Lecture 13 – Separation Logic (2)

AAA551: Programming Language Theory

Jihyeok Park

2026 Spring

Recall

Assertions for separation logic are defined as follows:

$$\sigma, h \models_I \texttt{emp} \iff h = \varnothing$$

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

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

$$\sigma, h \models_I \phi \mathrel{-\!\!*} \phi' \iff \forall h'.\ (h \perp h') \wedge (\sigma, h' \models_I \phi) \implies (\sigma, h \cdot h' \models_I \phi')$$

$$\begin{aligned}
e \mapsto - &\triangleq \exists v.\ e \mapsto v \\
e \hookrightarrow e' &\triangleq (e \mapsto e') * \texttt{true} \\
e \mapsto e_1, \ldots, e_n &\triangleq (e \mapsto e_1) * \cdots * ((e + n - 1) \mapsto e_n) \\
e \hookrightarrow e_1, \ldots, e_n &\triangleq (e \mapsto e_1, \ldots, e_n) * \texttt{true}
\end{aligned}$$

Contents

Contents

Inference Rules (Local)

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

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

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

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

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

Inference Rules (Local)

PLRG

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

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

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

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

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

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

$$\textsc{Frame}\ \frac{\vdash \{\phi\}\ s\ \{\phi'\} \qquad \mathsf{Modified}(s) \cap \mathsf{Free}(\phi'') = \varnothing}{\vdash \{\phi * \phi''\}\ s\ \{\phi' * \phi''\}}$$

Inference Rules (Local)

PLRG

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

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

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

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

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

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

$$\textsc{Frame}\ \frac{\vdash \{\phi\}\ s\ \{\phi'\} \qquad \mathsf{Modified}(s) \cap \mathsf{Free}(\phi'') = \varnothing}{\vdash \{\phi * \phi''\}\ s\ \{\phi' * \phi''\}}$$

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

Inference Rules (Global) – Mutation / Deallocation

PLRG

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

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

Inference Rules (Global) – Mutation / Deallocation

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

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

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

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

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

Inference Rules (Global) – Allocation

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

Inference Rules (Global) – Allocation

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

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

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

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

Inference Rules (Global) – Allocation

PLRG

$$\frac{}{\vdash \{x = i \land \texttt{emp}\}\ x\ \texttt{:=}\ \texttt{new}(e_1, \ldots, e_n)\ \{x \mapsto e_1[x \mapsto i], \ldots, e_n[x \mapsto i]\}}$$

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

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

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

For example,

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

Inference Rules (Global) – Lookup

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

Inference Rules (Global) – Lookup

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

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

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

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

Inference Rules (Global) – Lookup

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

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

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

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

For example,

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

Inference Rules (Backward) – Mutation / Dealloc.

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

Inference Rules (Backward) – Mutation / Dealloc.

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

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

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

Inference Rules (Backward) – Mutation / Dealloc.

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

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

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

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

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

Inference Rules (Backward) – Allocation

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

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

Inference Rules (Backward) – Allocation

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

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

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

$$\frac{}{\vdash \{\forall i.\ (i \mapsto e_1, \ldots, e_n) \mathrel{-\!\!*} \phi[x \mapsto i]\}\ x\ \texttt{:=}\ \texttt{new}(e_1, \ldots, e_n)\ \{\phi\}}$$

where i is a fresh variable not appearing in ϕ.

Inference Rules (Backward) – Allocation

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

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

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

$$\frac{}{\vdash \{\forall i.\ (i \mapsto e_1, \ldots, e_n) \mathrel{-\!\!*} \phi[x \mapsto i]\}\ x\ \texttt{:=}\ \texttt{new}(e_1, \ldots, e_n)\ \{\phi\}}$$

where i is a fresh variable not appearing in ϕ.

For example,

$$\frac{}{\vdash \{\forall i.\ (i \mapsto x + 1) \mathrel{-\!\!*} (i \mapsto 11)\}\ x\ \texttt{:=}\ \texttt{new}(x + 1)\ \{x \mapsto 11\}}$$

Inference Rules (Backward) – Allocation

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

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

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

$$\frac{}{\vdash \{\forall i.\ (i \mapsto e_1, \ldots, e_n) \mathrel{-\!\!*} \phi[x \mapsto i]\}\ x\ \texttt{:=}\ \texttt{new}(e_1, \ldots, e_n)\ \{\phi\}}$$

where i is a fresh variable not appearing in ϕ.

