]

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

\[
\{x = \text{null} \land \exists \ell. \ell \leftrightarrow 5\}
\]
Logical memory (adapted from [McCreight et al. 2007]):

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

\[
\begin{align*}
\ell & \leftrightarrow 5 \\
0x80 & \leftrightarrow 5
\end{align*}
\]

C.-K. Hur, D. Dreyer, V. Vafeiadis

Separation Logic in the Presence of Garbage Collection
$LM \overset{iso}{\sim} M$: isomorphism between reachable blocks of $LM$ and $M$

\[
x \leftarrow \ell \ast \ell \rightarrow 7
\]

\[
x \leftarrow 0x40 \ast 0x40 \rightarrow 7
\]

\[
\{ x \leftarrow \ell \ast \ell \rightarrow 7 \}
\]
**Logical memory (adapted from [McCreight et al. 2007])**

\( LM \iso M \): isomorphism between reachable blocks of \( LM \) and \( M \)

\[ x \leftarrow l \ast l \leftarrow 7 \]

\(\begin{array}{ccc}
LM & \xrightarrow{\text{iso}} & M \\
\text{iso} & \text{iso} & \text{iso}
\end{array}\)

\[ M \xrightarrow{\text{GC}} M' \]

\[ x \leftarrow 0x40 \ast 0x40 \leftarrow 7 \]

\[ x \leftarrow 0x60 \ast 0x60 \leftarrow 7 \]

\[ \{ x \leftarrow l \ast l \leftarrow 7 \} \]

\[ \text{GC()} \]

\[ \{ x \leftarrow l \ast l \leftarrow 7 \} \]
Semantics of Hoare triples with logical memories

\[
\begin{align*}
\{\{P\}\} \ C \ \{\{Q\}\} \\
\iff \ \forall M, LM \text{ such that } LM \models P \land LM \overset{\text{iso}}{\sim} M \\
1. \ C, M \text{ does not get stuck} \\
2. \text{if } C, M \overset{*}{\leadsto} \text{skip, } M' \\
\text{then } \exists LM'. \ LM' \models Q \land LM' \overset{\text{iso}}{\sim} M'
\end{align*}
\]
\{\{P\}\} \ C \ \{\{Q\}\}

\iff \forall M, LM \text{ such that } LM \models P \land LM \isom M

1. C, M does not get stuck
2. if C, M \leadsto^* \text{ skip, } M'

then \exists LM'. LM' \models Q \land LM' \isom M'

But in order to guarantee \{\{P\}\} \ GC() \ \{\{P\}\}, we need to ensure that we only invoke the GC under GC-safe memories
Semantics of Hoare triples with logical memories

\[
\vdash \{\{P\}\} \ C \ \{\{Q\}\}
\]

\[\iff \forall M, LM \text{ such that } LM \models P \land LM \isomorph M \land LM \text{ safe}
\]

1. \(C, M\) does not get stuck
2. if \(C, M \leadsto^* \text{ skip}, M'\)
   then \(\exists LM'. \ LM' \models Q \land LM' \isomorph M' \land LM' \text{ safe}\)

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.

\[ LM \text{ safe} : \ LM. s \text{ safe} \land \ LM. h \text{ safe}. \]
Semantics of Hoare triples with logical memories

\[
\{\{P\}\} \ C \ \{\{Q\}\}
\]

\[\iff \ \forall M, LM \text{ such that } LM \models P \land LM \iso M \land LM \text{ safe}\]

1. \(C, M\) does not get stuck
2. if \(C, M \Rightarrow^* \text{skip}, M'\)
   
   then \(\exists LM'. \ LM' \models Q \land LM' \iso M' \land LM' \text{ safe}\)

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

\begin{align*}
\text{GC safe} & \rightarrow \\
& x := \text{ALLOC}(n); \\
& t := x + 4n; \\
& \text{while } x < t \text{ do} \\
& \quad [x] := 0; \\
& \quad x := x + 4 \\
\text{GC unsafe} & \rightarrow \\
& \text{od;} \\
& x := x - 4n; \\
& t := 0 \\
\text{GC safe} & \rightarrow 
\end{align*}
Two-level logic

