Structuring the Synthesis of Heap-Manipulating Programs

Nadia Polikarpova
Ilya Sergey

Curry-Howard Correspondence

- Type Theories are Proofs Systems
    - Types are Propositions
    - Programs are Proofs
- Hence, Proof Search is Program Synthesis
- Separation Logic is a Type Theory *of state*

This Work

Program Synthesis
using
Separation Logic

Let’s *swap* values of two *distinct* pointers

Let's *swap* values of two *distinct* pointers

$x \mapsto$ a $\qquad y \mapsto$ b

Let's *swap* values of two *distinct* pointers

$x \mapsto b$ $\quad\quad$ $y \mapsto a$

```
void swap(loc x, loc y)
```

“separately”

$$\{\ x \mapsto a \ast y \mapsto b\ \}$$

```
void swap(loc x, loc y)
```

$$\{\ x \mapsto b \ast y \mapsto a\ \}$$

Peter W. O'Hearn, John C. Reynolds, Hongseok Yang:
Local Reasoning about Programs that Alter Data Structures. CSL 2001

$$\{\ x \mapsto \boxed{a} * y \mapsto b\ \}$$

??

$$\{\ x \mapsto b * y \mapsto \boxed{a}\ \}$$

```
let a2 = *x;
```

$\{\ x \mapsto a2 * y \mapsto \boxed{b}\ \}$

??

$\{\ x \mapsto \boxed{b} * y \mapsto a2\ \}$

```
let a2 = *x;
let b2 = *y;
```

$$\{ x \mapsto a2 * y \mapsto b2 \}$$

??

$$\{ x \mapsto b2 * y \mapsto a2 \}$$

```
let a2 = *x;
let b2 = *y;
*x = b2;
```

$\{ x \mapsto b2 * y \mapsto b2 \}$

??

$\{ x \mapsto b2 * y \mapsto a2 \}$

```
let a2 = *x;
let b2 = *y;
*x = b2;
*y = a2;
```

$\{ x \mapsto b2 * y \mapsto a2 \}$

??

$\{ x \mapsto b2 * y \mapsto a2 \}$

```
let a2 = *x;
let b2 = *y;
*x = b2;
*y = a2;
```

$\{\ x \mapsto b2 * y \mapsto a2\ \}$

??

$\{\ x \mapsto b2 * y \mapsto a2\ \}$

$x \mapsto b2 * y \mapsto a2 \vdash x \mapsto b2 * y \mapsto a2$

```
let a2 = *x;
let b2 = *y;
*x = b2;
*y = a2;
```

$$\{\ x \mapsto b2 \ast y \mapsto a2\ \}$$

??

$$\{\ x \mapsto b2 \ast y \mapsto a2\ \}$$

$$x \mapsto b2 \ast y \mapsto a2 \ \vdash\ x \mapsto b2 \ast y \mapsto a2$$ ✔

```
void swap(loc x, loc y) {
    let a2 = *x;
    let b2 = *y;
    *x = b2;
    *y = a2;
}
```

Reasoning with Symbolic Heaps

Symbolic Heap Entailment

$$P \vdash Q$$

Any heap (state) that satisfies P, also satisfies Q.

Program Validity *wrt*. Pre/Postcondition

{ P } **c** { Q }

If the initial state satisfies P, then, after **c** terminates, the final state satisfies Q.

Transforming Entailment

(our work)

$$P \rightsquigarrow Q$$

There *exists* a program **c**, such that
for any initial state satisfying P,
c, after it terminates,
will transform to a state satisfying Q.

$P \vdash Q$ implies $P \rightsquigarrow Q$

"Proof": **`skip`**

$x \mapsto a \quad \rightsquigarrow \quad x \mapsto 42$

"Proof": `*x = 42`

$x \mapsto a \rightsquigarrow x \mapsto 42 \mid$ `*x = 42`

$P \rightsquigarrow Q \mid \mathbf{c}$

P transforms to Q via a program **c**.

Synthetic Separation Logic

$$\Gamma \; ; \; P \rightsquigarrow Q \; | \; \mathbf{c}$$

$$\Gamma \; ; \; P \rightsquigarrow Q \mid \mathbf{c}$$

- (Γ, P, Q) — *goal*
- **GV** (Γ, P, Q) — *ghost* variables (scope: *pre/postcondition*)
- **EV** (Γ, P, Q) — *existentials* (scope: *postcondition*)

$$\Gamma; \{\mathrm{emp}\} \rightsquigarrow \{\mathrm{emp}\} \mid ??$$

$$\Gamma; \{\mathrm{emp}\} \rightsquigarrow \{\mathrm{emp}\} \mid \mathtt{skip} \qquad (\mathrm{Emp})$$

$$a \in \mathrm{GV}(\Gamma, \mathrm{P}, \mathrm{Q})$$

$$\Gamma; \{\, x \mapsto a \ast \mathrm{P} \,\} \rightsquigarrow \{\, \mathrm{Q} \,\} \mid ??$$