For example,

$$\frac{}{\vdash \{\forall i.\ (i \mapsto x + 1) \mathrel{-\!\!*} (i \mapsto 11)\}\ x\ \texttt{:=}\ \texttt{new}(x + 1)\ \{x \mapsto 11\}}$$

which is equivalent to

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

Inference Rules (Backward) – Lookup

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

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

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

Inference Rules (Backward) – Lookup

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

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

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

which is equivalent to

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

Inference Rules (Backward) – Lookup

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

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

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

which is equivalent to

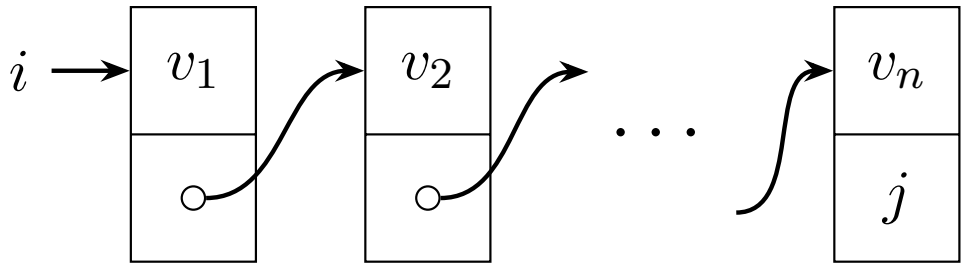
$$\frac{}{\vdash \{\exists i.\ (e \hookrightarrow i) \wedge \phi[x \mapsto i])\}\ x\ \texttt{:=}\ [e]\ \{\phi\}}$$

For example,

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

Contents

Lists



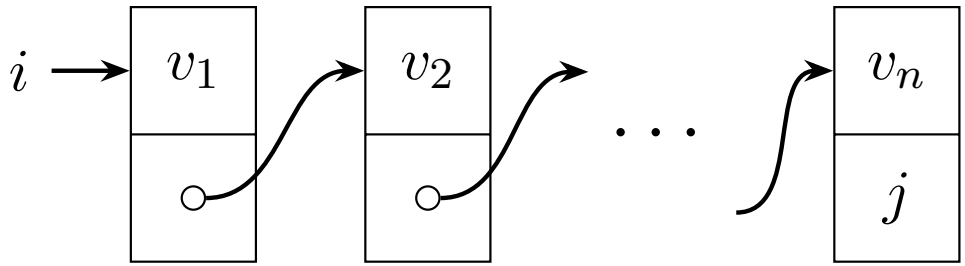
We can define a predicate $\texttt{list}(\alpha, i, j)$ that represents a **linked list segment** from i to j with the contents α:

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

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

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

Lists



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

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

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

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

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

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

Lists – Example

PLRG

$\{\texttt{list}(\alpha, x)\}$

```
y := nil;

while (x != nil) @[∃β, γ. list(β, x) * list(γ, y) ∧ α^R = β^R · γ] {

    z := [x + 1];

    [x + 1] := y;

    x, y := z, x;

}
```

Lists – Example

PLRG

$\{\texttt{list}(\alpha, x)\}$

```
y := nil;
```

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

```
while (x != nil) @[∃β, γ. list(β, x) * list(γ, y) ∧ α^R = β^R · γ] {


    z := [x + 1];


    [x + 1] := y;


    x, y := z, x;


}
```

Lists – Example

PLRG

$\{\texttt{list}(\alpha, x)\}$

```
y := nil;
```

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

```
while (x != nil) @[∃β, γ. list(β, x) * list(γ, y) ∧ α^R = β^R · γ] {
```

$\{\exists \beta', \gamma.\ \texttt{list}(v :: \beta', x) * \texttt{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma\}$

```
    z := [x + 1];

    [x + 1] := y;

    x, y := z, x;

}
```

