

Lecture 13 – Separation Logic (2)

AAA551: Programming Language Theory

Jihyeok Park



2026 Spring

Assertions for separation logic are defined as follows:

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

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

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

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

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

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

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

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

1. Inference Rules for Global and Backward Reasoning

- Inference Rules (Local)

- Inference Rules (Global)

- Inference Rules (Backward)

2. Lists

- Example

3. Trees

1. Inference Rules for Global and Backward Reasoning

Inference Rules (Local)

Inference Rules (Global)

Inference Rules (Backward)

2. Lists

Example

3. Trees

We learned the following inference rules for **local reasoning**:

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

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

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

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

Using the **frame rule**, we can define inference rules for **global reasoning**:

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

Using **separating implication** ($*$), we can derive **backward reasoning**.

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

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

The **global** inference rules for **mutation** and **deallocation** can be derived:

$$\frac{}{\vdash \{(e_1 \mapsto -) * \phi\} [e_1] := e_2 \{(e_1 \mapsto e_2) * \phi\}}$$

$$\frac{}{\vdash \{(e \mapsto -) * \phi\} \text{free}(e) \{\phi\}}$$

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

The **global** inference rules for **allocation** can be derived:

$$\frac{}{\vdash \{ \phi \} \ x := \mathbf{new}(e_1, \dots, e_n) \ \{ \exists i. (x \mapsto e_1[x \mapsto i], \dots, e_n[x \mapsto i] * \phi[x \mapsto i]) \}}$$

where i is a fresh (logical) variable not appearing in ϕ .

For example,

$$\frac{}{\vdash \{x \mapsto 10\} \ x := \mathbf{new}(x + 1) \ \{ \exists i. (x \mapsto i + 1) * (i \mapsto 10) \}}$$

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

The **global** inference rules for **lookup** can be derived:

$$\frac{}{\vdash \{(e \mapsto j) * \phi\} \ x := [e] \ \{\exists i. (x = j \wedge e[x \mapsto i] \mapsto j * \phi[x \mapsto i])\}}$$

where i and j are fresh (logical) variables not appearing in ϕ .

For example,

$$\frac{}{\vdash \{(x + 1 \mapsto 10) * (x \mapsto 20)\} \ x := [x + 1] \ \{\exists i. (x = 10) \wedge (i + 1 \mapsto 10) * (i \mapsto 20)\}}$$

$$\frac{}{\vdash \{(e_1 \mapsto -) * \phi\} [e_1] := e_2 \{(e_1 \mapsto e_2) * \phi\}}$$

The **backward** inference rules for **mutation** can be derived:

$$\frac{}{\vdash \{(e_1 \mapsto -) * ((e_1 \mapsto e_2) * \phi)\} [e_1] := e_2 \{\phi\}}$$

The **global** inference rules for **deallocation** is already **backward**:

$$\frac{}{\vdash \{(e \mapsto -) * \phi\} \text{free}(e) \{\phi\}}$$

$$\frac{}{\vdash \{\phi\} x := \mathbf{new}(e_1, \dots, e_n) \{\exists i. (x \mapsto e_1[x \mapsto i], \dots, e_n[x \mapsto i]) * \phi[x \mapsto i]\}}$$

where i is a fresh (logical) variable not appearing in ϕ .

The **backward** inference rules for **allocation** can be derived:

$$\frac{}{\vdash \{\forall i. (i \mapsto e_1, \dots, e_n) \rightarrow \phi[x \mapsto i]\} x := \mathbf{new}(e_1, \dots, e_n) \{\phi\}}$$

where i is a fresh variable not appearing in ϕ .

For example,

$$\frac{}{\vdash \{\forall i. (i \mapsto x + 1) \rightarrow (i \mapsto 11)\} x := \mathbf{new}(x + 1) \{x \mapsto 11\}}$$

which is equivalent to

$$\frac{}{\vdash \{x = 10 \wedge \mathbf{emp}\} x := \mathbf{new}(x + 1) \{x \mapsto 11\}}$$

The **backward** inference rules for **lookup** can be derived:

$$\frac{}{\vdash \{\exists i. (e \mapsto i) * ((e \mapsto i) -* \phi[x \mapsto i])\} x := [e] \{\phi\}}$$

where i is a fresh (logical) variable not appearing in ϕ .

which is equivalent to

$$\frac{}{\vdash \{\exists i. (e \hookrightarrow i) \wedge \phi[x \mapsto i]\} x := [e] \{\phi\}}$$

For example,

$$\frac{}{\vdash \{\exists i. ((x + 1) \hookrightarrow i) \wedge (i \mapsto 10)\} x := [x + 1] \{x \mapsto 10\}}$$

1. Inference Rules for Global and Backward Reasoning

Inference Rules (Local)

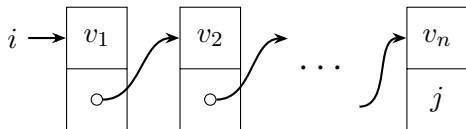
Inference Rules (Global)

Inference Rules (Backward)

2. Lists

Example

3. Trees



We can define a predicate $\text{list}(\alpha, i, j)$ that represents a **linked list segment** from i to j with the contents α :

$$\text{list}(\epsilon, i, j) \quad \triangleq \quad \text{emp} \wedge (i = j)$$

$$\text{list}(v :: \alpha, i, j) \quad \triangleq \quad \exists k. (i \mapsto v, k) * \text{list}(\alpha, k, j)$$

We use $\alpha \cdot \beta$ and α^R to denote the concatenation and the reverse.