$$\frac{a \in GV(\Gamma, P, Q) \qquad y \text{ is fresh} \qquad \Gamma, y ; [y/a]\{ x \mapsto y * P \} \rightsquigarrow [y/a]\{ Q \} \mid \texttt{c}}{\Gamma; \{ x \mapsto a * P \} \rightsquigarrow \{ Q \} \mid \texttt{let y = *x; c}} \text{(Read)}$$

$$Vars(e) \subseteq \Gamma$$

$$\Gamma; \{ x \mapsto - * P \} \rightsquigarrow \{ x \mapsto e * Q \} \mid ??$$

$$\frac{\begin{array}{c} \mathit{Vars}(e) \subseteq \Gamma \\ \Gamma\,;\{\, x \mapsto e * P \,\} \rightsquigarrow \{\, x \mapsto e * Q \,\} \mid \texttt{c} \end{array}}{\Gamma;\{\, x \mapsto - * P \,\} \rightsquigarrow \{\, x \mapsto e * Q \,\} \mid \texttt{*x = e; c}}\ \text{(Write)}$$

$$\Gamma; \{ P * R \} \rightsquigarrow \{ Q * R \} \mid ??$$

$$\frac{\begin{array}{c}EV(\Gamma, P, Q) \cap \mathit{Vars}(R) = \emptyset \\ \Gamma ; \{ P \} \rightsquigarrow \{ Q \} \mid \mathbf{c}\end{array}}{\Gamma; \{ P * R \} \rightsquigarrow \{ Q * R \} \mid \mathbf{c}} \text{ (Frame)}$$

$$\Gamma;\ \{\mathsf{emp}\} \rightsquigarrow \{\mathsf{emp}\} \mid \mathtt{skip} \qquad \text{(Emp)}$$

$$\frac{a \in GV(\Gamma, P, Q) \qquad y \text{ is fresh} \qquad \Gamma, y;\ [y/a]\{ x \mapsto y * P \} \rightsquigarrow [y/a]\{ Q \} \mid \mathtt{c}}{\Gamma;\ \{ x \mapsto a * P \} \rightsquigarrow \{ Q \} \mid \mathtt{let\ y\ =\ *x;\ c}} \ \text{(Read)}$$

$$\frac{EV(\Gamma, P, Q) \cap \mathit{Vars}(R) = \emptyset \qquad \Gamma;\ \{ P \} \rightsquigarrow \{ Q \} \mid \mathtt{c}}{\Gamma;\ \{ P * R \} \rightsquigarrow \{ Q * R \} \mid \mathtt{c}} \ \text{(Frame)}$$

$$\frac{\mathit{Vars}(e) \subseteq \Gamma \qquad \Gamma;\ \{ x \mapsto e * P \} \rightsquigarrow \{ x \mapsto e * Q \} \mid \mathtt{c}}{\Gamma;\ \{ x \mapsto - * P \} \rightsquigarrow \{ x \mapsto e * Q \} \mid \mathtt{*x\ =\ e;\ c}} \ \text{(Write)}$$

$$\{ x \mapsto a * y \mapsto b \}$$

```
void swap(loc x, loc y)
```

$$\{ x \mapsto b * y \mapsto a \}$$

$$\{ x, y \} \,;\, \{ x \mapsto a * y \mapsto b \} \rightsquigarrow \{ x \mapsto b * y \mapsto a \} \quad | \quad ??$$

$$\dfrac{\{\, x, y, a2 \,\} ;\ \{\, x \mapsto a2 * y \mapsto b \,\} \rightsquigarrow \{\, x \mapsto b * y \mapsto a2 \,\} \mid \texttt{??}}{\{\, x, y \,\} ;\ \{\, x \mapsto a * y \mapsto b \,\} \rightsquigarrow \{\, x \mapsto b * y \mapsto a \,\} \mid \texttt{let a2 = *x; ??}}\ \text{(Read)}$$

$$\dfrac{\dfrac{\{\, x, y, a2, b2 \,\} ;\ \{\, x \mapsto a2 * y \mapsto b2 \,\} \leadsto \{\, x \mapsto b2 * y \mapsto a2 \,\} \mid \texttt{??}}{\{\, x, y, a2 \,\} ;\ \{\, x \mapsto a2 * y \mapsto b \,\} \leadsto \{\, x \mapsto b * y \mapsto a2 \,\} \mid \texttt{let b2 = *y; ??}}\ \text{(Read)}}{\{\, x, y \,\} ;\ \{\, x \mapsto a * y \mapsto b \,\} \leadsto \{\, x \mapsto b * y \mapsto a \,\} \mid \texttt{let a2 = *x; ??}}\ \text{(Read)}$$

$$\{ x, y, a2, b2 \} ; \{ x \mapsto b2 * y \mapsto b2 \} \rightsquigarrow \{ x \mapsto b2 * y \mapsto a2 \} \mid \texttt{??}$$

(Write)

$$\{ x, y, a2, b2 \} ; \{ x \mapsto a2 * y \mapsto b2 \} \rightsquigarrow \{ x \mapsto b2 * y \mapsto a2 \} \mid \texttt{*x = b2; ??}$$