Lists – Example

PLRG

$\{\texttt{list}(\alpha, x)\}$

```
y := nil;
```

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

```
while (x != nil) @[∃β, γ. list(β, x) * list(γ, y) ∧ α^R = β^R · γ] {
```

$\{\exists \beta', \gamma.\ \texttt{list}(v :: \beta', x) * \texttt{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma\}$
$\{\exists i.\ (x \mapsto v, i) * \texttt{list}(\beta', i) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

```
    z := [x + 1];


    [x + 1] := y;



    x, y := z, x;



}
```

Lists – Example

PLRG

$\{\texttt{list}(\alpha, x)\}$

`y := nil;`

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

`while (x != nil)` $@[\exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma]$ `{`

$\{\exists \beta', \gamma.\ \texttt{list}(v :: \beta', x) * \texttt{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma\}$
$\{\exists i.\ (x \mapsto v, i) * \texttt{list}(\beta', i) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`z := [x + 1];`

$\{(x \mapsto v, z) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`[x + 1] := y;`

`x, y := z, x;`

`}`

Lists – Example

PLRG

$\{\texttt{list}(\alpha, x)\}$

`y := nil;`

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

`while (x != nil)` $@[\exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma]$ `{`

$\{\exists \beta', \gamma.\ \texttt{list}(v :: \beta', x) * \texttt{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma\}$
$\{\exists i.\ (x \mapsto v, i) * \texttt{list}(\beta', i) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`z := [x + 1];`

$\{(x \mapsto v, z) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`[x + 1] := y;`

$\{(x \mapsto v, y) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`x, y := z, x;`

`}`

Lists – Example

$\{\texttt{list}(\alpha, x)\}$

```
y := nil;
```

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

```
while (x != nil) @[∃β, γ. list(β, x) * list(γ, y) ∧ α^R = β^R · γ] {
```

$\{\exists \beta', \gamma.\ \texttt{list}(v :: \beta', x) * \texttt{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma\}$
$\{\exists i.\ (x \mapsto v, i) * \texttt{list}(\beta', i) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

```
    z := [x + 1];
```

$\{(x \mapsto v, z) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

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

$\{(x \mapsto v, y) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$
$\{\texttt{list}(\beta', z) * \texttt{list}(v :: \gamma, x) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

```
    x, y := z, x;



}
```

Lists – Example

$\{\texttt{list}(\alpha, x)\}$

```
y := nil;
```

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

```
while (x != nil) @[∃β, γ. list(β, x) * list(γ, y) ∧ α^R = β^R · γ] {
```

$\{\exists \beta', \gamma.\ \texttt{list}(v :: \beta', x) * \texttt{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma\}$
$\{\exists i.\ (x \mapsto v, i) * \texttt{list}(\beta', i) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

```
    z := [x + 1];
```

$\{(x \mapsto v, z) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

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

$\{(x \mapsto v, y) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$
$\{\texttt{list}(\beta', z) * \texttt{list}(v :: \gamma, x) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

```
    x, y := z, x;
```

$\{\texttt{list}(\beta', x) * \texttt{list}(v :: \gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

```
}
```

Lists – Example

$\{\texttt{list}(\alpha, x)\}$

`y := nil;`

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

`while (x != nil)` $@[\exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma]$ `{`

$\{\exists \beta', \gamma.\ \texttt{list}(v :: \beta', x) * \texttt{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma\}$
$\{\exists i.\ (x \mapsto v, i) * \texttt{list}(\beta', i) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`z := [x + 1];`

$\{(x \mapsto v, z) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`[x + 1] := y;`

$\{(x \mapsto v, y) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$
$\{\texttt{list}(\beta', z) * \texttt{list}(v :: \gamma, x) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`x, y := z, x;`

$\{\texttt{list}(\beta', x) * \texttt{list}(v :: \gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$
$\{\exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma\}$

`}`

Lists – Example

$\{\texttt{list}(\alpha, x)\}$

```
y := nil;
```

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

```
while (x != nil) @[∃β, γ. list(β, x) * list(γ, y) ∧ α^R = β^R · γ] {
```