We can define a predicate $\text{list}(\alpha, i)$ that represents a **linked list** from i to nil with the contents α :

$$\text{list}(\alpha, i) \quad \triangleq \quad \text{list}(\alpha, i, \text{nil})$$

```

{list( $\alpha$ ,  $x$ )}
y := nil;
{list( $\alpha$ ,  $x$ ) * ( $y = \text{nil} \wedge \text{emp}$ )}
while (x != nil) @[ $\exists \beta, \gamma. \text{list}(\beta, x) * \text{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma$ ] {
  { $\exists \beta', \gamma. \text{list}(v :: \beta', x) * \text{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma$ }
  { $\exists i. (x \mapsto v, i) * \text{list}(\beta', i) * \text{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)$ }
  z := [x + 1];
  {(x  $\mapsto$  v, z) * list( $\beta'$ , z) * list( $\gamma$ , y)  $\wedge$   $\alpha^R = \beta'^R \cdot (v :: \gamma)$ }
  [x + 1] := y;
  {(x  $\mapsto$  v, y) * list( $\beta'$ , z) * list( $\gamma$ , y)  $\wedge$   $\alpha^R = \beta'^R \cdot (v :: \gamma)$ }
  {list( $\beta'$ , z) * list( $v :: \gamma$ , x)  $\wedge$   $\alpha^R = \beta'^R \cdot (v :: \gamma)$ }
  x, y := z, x;
  {list( $\beta'$ , x) * list( $v :: \gamma$ , y)  $\wedge$   $\alpha^R = \beta'^R \cdot (v :: \gamma)$ }
  { $\exists \beta, \gamma. \text{list}(\beta, x) * \text{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma$ }
}
{x = nil  $\wedge$   $\exists \beta, \gamma. \text{list}(\beta, x) * \text{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma$ }
{emp * list( $\gamma$ , y)  $\wedge$   $\alpha^R = \text{nil}^R \cdot \gamma$ }
{list( $\alpha^R$ , y)}

```

1. Inference Rules for Global and Backward Reasoning

Inference Rules (Local)

Inference Rules (Global)

Inference Rules (Backward)

2. Lists

Example

3. Trees

We can define a predicate $\text{tree}(\tau, i)$ that represents a **binary tree** rooted at i with the contents τ :

$$\text{tree}(v_0, i) \triangleq \text{emp} \wedge (i = v_0)$$

$$\text{tree}(\tau_1 \cdot \tau_2, i) \triangleq \exists l, r. (i \mapsto l, r) * \text{tree}(\tau_1, l) * \text{tree}(\tau_2, r)$$

where v_0 is an atomic value that cannot be used as a pointer, and $\tau_1 \cdot \tau_2$ denotes a binary tree with the left and the right sub-trees τ_1 and τ_2 .

The predicate $\text{isatom}(v)$ holds if v is an atomic value.

```

copytree(x, y) @[tree( $\tau$ , x)] {
  if (isatom(x)) {
    {emp  $\wedge$  ( $x = \tau$ )}
    y := x;
    {(emp  $\wedge$  ( $x = \tau$ )) * (emp  $\wedge$  ( $y = \tau$ ))}
  } else {
    { $\exists l, r. (x \mapsto l, r) * \text{tree}(\tau_1, l) * \text{tree}(\tau_2, r) \wedge \tau = \tau_1 \cdot \tau_2$ }
    lx, rx := [x], [x + 1];
    {(x  $\mapsto$  lx, rx) * tree( $\tau_1$ , lx) * tree( $\tau_2$ , rx)  $\wedge$   $\tau = \tau_1 \cdot \tau_2$ }
    copytree(lx, ly); copytree(rx, ry);
    {(x  $\mapsto$  lx, rx) * tree( $\tau_1$ , lx) * tree( $\tau_2$ , rx)
      * tree( $\tau_1$ , ly) * tree( $\tau_2$ , ry)  $\wedge$   $\tau = \tau_1 \cdot \tau_2$ }
    y := new(ly, ry);
    {(x  $\mapsto$  lx, rx) * tree( $\tau_1$ , lx) * tree( $\tau_2$ , rx)
      (y  $\mapsto$  ly, ry) * tree( $\tau_1$ , ly) * tree( $\tau_2$ , ry)  $\wedge$   $\tau = \tau_1 \cdot \tau_2$ }
  }
} @[tree( $\tau$ , x) * tree( $\tau$ , y)]

```

- Simply-Typed Lambda Calculus

Jihyeok Park
jihyeok_park@korea.ac.kr
<https://plrg.korea.ac.kr>