(Read)

$$\{ x, y, a2 \} ; \{ x \mapsto a2 * y \mapsto b \} \rightsquigarrow \{ x \mapsto b * y \mapsto a2 \} \mid \texttt{let b2 = *y; ??}$$

(Read)

$$\{ x, y \} ; \{ x \mapsto a * y \mapsto b \} \rightsquigarrow \{ x \mapsto b * y \mapsto a \} \mid \texttt{let a2 = *x; ??}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\{ x, y, a2, b2 \} ; \{ y \mapsto b2 \} \rightsquigarrow \{ y \mapsto a2 \} \mid \texttt{??}}{\{ x, y, a2, b2 \} ; \{ x \mapsto b2 * y \mapsto b2 \} \rightsquigarrow \{ x \mapsto b2 * y \mapsto a2 \} \mid \texttt{??}}\ \text{(Frame)}}{\{ x, y, a2, b2 \} ; \{ x \mapsto a2 * y \mapsto b2 \} \rightsquigarrow \{ x \mapsto b2 * y \mapsto a2 \} \mid \texttt{*x = b2; ??}}\ \text{(Write)}}{\{ x, y, a2 \} ; \{ x \mapsto a2 * y \mapsto b \} \rightsquigarrow \{ x \mapsto b * y \mapsto a2 \} \mid \texttt{let b2 = *y; ??}}\ \text{(Read)}}{\{ x, y \} ; \{ x \mapsto a * y \mapsto b \} \rightsquigarrow \{ x \mapsto b * y \mapsto a \} \mid \texttt{let a2 = *x; ??}}\ \text{(Read)}$$

$$\{\, x, y, a2, b2 \,\} ; \{\, y \mapsto a2 \,\} \leadsto \{\, y \mapsto a2 \,\} \mid \texttt{??}$$

(Write)

$$\{\, x, y, a2, b2 \,\} ; \{\, y \mapsto b2 \,\} \leadsto \{\, y \mapsto a2 \,\} \mid \texttt{*y = a2; ??}$$

(Frame)

$$\{\, x, y, a2, b2 \,\} ; \{\, x \mapsto b2 * y \mapsto b2 \,\} \leadsto \{\, x \mapsto b2 * y \mapsto a2 \,\} \mid \texttt{??}$$

(Write)

$$\{\, x, y, a2, b2 \,\} ; \{\, x \mapsto a2 * y \mapsto b2 \,\} \leadsto \{\, x \mapsto b2 * y \mapsto a2 \,\} \mid \texttt{*x = b2; ??}$$

(Read)

$$\{\, x, y, a2 \,\} ; \{\, x \mapsto a2 * y \mapsto b \,\} \leadsto \{\, x \mapsto b * y \mapsto a2 \,\} \mid \texttt{let b2 = *y; ??}$$

(Read)

$$\{\, x, y \,\} ; \{\, x \mapsto a * y \mapsto b \,\} \leadsto \{\, x \mapsto b * y \mapsto a \,\} \mid \texttt{let a2 = *x; ??}$$

$$\{\, x, y, a2, b2 \,\} ; \{\, \mathsf{emp} \,\} \leadsto \{\, \mathsf{emp} \,\} \mid \texttt{??}$$

(Frame)

$$\{\, x, y, a2, b2 \,\} ; \{\, y \mapsto a2 \,\} \leadsto \{\, y \mapsto a2 \,\} \mid \texttt{??}$$

(Write)

$$\{\, x, y, a2, b2 \,\} ; \{\, y \mapsto b2 \,\} \leadsto \{\, y \mapsto a2 \,\} \mid \texttt{*y = a2; ??}$$

(Frame)

$$\{\, x, y, a2, b2 \,\} ; \{\, x \mapsto b2 * y \mapsto b2 \,\} \leadsto \{\, x \mapsto b2 * y \mapsto a2 \,\} \mid \texttt{??}$$

(Write)

$$\{\, x, y, a2, b2 \,\} ; \{\, x \mapsto a2 * y \mapsto b2 \,\} \leadsto \{\, x \mapsto b2 * y \mapsto a2 \,\} \mid \texttt{*x = b2; ??}$$

(Read)

$$\{\, x, y, a2 \,\} ; \{\, x \mapsto a2 * y \mapsto b \,\} \leadsto \{\, x \mapsto b * y \mapsto a2 \,\} \mid \texttt{let b2 = *y; ??}$$