$\{\exists \beta', \gamma.\ \texttt{list}(v :: \beta', x) * \texttt{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma\}$
$\{\exists i.\ (x \mapsto v, i) * \texttt{list}(\beta', i) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

```
    z := [x + 1];
```

$\{(x \mapsto v, z) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

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

$\{(x \mapsto v, y) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$
$\{\texttt{list}(\beta', z) * \texttt{list}(v :: \gamma, x) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

```
    x, y := z, x;
```

$\{\texttt{list}(\beta', x) * \texttt{list}(v :: \gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$
$\{\exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma\}$

```
}
```

$\{x = \texttt{nil} \wedge \exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma\}$

Lists – Example

$\{\texttt{list}(\alpha, x)\}$

```
y := nil;
```

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

```
while (x != nil) @[∃β, γ. list(β, x) * list(γ, y) ∧ α^R = β^R · γ] {
```

$\{\exists \beta', \gamma.\ \texttt{list}(v :: \beta', x) * \texttt{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma\}$
$\{\exists i.\ (x \mapsto v, i) * \texttt{list}(\beta', i) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

```
    z := [x + 1];
```

$\{(x \mapsto v, z) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

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

$\{(x \mapsto v, y) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$
$\{\texttt{list}(\beta', z) * \texttt{list}(v :: \gamma, x) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

```
    x, y := z, x;
```

$\{\texttt{list}(\beta', x) * \texttt{list}(v :: \gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$
$\{\exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma\}$

```
}
```

$\{x = \texttt{nil} \wedge \exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma\}$
$\{\texttt{emp} * \texttt{list}(\gamma, y) \wedge \alpha^R = \texttt{nil}^R \cdot \gamma\}$

Lists – Example

$\{\texttt{list}(\alpha, x)\}$

`y := nil;`

$\{\texttt{list}(\alpha, x) * (y = \texttt{nil} \wedge \texttt{emp})\}$

`while (x != nil)` $@[\exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma]$ `{`

$\{\exists \beta', \gamma.\ \texttt{list}(v :: \beta', x) * \texttt{list}(\gamma, y) \wedge \alpha^R = (v :: \beta')^R \cdot \gamma\}$
$\{\exists i.\ (x \mapsto v, i) * \texttt{list}(\beta', i) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`z := [x + 1];`

$\{(x \mapsto v, z) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`[x + 1] := y;`

$\{(x \mapsto v, y) * \texttt{list}(\beta', z) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$
$\{\texttt{list}(\beta', z) * \texttt{list}(v :: \gamma, x) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$

`x, y := z, x;`

$\{\texttt{list}(\beta', x) * \texttt{list}(v :: \gamma, y) \wedge \alpha^R = \beta'^R \cdot (v :: \gamma)\}$
$\{\exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma\}$

`}`

$\{x = \texttt{nil} \wedge \exists \beta, \gamma.\ \texttt{list}(\beta, x) * \texttt{list}(\gamma, y) \wedge \alpha^R = \beta^R \cdot \gamma\}$
$\{\texttt{emp} * \texttt{list}(\gamma, y) \wedge \alpha^R = \texttt{nil}^R \cdot \gamma\}$
$\{\texttt{list}(\alpha^R, y)\}$

Contents

Trees

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

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

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

where v_0 is 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 $\texttt{isatom}(v)$ holds if v is an atomic value.

Trees – Example

```
copytree(x, y) @[tree(τ, x)] {
    if (isatom(x)) {

        y := x;

    } else {

        lx, rx := [x], [x + 1];

        copytree(lx, ly); copytree(rx, ry);


        y := new(ly, ry);


    }
} @[tree(τ, x) * tree(τ, y)]
```

Trees – Example

```
copytree(x, y) @[tree(τ, x)] {
    if (isatom(x)) {
        {emp ∧ (x = τ)}
        y := x;

    } else {

        lx, rx := [x], [x + 1];

        copytree(lx, ly); copytree(rx, ry);


        y := new(ly, ry);


    }
} @[tree(τ, x) ∗ tree(τ, y)]
```

Trees – Example

PLRG

```
copytree(x, y) @[tree(τ, x)] {
    if (isatom(x)) {
        {emp ∧ (x = τ)}
        y := x;
        {(emp ∧ (x = τ)) ∗ (emp ∧ (y = τ))}
    } else {

        lx, rx := [x], [x + 1];

        copytree(lx, ly); copytree(rx, ry);

        y := new(ly, ry);

    }
} @[tree(τ, x) ∗ tree(τ, y)]
```

Trees – Example

PLRG

```
copytree(x, y) @[tree(τ, x)] {
    if (isatom(x)) {
        {emp ∧ (x = τ)}
        y := x;
        {(emp ∧ (x = τ)) ∗ (emp ∧ (y = τ))}
    } else {
        {∃l, r. (x ↦ l, r) ∗ tree(τ1, l) ∗ tree(τ2, r) ∧ τ = τ1 · τ2}
        lx, rx := [x], [x + 1];

        copytree(lx, ly); copytree(rx, ry);

        y := new(ly, ry);

    }
} @[tree(τ, x) ∗ tree(τ, y)]
```

Trees – Example

PLRG

```
copytree(x, y) @[tree(τ, x)] {
    if (isatom(x)) {
        {emp ∧ (x = τ)}
        y := x;
        {(emp ∧ (x = τ)) ∗ (emp ∧ (y = τ))}
    } else {
        {∃l, r. (x ↦ l, r) ∗ tree(τ1, l) ∗ tree(τ2, r) ∧ τ = τ1 · τ2}
        lx, rx := [x], [x + 1];
        {(x ↦ lx, rx) ∗ tree(τ1, lx) ∗ tree(τ2, rx) ∧ τ = τ1 · τ2}
        copytree(lx, ly); copytree(rx, ry);


        y := new(ly, ry);


    }
} @[tree(τ, x) ∗ tree(τ, y)]
```

Trees – Example

PLRG

```
copytree(x, y) @[tree(τ, x)] {
    if (isatom(x)) {
        {emp ∧ (x = τ)}
        y := x;
        {(emp ∧ (x = τ)) ∗ (emp ∧ (y = τ))}
    } else {
        {∃l, r. (x ↦ l, r) ∗ tree(τ1, l) ∗ tree(τ2, r) ∧ τ = τ1 · τ2}
        lx, rx := [x], [x + 1];
        {(x ↦ lx, rx) ∗ tree(τ1, lx) ∗ tree(τ2, rx) ∧ τ = τ1 · τ2}
        copytree(lx, ly); copytree(rx, ry);
        {(x ↦ lx, rx) ∗ tree(τ1, lx) ∗ tree(τ2, rx)
                      ∗ tree(τ1, ly) ∗ tree(τ2, ry) ∧ τ = τ1 · τ2}
        y := new(ly, ry);
    }
} @[tree(τ, x) ∗ tree(τ, y)]
```

Trees – Example

PLRG

```
copytree(x, y) @[tree(τ, x)] {
    if (isatom(x)) {
        {emp ∧ (x = τ)}
        y := x;
        {(emp ∧ (x = τ)) ∗ (emp ∧ (y = τ))}
    } else {
        {∃l, r. (x ↦ l, r) ∗ tree(τ1, l) ∗ tree(τ2, r) ∧ τ = τ1 · τ2}
        lx, rx := [x], [x + 1];
        {(x ↦ lx, rx) ∗ tree(τ1, lx) ∗ tree(τ2, rx) ∧ τ = τ1 · τ2}
        copytree(lx, ly); copytree(rx, ry);
        {(x ↦ lx, rx) ∗ tree(τ1, lx) ∗ tree(τ2, rx)
                      ∗ tree(τ1, ly) ∗ tree(τ2, ry) ∧ τ = τ1 · τ2}
        y := new(ly, ry);
        {(x ↦ lx, rx) ∗ tree(τ1, lx) ∗ tree(τ2, rx)
         (y ↦ ly, ry) ∗ tree(τ1, ly) ∗ tree(τ2, ry) ∧ τ = τ1 · τ2}
    }
} @[tree(τ, x) ∗ tree(τ, y)]
```

Next Lecture

- Simply-Typed Lambda Calculus

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