- **Outer-level logic**

  \[
  \{\{P\}\} \ C \ \{\{Q\}\} \iff \forall M, LM \text{ such that } LM \models P \land LM^{iso} \sim M \land LM \text{ safe}
  \]

  1. \(C, M\) does not get stuck
  2. if \(C, M \rightsquigarrow^* \text{skip, } M'\)

     then \(\exists LM'. LM' \models Q \land LM'^{iso} \sim M' \land LM' \text{ safe}\)

- **Inner-level logic**

  \[
  \{P\} \ C \ {Q} \iff \forall M, LM \text{ such that } LM \models P \land LM^{iso} \sim M
  \]

  1. \(C, M\) does not get stuck
  2. if \(C, M \rightsquigarrow^* \text{skip, } M'\)

     then \(\exists LM'. LM' \models Q \land LM'^{iso} \sim M'\)
Towards an “inclusion” rule

Obviously unsound:

\[
\{ P \} \quad C \quad \{ Q \}
\]

\[
\{ \{ P \} \} \quad C \quad \{ \{ Q \} \}
\]
Towards an “inclusion” rule

We want something like this . . .

\[
\begin{align*}
\{ P \land \text{mem is GC-safe} \} & \xRightarrow{C} \{ Q \land \text{mem is GC-safe} \} \\
\{\{ P \}\} & \xRightarrow{C} \{\{ Q \}\}
\end{align*}
\]
We want something like this . . .

\[
\{ P \land \text{mem is GC-safe} \} \xRightarrow{C} \{ Q \land \text{mem is GC-safe} \}
\]

\[
\{ \{ P \} \} \xRightarrow{C} \{ \{ Q \} \}
\]

...but how do we characterize \text{mem is GC-safe}?
Towards an “inclusion” rule

We want something like this . . .

\[
\{ P \land \text{mem is GC-safe} \} \ C \ \{ Q \land \text{mem is GC-safe} \} \\
\{\{ P\}\} \ C \ \{\{ Q\}\}
\]

. . . but how do we characterize \text{mem is GC-safe}?

\textbf{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 . . .

\[
\begin{align*}
\{ P \land \text{store is GC-safe} \} & \implies C \{ Q \land \text{store is GC-safe} \} \\
\{ \{ P \} \} & \implies C \{ \{ Q \} \}
\end{align*}
\]

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

\textbf{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.
safe is a new primitive predicate in our inner-level logic.
Two-level logic (revisited)

- **Outer-level logic**
  \[
  \{\{P\}\} \ C \ \{\{Q\}\}
  \]
  \[\iff \forall M, LM \text{ such that } LM \models P \land LM \iso M \land LM \text{ safe}\]
  1. $C, M$ does not get stuck
  2. if $C, M \rightsquigarrow^* \text{ skip}, M'$
     then $\exists LM'. LM' \models Q \land LM' \iso M' \land LM' \text{ safe}$

- **Inner-level logic**
  \[
  \{P\} \ C \ \{Q\}
  \]
  \[\iff \forall M, LM \text{ such that } LM \models P \land LM \iso M\]
  1. $C, M$ does not get stuck
  2. if $C, M \rightsquigarrow^* \text{ skip}, M'$
     then $\exists LM'. LM' \models Q \land LM' \iso M'$
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 \text{ does not get stuck}
  
  2. \text{if } C, M \rightsquigarrow \ast \text{ skip, } M'
  
  \text{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 \text{ does not get stuck}
  
  2. \text{if } C, M \rightsquigarrow \ast \text{ skip, } M'
  
  \text{then } \exists LM'. \ LM' \models Q \land LM' \overset{\text{iso}}{\sim} M' \land LM'.h \text{ safe}
  \]
Frame rule

\[
\frac{\{P\} \ C \ \{Q\}}{\{P \ast R\} \ C \ \{Q \ast R\}} \quad FV(R) \cap \text{Mod}(C) = \emptyset
\]