(Read)

$$\{\, x, y \,\} ; \{\, x \mapsto a * y \mapsto b \,\} \leadsto \{\, x \mapsto b * y \mapsto a \,\} \mid \texttt{let a2 = *x; ??}$$

$$\overline{\{ x, y, a2, b2 \} ; \{ \mathrm{emp} \} \rightsquigarrow \{ \mathrm{emp} \} \mid \texttt{skip}}\ \text{(Emp)}$$

$$\overline{\{ x, y, a2, b2 \} ; \{ y \mapsto a2 \} \rightsquigarrow \{ y \mapsto a2 \} \mid \texttt{??}}\ \text{(Frame)}$$

$$\overline{\{ x, y, a2, b2 \} ; \{ y \mapsto b2 \} \rightsquigarrow \{ y \mapsto a2 \} \mid \boxed{\texttt{*y = a2;}}\ \texttt{??}}\ \text{(Write)}$$

$$\overline{\{ x, y, a2, b2 \} ; \{ x \mapsto b2 * y \mapsto b2 \} \rightsquigarrow \{ x \mapsto b2 * y \mapsto a2 \} \mid \texttt{??}}\ \text{(Frame)}$$

$$\overline{\{ x, y, a2, b2 \} ; \{ x \mapsto a2 * y \mapsto b2 \} \rightsquigarrow \{ x \mapsto b2 * y \mapsto a2 \} \mid \boxed{\texttt{*x = b2;}}\ \texttt{??}}\ \text{(Write)}$$

$$\overline{\{ x, y, a2 \} ; \{ x \mapsto a2 * y \mapsto b \} \rightsquigarrow \{ x \mapsto b * y \mapsto a2 \} \mid \boxed{\texttt{let b2 = *y;}}\ \texttt{??}}\ \text{(Read)}$$

$$\{ x, y \} ; \{ x \mapsto a * y \mapsto b \} \rightsquigarrow \{ x \mapsto b * y \mapsto a \} \mid \boxed{\texttt{let a2 = *x;}}\ \texttt{??}\ \text{(Read)}$$

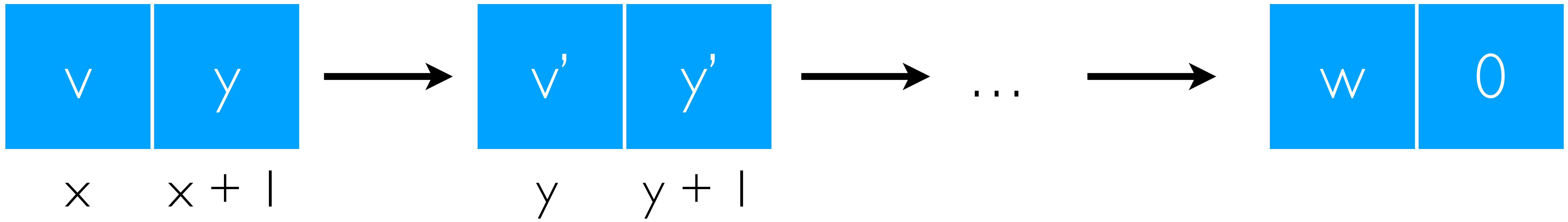
```
void swap(loc x, loc y) {
        let a2 = *x;
        let b2 = *y;
        *x = b2;
        *y = a2;
}
```

Pure Parts

$$\Gamma \, ; \, \{ \, P \, \} \rightsquigarrow \{ \, Q \, \} \quad | \quad \mathbf{c}$$

$$\Gamma \,;\, \{\, \varphi; P \,\} \rightsquigarrow \{\, \psi; Q \,\} \;\mid\; c$$

