

Lecture 12 – Separation Logic (1)

AAA551: Programming Language Theory

Jihyeok Park



2026 Spring

- Weakest Liberal Precondition
 - Motivation
 - Weakest Liberal Precondition
 - Example – Factorial
 - Provability and Correctness of `wlp`
- Dafny
 - Example 1 – Factorial
 - Example 2 – Fibonacci
 - Example 3 – IndexOf

1. Motivation

2. IMP with Mutation

3. Separation Logic

1. Motivation

2. IMP with Mutation

3. Separation Logic

Consider mutable variables x and y :

$$\{x \mapsto 1, y \mapsto -\} [y] := 2 \{x \mapsto 1 \wedge y \mapsto 2\}$$

where $x \mapsto 1$ means that the variable x stores an address a that points to a heap location storing the value 1, and $[y] := 2$ is an assignment that updates the value stored at the heap location pointed to by y to 2.

Is it correct? **No!** The assignment $[y] := 2$ may change the value of x .

Consider the following state $\langle h, \rho \rangle$:

$$\begin{array}{ll} \text{Heap} & h = \{a \mapsto 1\} \\ \text{Environment} & \rho = \{x \mapsto a, y \mapsto a\} \end{array}$$

After the assignment $[y] := 2$, the state becomes $\langle h', \rho \rangle$:

$$\begin{array}{ll} \text{Heap} & h' = \{a \mapsto 2\} \\ \text{Environment} & \rho = \{x \mapsto a, y \mapsto a\} \end{array}$$

Thus, x points to a heap location that stores the value 2.

If we can **separate** the heaps for x and y , then we can prove the correctness of the triple:

$$\{x \mapsto 1 * y \mapsto -\} [y] := 2 \{x \mapsto 1 * y \mapsto 2\}$$

where $\phi * \phi'$ means that ϕ and ϕ' hold for **disjoint** parts of the heap.

Assume that the variables x and y are defined in disjoint parts of the heap, and x stores the value 1 while y stores an arbitrary value (denoted by $-$).

Then, after the assignment $[y] := 2$, the variable x still stores the value 1, and the variable y stores the value 2.

We call the logic that allows us to reason about disjointness of heap locations **Separation Logic**.

1. Motivation

2. IMP with Mutation

3. Separation Logic

Let's define a simple imperative language with mutation.

Expressions	e	$::= n \mid x \mid e + e \mid e * e \mid e < e \mid \text{true} \mid \text{false}$
Statements	s	$::= \text{skip} \mid x := e \mid s; s$ $\mid \text{if } e \text{ then } s \text{ else } s$ $\mid \text{while } e \text{ do } s$ $\mid x := \text{new}(e, \dots, e)$ $\mid x := [e]$ $\mid [e] := e$ $\mid \text{free}(e)$
Values	$v \in \mathbb{V}$	$::= n \mid \text{true} \mid \text{false}$
Addresses	$a \in \mathbb{A}$	$= \mathbb{Z}$
Environments	ρ	$\in \mathbb{X} \rightarrow \mathbb{V}$
Heaps	h	$\in \mathbb{A} \rightarrow \mathbb{V}$

We can use an integer n to represent both values and addresses.

Note that expressions do not lookup/update the heap.

There are two kinds of **configurations**:

- Nonterminal: $\langle s, (\rho, h) \rangle$
- Terminal: $\langle \rho, h \rangle$ or abort

Allocation:

$$\frac{a, \dots, (a + n - 1) \notin \text{dom}(h)}{\langle x := \text{new}(e_1, \dots, e_n), \rho, h \rangle \rightarrow \langle \rho[x \mapsto a], h[a \mapsto E[e_1]](\rho), \dots, (a + n - 1) \mapsto E[e_n]](\rho) \rangle}$$

Lookup:

$$\frac{E[e](\rho) = a \quad a \in \text{dom}(h)}{\langle x := [e], \rho, h \rangle \rightarrow \langle \rho[x \mapsto h(a)], h \rangle} \qquad \frac{E[e](\rho) \notin \text{dom}(h)}{\langle x := [e], \rho, h \rangle \rightarrow \text{abort}}$$

Mutation:

$$\frac{E[e_1](\rho) = a \quad a \in \text{dom}(h)}{\langle [e_1] := e_2, \rho, h \rangle \rightarrow \langle \rho, h[a \mapsto E[e_2]](\rho) \rangle} \qquad \frac{E[e_1](\rho) \notin \text{dom}(h)}{\langle [e_1] := e_2, \rho, h \rangle \rightarrow \text{abort}}$$

Deallocation:

$$\frac{E[e](\rho) = a \quad a \in \text{dom}(h)}{\langle \text{free}(e), \rho, h \rangle \rightarrow \langle \rho, h - \{a\} \rangle} \qquad \frac{E[e](\rho) \notin \text{dom}(h)}{\langle \text{free}(e), \rho, h \rangle \rightarrow \text{abort}}$$

We use the following notation:

- $(\sigma \rightarrow^* \sigma')$ – σ' is reachable from σ in a finite number of steps.
- $(\sigma \uparrow)$ – there exists an infinite sequence of steps starting from σ .
- $(h \perp h')$ – the heaps h and h' are disjoint:

$$h \perp h' \iff \text{dom}(h) \cap \text{dom}(h') = \emptyset$$

- $(h \cdot h')$ – the union of two disjoint heaps h and h' :

$$h \cdot h' = h \cup h'$$

only defined when $h \perp h'$.

1. Motivation

2. IMP with Mutation

3. Separation Logic

Let's extend the **assertion language** of Hoare Logic to reason about heaps.

$i, j \in \mathbf{LVar}$

$e ::= x \mid i \mid n \mid e + e \mid e * e$

$\phi ::= \mathbf{true} \mid \mathbf{false} \mid e = e \mid e < e$
| $\neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists i. \phi \mid \forall i. \phi$

| \mathbf{emp}

(empty heap)

| $e \mapsto e$

(singleton heap)

| $\phi * \phi$

(separating conjunction)

| $\phi \multimap \phi$

(separating implication)

$$\rho, h \models_I \text{emp} \iff h = \emptyset$$

$$\rho, h \models_I e \mapsto e' \iff h = \{E[[e]](\rho, I) \mapsto E[[e']](\rho, I)\}$$

$$\rho, h \models_I \phi * \phi' \iff \exists h_1, h_2. (h_1 \perp h_2) \wedge (h = h_1 \cdot h_2) \wedge (\rho, h_1 \models_I \phi) \wedge (\rho, h_2 \models_I \phi')$$

$$\rho, h \models_I \phi * \phi' \iff \forall h'. (h \perp h') \wedge (\rho, h' \models_I \phi) \implies (\rho, h \cdot h' \models_I \phi')$$

We use the following notation:

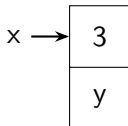
$$e \mapsto - \quad \triangleq \quad \exists v. e \mapsto v$$

$$e \hookrightarrow e' \quad \triangleq \quad (e \mapsto e') * \mathbf{true}$$

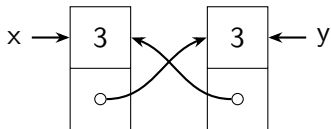
$$e \mapsto e_1, \dots, e_n \quad \triangleq \quad (e \mapsto e_1) * \dots * ((e + n - 1) \mapsto e_n)$$

$$e \hookrightarrow e_1, \dots, e_n \quad \triangleq \quad (e \mapsto e_1, \dots, e_n) * \mathbf{true}$$

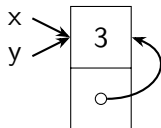
$x \mapsto 3, y$



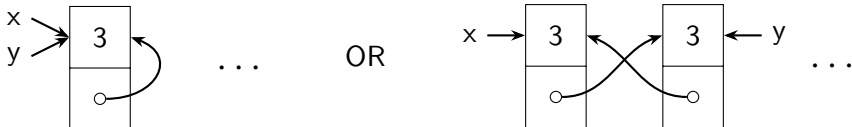
$$(x \mapsto 3, y) * (y \mapsto 3, x)$$



$$(x \mapsto 3, y) \wedge (y \mapsto 3, x)$$



$$(x \hookrightarrow 3, y) \wedge (y \hookrightarrow 3, x)$$



We call an assertion ϕ **pure** if it does not contain any heap predicate (i.e., emp , $e \mapsto e$, $\phi * \phi$, and $\phi \multimap \phi$).

Pure assertions only talk about the values of variables, and are independent of the heap.

For example, the following assertions are pure:

`true`

`false`

`x = 3`

`x < y + 1`

`$\neg(x < y) \vee (y < z)$`

`$\exists i.(i > 0) \wedge (x = i * i)$`

We call an assertion ϕ **intuitionistic** if it holds for a larger heap whenever it holds for a smaller heap:

$$\forall \rho, h, h'. (h \perp h') \wedge (\rho, h \models_I \phi) \implies (\rho, h \cdot h' \models_I \phi)$$

Intuitionistic assertions are **preserved** when we add more heap locations.

For example, the following assertions are intuitionistic:

(all pure assertions)	$e \mapsto e'$
$\phi_1 \wedge \phi_2$	$\phi_1 \vee \phi_2$
$\forall i. \phi_1$	$\exists i. \phi_1$
$\phi * \mathbf{true}$	$\mathbf{true} \multimap \phi$
$\phi_1 * \phi_2$	$\phi_1 \multimap \phi_2$

where ϕ_1 and ϕ_2 are intuitionistic but ϕ is any assertion.

We call an assertion ϕ **strictly exact** if it holds for exactly one heap:

$$\forall \rho, h, h'. (\rho, h \models_I \phi) \wedge (\rho, h' \models_I \phi) \implies (h = h')$$

All assertions built from \mapsto and $*$ are strictly exact.

We call an assertion ϕ **domain-exact** if it holds for heaps with the fixed domain:

$$\forall \rho, h, h'. (\rho, h \models_I \phi) \wedge (\rho, h' \models_I \phi) \implies (\text{dom}(h) = \text{dom}(h'))$$

All assertions built from \mapsto , $*$, and quantifiers are domain-exact.

$$\frac{}{\vdash \{x = i \wedge \mathbf{emp}\} x := \mathbf{new}(e_1, \dots, e_n) \{x \mapsto e_1[x \mapsto i], \dots, e_n[x \mapsto i]\}}$$

$$\frac{}{\vdash \{x = i \wedge e \mapsto j\} x := [e] \{x = j \wedge e[x \mapsto i] \mapsto j\}}$$

$$\frac{}{\vdash \{e_1 \mapsto -\} [e_1] := e_2 \{e_1 \mapsto e_2\}}$$

$$\frac{}{\vdash \{e \mapsto -\} \mathbf{free}(e) \{\mathbf{emp}\}}$$

The **frame rule** allows us to reason about the part of the heap that is not modified by a statement:

$$\text{FRAME} \frac{\vdash \{\phi\} s \{\phi'\} \quad \text{Modified}(s) \cap \text{Free}(\phi'') = \emptyset}{\vdash \{\phi * \phi''\} s \{\phi' * \phi''\}}$$

where $\text{Modified}(s)$ is the set of variables that may be modified by s and $\text{Free}(A)$ is the set of free variables in A .

Still other inference rules of Hoare Logic (e.g., consequence) are also valid in Separation Logic:

$$\text{CONSEQUENCE} \frac{\vdash \phi \Rightarrow \phi_1 \quad \vdash \{\phi_1\} s \{\phi'_1\} \quad \vdash \phi'_1 \Rightarrow \phi'}{\vdash \{\phi\} s \{\phi'\}}$$

```
{emp}
```

```
x := new(a, a);
```

```
y := new(b, b);
```

```
[x + 1] := y - x;
```

```
[y + 1] := x - y;
```

$\{\text{emp}\}$

$x := \text{new}(a, a);$

$\{x \mapsto a, a\}$

$y := \text{new}(b, b);$

$[x + 1] := y - x;$

$[y + 1] := x - y;$

```
{emp}
x := new(a, a);
{x ↦ a, a}
y := new(b, b);
{(x ↦ a, a) * (y ↦ b, b)}
[x + 1] := y - x;

[y + 1] := x - y;
```

$\{\text{emp}\}$

$x := \text{new}(a, a);$

$\{x \mapsto a, a\}$

$y := \text{new}(b, b);$

$\{(x \mapsto a, a) * (y \mapsto b, b)\}$

$[x + 1] := y - x;$

$\{(x \mapsto a, y - x) * (y \mapsto b, b)\}$

$[y + 1] := x - y;$

$\{\text{emp}\}$

$x := \text{new}(a, a);$

$\{x \mapsto a, a\}$

$y := \text{new}(b, b);$

$\{(x \mapsto a, a) * (y \mapsto b, b)\}$

$[x + 1] := y - x;$

$\{(x \mapsto a, y - x) * (y \mapsto b, b)\}$

$[y + 1] := x - y;$

$\{(x \mapsto a, y - x) * (y \mapsto b, x - y)\}$

$\{\text{emp}\}$

$x := \text{new}(a, a);$

$\{x \mapsto a, a\}$

$y := \text{new}(b, b);$

$\{(x \mapsto a, a) * (y \mapsto b, b)\}$

$[x + 1] := y - x;$

$\{(x \mapsto a, y - x) * (y \mapsto b, b)\}$

$[y + 1] := x - y;$

$\{(x \mapsto a, y - x) * (y \mapsto b, x - y)\}$

$\{\exists z. (x \mapsto a, z) * ((x + z) \mapsto b, -z)\}$

- Separation Logic (2)

Jihyeok Park

jihyeok_park@korea.ac.kr

<https://plrg.korea.ac.kr>