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**

\[
\{\{P\}\} \ C \ \{\{Q\}\}
\]

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

1. \(C, M\) does not get stuck
2. if \(C, M \xrightarrow{\ast} \text{ skip}, M'\) 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 \(C, M \xrightarrow{\ast} \text{ skip}, M'\) then \(\exists LM'. LM' \models Q \land LM' \overset{\text{iso}}{\sim} M' \land LM'.h \text{ safe}\)
Baking the frame rule in

- **Outer-level logic**

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

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

  1. $C, M$ does not get stuck

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

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

- **Inner-level logic**

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

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

  1. $C, M$ does not get stuck

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

    then $\exists LM'. \ LM' \models Q \land LM' \uplus LM_f \overset{\text{iso}}{\sim} M' \land (LM' \uplus LM_f).h \text{ safe}$

C.-K. Hur, D. Dreyer, V. Vafeiadis

Separation Logic in the Presence of Garbage Collection
Proof rules & Examples
Logical entities

Words \( \overset{\text{def}}{=} \{ w \in \mathbb{Z} \} \)

Locs \( \overset{\text{def}}{=} \{ \ell_1, \ell_2, \ldots \} \)

LogPtrs \( \overset{\text{def}}{=} \{ \ell + i \mid \ell \in \text{Locs} \land i \in \mathbb{Z} \} \)

LogVals \( \overset{\text{def}}{=} \{ v \in \text{Words } \cup \text{LogPtrs} \} \)

LStores \( \overset{\text{def}}{=} \{ s \in \text{ProgVars } \rightarrow \text{LogVals} \} \)

LHeaps \( \overset{\text{def}}{=} \{ h \in \text{Locs } \rightarrow_{\text{fin}} \mathbb{N} \rightarrow_{\text{fin}} \text{LogVals} \} \)
Assertions

- **Outer-level assertions**

  \[ P ::= E \mid \text{logptr}(E) \mid \text{word}(E) \]

  \[ \mid E \leftrightarrow E \mid P \ast P \mid P \rightarrow P \]

  \[ \mid P \Rightarrow P \mid P \land P \mid P \lor P \mid \forall v. P \mid \exists v. P \]

- **Inner-level assertions**

  \[ P ::= \text{safe}(E) \]

  \[ \mid E \mid \text{logptr}(E) \mid \text{word}(E) \]

  \[ \mid E \leftrightarrow E \mid P \ast P \mid P \rightarrow P \]

  \[ \mid P \Rightarrow P \mid P \land P \mid P \lor P \mid \forall v. P \mid \exists v. P \]
Selected proof rules

\[
\begin{align*}
\{x = v \land E = E\} & \quad x := E \quad \{x = E[v/x]\} & \quad \text{(Assign)} \\
\{x = u \land E \leftrightarrow v\} & \quad x := \left[ E \right] \quad \{x = v \land E[u/x] \leftrightarrow v\} & \quad \text{(Read)} \\
\{E \leftrightarrow - \land \text{safe}(E')\} & \quad \left[ E \right] := E' \quad \{E \leftrightarrow E'\} & \quad \text{(Write)} \\
\{\text{true}\} & \quad x := \text{ALLOC}(n) \quad \{x \leftrightarrow n -, \ldots, -\} & \quad \text{(Alloc)}
\end{align*}
\]
Example 1: Array initialization

\[
x := \text{ALLOC}(n); \\
t := x + 4n; \\
\text{while } x < t \text{ do } \\
    [x] := 0; \\
    x := x + 4 \\
\text{od; } \\
x := x - 4n; \\
t := 0
\]
Example 1: Array initialization

\[
\begin{align*}
x := & \text{ALLOC}(n); \\
& \{ \text{\_} \} \text{C} \{ \text{\_} \} \\
& \{ \text{\_} \} \text{C} \{ \text{\_} \} \\
& \{ \text{\_} \} \text{C} \{ \text{\_} \} \\
& \{ \text{\_} \} \text{C} \{ \text{\_} \} \\
\end{align*}
\]

\[
\begin{align*}
\text{\_} & := \text{ALLOC}(n); \\
& \text{n times} \\
& ([x] := 0; x := x + 4); \ldots; ([x] := 0; x := x + 4) \\
& x := x - 4n
\end{align*}
\]
Example 1: Array initialization

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

\ x := \text{ALLOC}(n); \n
\{\{ x \rightarrow n \rightarrow, \ldots, \rightarrow \}\}

\underbrace{( [x] := 0; \ x := x + 4); \ \ldots; \ ([x] := 0; \ x := x + 4)}_{n \ \text{times}}

\ x := x - 4n

\{\{ x \rightarrow n \ 0, \ldots, 0 \}\}
Example 1: Array initialization

\[
\begin{align*}
\{ \{ \text{true} \} \} \\
\xrightarrow{n} \{ x \leftarrow n -, \ldots, - \} \\
\{ x \leftarrow n -, \ldots, - \land \text{safe}(x) \} \\
\text{n times} \\
([x] := 0; \ x := x + 4); \ldots; \ ([x] := 0; \ x := x + 4) \\
\xrightarrow{n} \{ x := x - 4n \} \\
\{ x \leftarrow n 0, \ldots, 0 \land \text{safe}(x) \} \\
\{\{ x \leftarrow n 0, \ldots, 0 \}\}
\end{align*}
\]
Example 1: Array initialization

\[
\begin{align*}
\{\text{true}\} & \quad C \\
\{P \land \text{safe}(V)\} & \quad C \quad \{Q \land \text{safe}(\text{Mod}(C))\} \\
\{P\} & \quad C \quad \{Q\}
\end{align*}
\]

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

\[
\begin{align*}
\{x & \mapsto n, \ldots, -\} \\
\{x & \mapsto n, \ldots, - \land \text{safe}(x)\}
\end{align*}
\]

\[n \text{ times}\]

\[
([x] := 0; \ x := x + 4); \ldots; ([x] := 0; \ x := x + 4)
\]

\[
\{x - 4n & \mapsto 0, \ldots, 0 \land \text{safe}(x - 4n)\}
\]

\[x := x - 4n\]

\[
\{x & \mapsto 0, \ldots, 0 \land \text{safe}(x)\}
\]

\[
\{x & \mapsto 0, \ldots, 0\}
\]
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.

\[
x := \text{ALLOC}(n);
\]
\[
t := x + 4n;
\]
\[
\text{while } x < t \text{ do}
\]
\[
[x] := 0;
\]
\[
    x := x + 4
\]
\[
\text{od;}
\]
\[
x := x - 4n;
\]
\[
t := 0
\]
Example 2: Add & Square

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

\[
i := i \times i; \quad i := 2 \times i + 1
\]
Example 2: Add & Square

\{\{ i = 2n + 1 \land j = 2m + 1 \}\}\}

\begin{align*}
i &:= (i + j - 2) \div 2; \\
i &:= i \times i; \ i := 2 \times i + 1
\end{align*}

\{\{ i = 2(n + m)^2 + 1 \land j = 2m + 1 \}\}\}
Example 2: Add & Square

\[
\begin{align*}
\{ & \{ i = 2n + 1 \land j = 2m + 1 \} \\
& \{ i = 2n + 1 \land j = 2m + 1 \land \text{word}(n, m) \} \\
& i := (i + j - 2) / 2; \\
& \{ i = n + m \land j = 2m + 1 \land \text{word}(n, m) \} \\
& i := i \times i; \ i := 2 \times i + 1 \\
& \{ i = 2(n + m)^2 + 1 \land j = 2m + 1 \} \\
& \{ i = 2(n + m)^2 + 1 \land j = 2m + 1 \land \text{safe}(i) \} \\
& \{ \{ i = 2(n + m)^2 + 1 \land j = 2m + 1 \} \}
\end{align*}
\]
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)
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:

\[
\begin{align*}
\{ x = v \land y = w \} \\
x & := x \oplus y; \quad y := x \oplus y; \quad x := x \oplus y \\
\{ x = w \land y = v \}
\end{align*}
\]