Inductive Predicates and Recursion

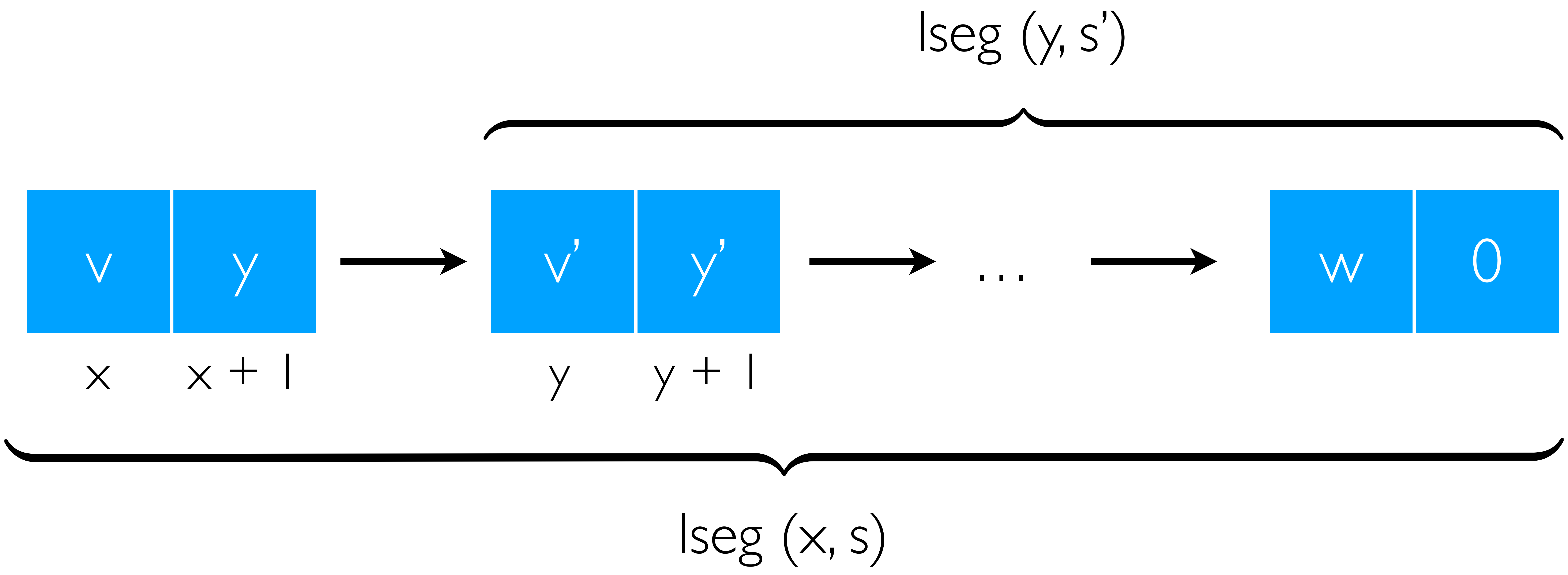

v
y
x
x + 1
v'
y'
y
y + 1
...
w
0

predicate lseg (**loc** x, **set** s) {

  | x = 0 ∧ { s = ∅ ; emp }

  | x ≠ 0 ∧ { s = {v} ∪ s' ; [x, 2] ∗ x ↦ v ∗ (x + 1) ↦ y ∗ lseg(y, s') }

}



```
predicate lseg (loc x, set s) {
  | x = 0  ∧  { s = ∅          ;  emp }
  | x ≠ 0 ∧  { s = {v} ∪ s'   ; [x, 2] ∗ x ↦ v ∗ (x + 1) ↦ y ∗ lseg(y, s') }
}
```

{ lseg (x, s) }

```
void listfree(loc x)
```

{ emp }

predicate lseg (**loc** x, **set** s) {
  | $x = 0 \land \{ s = \varnothing \; ; \; emp \}$
  | $x \neq 0 \land \{ s = \{v\} \cup s' \; ; [x, 2] * x \mapsto v * (x + 1) \mapsto y * lseg(y, s') \}$
}

$\{ lseg^{1}(x, s) \}$ `void listfree(loc x)` $\{ emp \}$

$\{ lseg^{0}(x, s) \}$

??

$\{ emp \}$

predicate lseg (loc x, set s) {
| $x = 0$ ∧ { $s = \varnothing$; emp }
| $x \neq 0$ ∧ { $s = \{v\} \cup s'$; $[x, 2] * x \mapsto v * (x + 1) \mapsto y * lseg(y, s')$ }
}

{ $lseg^{1}(x, s)$ } `void listfree(loc x)` { emp }

{ $lseg^{0}(x, s)$ }

??

{ emp }

predicate lseg (**loc** x, **set** s) {
  | $x = 0 \ \wedge$ { $s = \varnothing$; emp }
  | $x \neq 0 \wedge$ { $s = \{v\} \cup s'$; $[x, 2] * x \mapsto v * (x + 1) \mapsto y *$ lseg(y, s') }
}

{ lseg^{1} (x, s) } `void listfree(loc x)` { emp }

```
if (x == 0) {
```

{ $x = 0$; lseg^{0} (x, s) }

```
    ??
```

{ emp }

```
} else {
```

{ $x \neq 0$; lseg^{0} (x, s) }

```
    ??
```

{ emp }

```
}
```

predicate lseg (**loc** x, **set** s) {
| $x = 0 \land$ { $s = \varnothing$; emp }
| $x \neq 0 \land$ { $s = \{v\} \cup s'$; $[x, 2] * x \mapsto v * (x + 1) \mapsto y *$ lseg(y, s') }
}

{ lseg^1 (x, s) } `void listfree(loc x)` { emp }

```
if (x == 0) {
```

{ $x = 0 \land s = \varnothing$; emp }

```
??
```

{ emp }

```
} else {
```

{ $x \neq 0 \land s = \{v\} \cup s'$; $[x, 2] * x \mapsto v * (x + 1) \mapsto y * \text{lseg}^1 (y, s')$ }

```
??
```

{ emp }

```
}
```

predicate lseg (**loc** x, **set** s) {
  | $x = 0 \land \{ s = \varnothing \;;\; \mathsf{emp} \}$
  | $x \neq 0 \land \{ s = \{v\} \cup s' \;;\; [x, 2] * x \mapsto v * (x + 1) \mapsto y * \mathsf{lseg}(y, s') \}$
}

$\{ \mathsf{lseg}^{\mathbf{1}}(x, s) \}$ `void listfree(loc x)` $\{ \mathsf{emp} \}$

```
if (x == 0) {
```

$\{ x = 0 \land s = \varnothing \;;\; \mathsf{emp} \}$

```
    skip
```

$\{ \mathsf{emp} \}$

```
} else {
```

$\{ x \neq 0 \land s = \{v\} \cup s' \;;\; [x, 2] * x \mapsto v * (x + 1) \mapsto y * \mathsf{lseg}^{\mathbf{1}}(y, s') \}$

```
    ??
```

$\{ \mathsf{emp} \}$

```
}
```

predicate lseg (loc x, set s) {
  | $x = 0 \land$ { $s = \emptyset$; emp }
  | $x \neq 0 \land$ { $s = \{v\} \cup s'$; $[x, 2] * x \mapsto v * (x + 1) \mapsto y * \text{lseg}(y, s')$ }
}

{ $\text{lseg}^1(x, s)$ } `void listfree(loc x)` { emp }

```
if (x == 0) { } else {
```

{ $x \neq 0 \land s = \{v\} \cup s'$; $[x, 2] * x \mapsto v * (x + 1) \mapsto y * \text{lseg}^1(y, s')$ }

```
    ??
```

{ emp }

```
}
```

predicate lseg (loc x, set s) {
| $x = 0 \land \{ s = \varnothing \; ; \; emp \}$
| $x \neq 0 \land \{ s = \{v\} \cup s' \; ; [x, 2] * x \mapsto v * (x + 1) \mapsto y * lseg(y, s') \}$
}

$\{ lseg^{1}(x, s) \}$ `void listfree(loc x)` $\{ emp \}$

```
if (x == 0) { } else {

    let nxt2 = *(x + 1);
```

$\{ x \neq 0 \land s = \{v\} \cup s' \; ; \boxed{[x, 2] * x \mapsto v * (x + 1) \mapsto nxt2} * lseg^{1}(nxt2, s') \}$

```
    ??
```

$\{ emp \}$

```
}
```

predicate lseg (loc x, set s) {
| $x = 0 \wedge$ { $s = \varnothing$; emp }
| $x \neq 0 \wedge$ { $s = \{v\} \cup s'$; $[x, 2] * x \mapsto v * (x + 1) \mapsto y *$ lseg(y, s') }
}

{ $lseg^{1}(x, s)$ } `void listfree(loc x)` { emp }

```
if (x == 0) { } else {
    let nxt2 = *(x + 1);
    free(x);
```

{ $x \neq 0 \wedge s = \{v\} \cup s'$; $lseg^{1}(nxt2, s')$ }

```
    ??
```

{ emp }

```
}
```

predicate lseg (loc x, set s) {
| $x = 0 \land$ { $s = \varnothing$; emp }
| $x \neq 0 \land$ { $s = \{v\} \cup s'$; $[x, 2] * x \mapsto v * (x + 1) \mapsto y *$ lseg(y, s') }
}

{ lseg^1 (x, s) } `void listfree(loc x)` { emp }

```
if (x == 0) { } else {
    let nxt2 = *(x + 1);
    free(x);
    listfree(nxt2);
```

{ $x \neq 0 \land s = \{v\} \cup s'$; emp }

```
    ??
```

{ emp }

```
}
```

predicate lseg (**loc** x, **set** s) {
  | $x = 0 \ \wedge$ { $s = \varnothing$; emp }
  | $x \neq 0 \wedge$ { $s = \{v\} \cup s'$; $[x, 2] * x \mapsto v * (x + 1) \mapsto y *$ lseg(y, s') }
}

{ lseg^{1} (x, s) } `void listfree(loc x)` { emp }

```
if (x == 0) { } else {
    let nxt2 = *(x + 1);
    free(x);
    listfree(nxt2);
    skip;
}
```

```
void listfree(loc x) {
  if (x == 0) { } else {
    let nxt2 = *(x + 1);
    free(x);
    listfree(nxt2);
  }
}
```

All Rules

StarPartial

$$\frac{\begin{array}{c} x + \iota \neq y + \iota' \notin \phi \qquad \phi' \triangleq \phi \wedge (x + \iota \neq y + \iota') \\ \Sigma; \Gamma; \{\phi'; \langle x, \iota \rangle \mapsto e * \langle y, \iota' \rangle \mapsto e' * P\} \leadsto \{Q\} \,|\, c \end{array}}{\Sigma; \Gamma; \{\phi; \langle x, \iota \rangle \mapsto e * \langle y, \iota' \rangle \mapsto e' * P\} \leadsto \{Q\} \,|\, c}$$

Open

$$\frac{\begin{array}{c} \mathcal{D} \triangleq p(\overline{x_i}) \overline{\langle \xi_j, \{\chi_j, R_j\} \rangle}_{j \in 1 \ldots N} \in \Sigma \\ \ell < \mathsf{MaxUnfold} \quad \sigma \triangleq [\overline{x_i \mapsto y_i}] \quad \mathsf{Vars}(\overline{y_i}) \subseteq \Gamma \\ \phi_j \triangleq \phi \wedge [\sigma]\xi_j \wedge [\sigma]\chi_j \qquad P_j \triangleq \lceil [\sigma] R_j \rceil^{\ell+1} * \lceil P \rceil \\ \forall j \in 1 \ldots N, \quad \Sigma; \Gamma; \{\phi_j; P_j\} \leadsto \{Q\} \,|\, c_j \\ c \triangleq \textbf{if } ([\sigma]\xi_1)\, \{c_1\} \textbf{ else } \{\textbf{if } ([\sigma]\xi_2) \ldots \textbf{else } \{c_N\}\} \end{array}}{\Sigma; \Gamma; \{\phi; P * p^{\ell}(\overline{y_i})\} \leadsto \{Q\} \,|\, c}$$

AbduceCall

$$\frac{\begin{array}{c} \mathcal{F} \triangleq f(\overline{x_i}) : \{\phi_f; P_f * F_f\}\{\psi_f; Q_f\} \in \Sigma \\ F_f \text{ has no predicate instances} \qquad [\sigma]P_f = P \\ F_f \neq \mathsf{emp} \qquad F' \triangleq [\sigma]F_f \qquad \Sigma; \Gamma; \{\phi; F\} \leadsto \{\phi; F'\} \,|\, c_1 \\ \Sigma; \Gamma; \{\phi; P * F' * R\} \leadsto \{Q\} \,|\, c_2 \end{array}}{\Sigma; \Gamma; \{\phi; P * F * R\} \leadsto \{Q\} \,|\, c_1; c_2}$$

Read

$$\frac{\begin{array}{c} a \in \mathsf{GV}(\Gamma, \mathcal{P}, \mathcal{Q}) \qquad y \notin \mathsf{Vars}(\Gamma, \mathcal{P}, \mathcal{Q}) \\ \Gamma \cup \{y\}; [y/a]\{\phi; \langle x, \iota \rangle \mapsto a * P\} \leadsto [y/a]\{Q\} \,|\, c \end{array}}{\Sigma; \Gamma; \{\phi; \langle x, \iota \rangle \mapsto a * P\} \leadsto \{Q\} \,|\, \textbf{let } y = *(x + \iota); c}$$

Close

$$\frac{\begin{array}{c} \mathcal{D} \triangleq p(\overline{x_i}) \overline{\langle \xi_j, \{\chi_j, R_j\} \rangle}_{j \in 1 \ldots N} \in \Sigma \\ \ell < \mathsf{MaxUnfold} \qquad \sigma \triangleq [\overline{x_i \mapsto y_i}] \\ \text{for some } k,\ 1 \leq k \leq N \qquad R' \triangleq \lceil [\sigma] R_k \rceil^{\ell+1} \\ \Sigma; \Gamma; \{\mathcal{P}\} \leadsto \{\psi \wedge [\sigma]\xi_k \wedge [\sigma]\chi_k; Q * R'\} \,|\, c \end{array}}{\Sigma; \Gamma; \{\mathcal{P}\} \leadsto \{\psi; Q * p^{\ell}(\overline{y_i})\} \,|\, c}$$

Call

$$\frac{\begin{array}{c} \mathcal{F} \triangleq f(\overline{x_i}) : \{\phi_f; P_f\}\{\psi_f; Q_f\} \in \Sigma \\ R =^{\ell} [\sigma]P_f \qquad \phi \Rightarrow [\sigma]\phi_f \\ \phi' \triangleq [\sigma]\psi_f \qquad R' \triangleq \lceil [\sigma]Q_f \rceil \qquad \overline{e_i} = [\sigma]\overline{x_i} \\ \mathsf{Vars}(\overline{e_i}) \subseteq \Gamma \qquad \Sigma; \Gamma; \{\phi \wedge \phi'; P * R'\} \leadsto \{Q\} \,|\, c \end{array}}{\Sigma; \Gamma; \{\phi; P * R\} \leadsto \{Q\} \,|\, f(\overline{e_i}); c}$$

Alloc

$$\frac{\begin{array}{c} R = [z, n] * \circledast_{0 \leq i \leq n} (\langle z, i \rangle \mapsto e_i) \qquad z \in \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \\ (\{y\} \cup \{\overline{t_i}\}) \cap \mathsf{Vars}(\Gamma, \mathcal{P}, \mathcal{Q}) = \emptyset \\ R' \triangleq [y, n] * \circledast_{0 \leq i \leq n} (\langle y, i \rangle \mapsto t_i) \\ \Sigma; \Gamma; \{\phi; P * R'\} \leadsto \{\psi; Q * R\} \,|\, c \end{array}}{\Sigma; \Gamma; \{\phi; P\} \leadsto \{\psi; Q * R\} \,|\, \textbf{let } y = \mathtt{malloc}(n); c}$$

Free

$$\frac{\begin{array}{c} R = [x, n] * \circledast_{0 \leq i \leq n} (\langle x, i \rangle \mapsto e_i) \\ \mathsf{Vars}(\{x\} \cup \{\overline{e_i}\}) \subseteq \Gamma \qquad \Sigma; \Gamma; \{\phi; P\} \leadsto \{Q\} \,|\, c \end{array}}{\Sigma; \Gamma; \{\phi; P * R\} \leadsto \{Q\} \,|\, \mathtt{free}(n); c}$$

Write

$$\frac{\mathsf{Vars}(e) \subseteq \Gamma \qquad \Gamma; \{\phi; \langle x, \iota \rangle \mapsto e * P\} \leadsto \{\psi; \langle x, \iota \rangle \mapsto e * Q\} \,|\, c}{\Gamma; \{\phi; \langle x, \iota \rangle \mapsto e' * P\} \leadsto \{\psi; \langle x, \iota \rangle \mapsto e * Q\} \,|\, *(x + \iota) = e; c}$$

UnifyHeaps

$$\frac{\begin{array}{c} [\sigma]R' = R \\ \boxed{\mathsf{frameable}}\,(R') \qquad \emptyset \neq \mathsf{dom}(\sigma) \subseteq \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \\ \Gamma; \{P * R\} \leadsto [\sigma]\{\psi; Q * R'\} \,|\, c \end{array}}{\Gamma; \{\phi; P * R\} \leadsto \{\psi; Q * R'\} \,|\, c}$$

Frame

$$\frac{\begin{array}{c} \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \cap \mathsf{Vars}(R) = \emptyset \\ \boxed{\mathsf{frameable}}\,(R') \qquad \Gamma; \{\phi; P\} \leadsto \{\psi; Q\} \,|\, c \end{array}}{\Gamma; \{\phi; P * R\} \leadsto \{\psi; Q * R\} \,|\, c}$$

Induction

$$\frac{\begin{array}{c} f \triangleq \text{goal's name} \\ \overline{x_i} \triangleq \text{goal's formals} \\ P_f \triangleq p^1(\overline{y_i}) * \lceil P \rceil \qquad Q_f \triangleq \lceil Q \rceil \\ \mathcal{F} \triangleq f(\overline{x_i}) : \{\phi_f; P_f\}\{\psi_f; Q_f\} \\ \Sigma, \mathcal{F}; \Gamma; \{\phi; p^0(\overline{y_i}) * P\} \leadsto \{Q\} \,|\, c \end{array}}{\Sigma; \Gamma; \{\phi; p^0(\overline{y_i}) * P\} \leadsto \{Q\} \,|\, c}$$

Emp

$$\frac{\mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) = \emptyset \qquad \phi \Rightarrow \psi}{\Gamma; \{\phi; \mathsf{emp}\} \leadsto \{\psi; \mathsf{emp}\} \,|\, \mathtt{skip}}$$

NullNotLVal

$$\frac{\begin{array}{c} x \neq 0 \notin \phi \qquad \phi' \triangleq \phi \wedge x \neq 0 \\ \Sigma; \Gamma; \{\phi'; \langle x, \iota \rangle \mapsto e * P\} \leadsto \{Q\} \,|\, c \end{array}}{\Sigma; \Gamma; \{\phi; \langle x, \iota \rangle \mapsto e * P\} \leadsto \{Q\} \,|\, c}$$

Inconsistency

$$\frac{\phi \Rightarrow \bot}{\Gamma; \{\phi; P\} \leadsto \{Q\} \,|\, \mathtt{error}}$$

SubstLeft

$$\frac{\begin{array}{c} \phi \Rightarrow x = y \\ \Gamma; [y/x]\{\phi; P\} \leadsto [y/x]\{Q\} \,|\, c \end{array}}{\Gamma; \{\phi; P\} \leadsto \{Q\} \,|\, c}$$

Pick

$$\frac{\begin{array}{c} y \in \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \\ \mathsf{Vars}(e) \in \Gamma \cup \mathsf{GV}(\Gamma, \mathcal{P}, \mathcal{Q}) \\ \Gamma; \{\phi; P\} \leadsto [e/y]\{\psi; Q\} \,|\, c \end{array}}{\Gamma; \{\phi; P\} \leadsto \{\psi; Q\} \,|\, c}$$

UnifyPure

$$\frac{\begin{array}{c} [\sigma]\psi' = \phi' \\ \emptyset \neq \mathsf{dom}(\sigma) \subseteq \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \\ \Gamma; \{\mathcal{P}\} \leadsto [\sigma]\{\mathcal{Q}\} \,|\, c \end{array}}{\Gamma; \{\phi \wedge \phi'; P\} \leadsto \{\psi \wedge \psi'; Q\} \,|\, c}$$

SubstRight

$$\frac{\begin{array}{c} x \in \mathsf{EV}(\Gamma, \mathcal{P}, \mathcal{Q}) \\ \Sigma; \Gamma; \{\mathcal{P}\} \leadsto [e/x]\{\psi, Q\} \,|\, c \end{array}}{\Sigma; \Gamma; \{\mathcal{P}\} \leadsto \{\psi \wedge x = e; Q\} \,|\, c}$$

Theorem 1:

$$P \rightsquigarrow Q \mid \mathbf{c} \quad \text{implies} \quad \{ P \}\ \mathbf{c}\ \{ Q \}$$

Theorem 2:

If $\color{blue}{P \rightsquigarrow Q}$ | **c**

then **c** terminates.

Synthesis Algorithm

Proof Search Algorithm

- Goal-driven, with *backtracking* (in CPS), trying a fixed set of rules;
- *Branching*: some rules emit many alternatives;
- Along with the program, emits the ***complete proof tree.***
- ***Optimisations***: Invertible Rules (*cf. