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

\[
\{P\} \; C \; \{Q\} \\
\iff \forall s, h \text{ such that } s, h \models P, \\
1. \; C, s, h \text{ does not get stuck} \\
2. \text{ if } C, s, h \leadsto^* \text{ skip, } s', h' \\
\text{ then } s', h' \models Q
\]

+ Separating Conjunction “∗”

\[
s, h \models P \ast Q \\
\iff \exists h_1, h_2. \; h = h_1 \uplus h_2 \land s, h_1 \models P \land s, h_2 \models Q
\]
Frame rule

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

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

GC safe →

\[ x := \text{ALLOC}(n); \]
\[ t := x + 4n; \]
\[ \text{while } x < t \text{ do} \]
\[ [x] := 0; \]
\[ x := x + 4 \]

GC unsafe →

\[ \text{od}; \]
\[ x := x - 4n; \]
\[ t := 0\]

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]:

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

\[
\begin{align*}
\{ \text{true} \} \\
x & := \text{new}(); \ [x] := 5; \ x := \text{null}; \\
\{ x = \text{null} \land \exists \ell. \ \ell \rightarrow 5 \} \\
\text{GC()} \\
\{ x = \text{null} \land \exists \ell. \ \ell \rightarrow 5 \}
\end{align*}
\]
Problem 1: Unreachable blocks may be reclaimed

Conundrum due to [Reynolds 2000]:

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

Approach by [Calcagno et al. 2003]:
Impose “monster-barring” syntactic restriction on assertions \( P \).
This triple is easy to validate, even if the GC relocates $x$:

$$\{ x \mapsto 7 \} \quad \text{GC()} \quad \{ x \mapsto 7 \}$$
This triple is hard to validate, because the GC could move $\ell$:

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

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

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

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

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

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 $\ell$:

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

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

- 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 $M$
Logical memory (adapted from [McCreight et al. 2007]):

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

\[ \ell \leftrightarrow 5 \]

\[ 0x80 \leftrightarrow 5 \]

\{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 \\
LM & \overset{iso}{\rightarrow} M \overset{GC}{\rightarrow} M' \overset{emp}{\rightarrow} \\
\{true\} \ & x := \text{new}(); \ [x] := 5; \ x := \text{null}; \\
& \{x = \text{null} \land \exists \ell. \ \ell \leftrightarrow 5\} \\
\text{GC()} \\
& \{x = \text{null} \land \exists \ell. \ \ell \leftrightarrow 5\}
\end{align*}
Logical memory (adapted from [McCreight et al. 2007])

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

\[
\begin{align*}
    x & \leftrightarrow \ell \star \ell \leftrightarrow 7 \\
    x & \leftrightarrow 0\times40 \star 0\times40 \leftrightarrow 7
\end{align*}
\]
Logical memory (adapted from [McCreight et al. 2007])

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

\[
\begin{align*}
\text{x} & \leftrightarrow \ell \ast \ell \leftrightarrow 7 \\
\text{LM} & \\
\text{x} & \leftrightarrow \ell \ast \ell \leftrightarrow 7 \\
\text{M} & \xrightarrow{\text{GC}} \text{M}' \\
\text{x} & \leftrightarrow 0\times60 \ast 0\times60 \leftrightarrow 7
\end{align*}
\]

\[ \begin{align*}
\{ \text{x} & \leftrightarrow \ell \ast \ell \leftrightarrow 7 \} & \xrightarrow{\text{GC}(\cdot)} \{ \text{x} & \leftrightarrow \ell \ast \ell \leftrightarrow 7 \}
\end{align*} \]
Semantics of Hoare triples with logical memories

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

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

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

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

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

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

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

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

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

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

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

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

\[ LM = (s, h) \]

\( v \text{ safe} : \) \( v \) is either a non-pointer word
   or a pointer to the head of an allocated block.

\( s \text{ safe} : \) all program variables in \( s \) contain safe values.

\( h \text{ safe} : \) 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

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

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

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

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

GC safe $\rightarrow$
\[
x := \text{ALLOC}(n);
\]
\[
t := x + 4n;
\]
\[
\text{while } x < t \text{ do}
\]
\[
[x] := 0;
\]
\[
x := x + 4
\]

GC unsafe $\rightarrow$
\[
\text{od;}
\]
\[
x := x - 4n;
\]
\[
t := 0
\]

GC safe $\rightarrow$
Two-level logic

**Outer-level logic**

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

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

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

**Inner-level logic**

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

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

1. \( C, M \) does not get stuck
2. If \( C, M \sim^* \text{skip}, M' \)
   then \( \exists LM'. LM' \models Q \land LM'^{\text{iso}} \sim M' \)
Towards an “inclusion” rule

Obviously unsound:

\[
\begin{array}{c}
\{P\} \ C \ \{Q\} \\
\hline
\{\{P\}\} \ C \ \{\{Q\}\}
\end{array}
\]
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 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 . . .

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

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

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

We want something like this . . .

\[
\begin{array}{c}
\{ P \land \text{mem is GC-safe} \} \quad C \quad \{ Q \land \text{mem is GC-safe} \} \\
\{\{ P \}\} \quad C \quad \{\{ Q \}\}
\end{array}
\]

. . . 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 . . .

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

. . . but how do we characterize \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

\[
\begin{array}{c}
\{P \land \text{safe}(V)\} \ C \ \{Q \land \text{safe}(\text{Mod}(C))\} \\
\{\{P\}\} \ C \ \{\{Q\}\}
\end{array}
\quad V \subseteq \text{ProgVars}
\]

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

- **Outer-level logic**

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

- **Inner-level logic**

  \[
  \{P\} C \ \{Q\} \\
  \iff \forall M, LM \text{ such that } LM \models P \land LM \isom M \\
  \begin{align*}
  &1. C, M \text{ does not get stuck} \\
  &2. \text{if } C, M \leadsto^* \text{ skip, } M' \\
  &\text{then } \exists LM'. LM' \models Q \land LM' \isom M'
  \end{align*}
  \]
Two-level logic (revisited)

- **Outer-level logic**

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

- **Inner-level logic**

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

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

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
 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 \( C, M \sim^\ast \text{skip}, M' \)
    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 \land LM.h \text{ safe}

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

- **Outer-level logic**

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

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

1. C, M does not get stuck
2. if \( C, M \isometric \star \text{skip}, M' \)
   then \( \exists LM'. LM' \models Q \land LM' \cup LM_f \isometric M' \land LM' \cup LM_f \text{ safe} \)

- **Inner-level logic**

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

\[ \iff \forall M, LM, LM_f \text{ such that } LM \models P \land LM \cup LM_f \isometric M \land (LM \cup LM_f).h \text{ safe} \]

1. C, M does not get stuck
2. if \( C, M \isometric \star \text{skip}, M' \)
   then \( \exists LM'. LM' \models Q \land LM' \cup LM_f \isometric M' \land (LM' \cup LM_f).h \text{ safe} \)
Proof rules & Examples
Logical entities

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

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

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

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

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

LHeaps \( \overset{\text{def}}{=} \{ h \in \text{Locs} \rightarrow_{\text{fin}} \mathbb{N} \rightarrow_{\text{fin}} \text{LogVals} \} \)
Outer-level assertions

\[ P ::= E \mid \text{logptr}(E) \mid \text{word}(E) \]
\[ \mid E \leftrightarrow E \mid P \star P \mid P \rightarrow P \]
\[ \mid P \Rightarrow P \mid P \land P \mid P \lor P \mid \forall v. P \mid \exists v. P \]

Inner-level assertions

\[ P ::= \text{safe}(E) \]
\[ \mid E \mid \text{logptr}(E) \mid \text{word}(E) \]
\[ \mid E \leftrightarrow E \mid P \star P \mid P \rightarrow P \]
\[ \mid P \Rightarrow P \mid P \land P \mid P \lor P \mid \forall v. P \mid \exists v. P \]
Selected proof rules

\[
\begin{align*}
\{x = v \land E = E\} & \quad x := E \{x = E[v/x]\} & \text{(Assign)} \\
\{x = u \land E \leftrightarrow v\} & \quad x := [E] \{x = v \land E[u/x] \leftrightarrow v\} & \text{(Read)} \\
\{E \leftrightarrow - \land \text{safe}(E')\} & \quad [E] := E' \{E \leftrightarrow E'\} & \text{(Write)} \\
n \geq 0 & \quad \{\text{true}\} x := \text{ALLOC}(n) \{\{x \leftrightarrow n -, \ldots, -\}\} & \text{(Alloc)}
\end{align*}
\]
Example 1: Array initialization

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

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

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

\[ n \text{ times} \]

\[ x := x - 4n \]
Example 1: Array initialization

\[
\begin{align*}
&\{\{\text{true}\}\}\quad x := \text{ALLOC}(n); \\
&\{\{x \mapsto n -, \ldots, -\}\}
\end{align*}
\]

\[
\begin{align*}
&\text{n times} \\
&([x] := 0; \ x := x + 4); \ldots; ([x] := 0; \ x := x + 4)
\end{align*}
\]

\[
\begin{align*}
&x := x - 4n \\
&\{\{x \mapsto n \ 0, \ldots, 0\}\}
\end{align*}
\]
Example 1: Array initialization

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

\[ \begin{array}{c}
\{P \land \text{safe}(V)\} \quad C \quad \{Q \land \text{safe} \left( \text{Mod} \left( C \right) \right)\} \\
\{\{P\}\} \quad C \quad \{\{Q\}\}
\end{array} \]

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

\[ \{\{x \leftrightarrow n -\ldots, -\}\}\]

\[ \{x \leftrightarrow n -\ldots, -\land \text{safe}(x)\} \]

\[ n \text{ times} \]

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

\[ x := x - 4n \]

\[ \{x \leftrightarrow n \ 0,\ldots, 0 \land \text{safe}(x)\} \]

\[ \{\{x \leftrightarrow n \ 0,\ldots, 0\}\} \]
Example 1: Array initialization

{true} {P ∧ safe(V)} C {Q ∧ safe(Mod(C))} {P} C {{Q}}

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

\[
\{ \{ x \leftarrow n, \ldots, n \} \} \\
\{ x \leftarrow n, \ldots, n \land \text{safe}(x) \}
\]

\[ n \text{ times} \]

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

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

\[ x := x - 4n \]

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

\[
\{ x \leftarrow 0, \ldots, 0 \}
\]
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.

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

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

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

\[i := i \times i; \quad i := 2 \times i + 1\]

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

\{
\{ i = 2n + 1 \land j = 2m + 1 \}\}
\{ i = 2n + 1 \land j = 2m + 1 \land \text{word}(n, m) \}\}
i := (i + j - 2) \div 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 \}\}
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:

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

C.-K. Hur, D. Dreyer, V. Vafeiadis
Separation Logic in the Presence of Garbage Collection
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:

\[
\begin{aligned}
\{ x = v \land y = w \} \\
x := x \text{ xor } y; \quad y := x \text{ xor } y; \quad x := x \text{ xor } y \\
\{ x = w \land y = v \}
\end{aligned}
\]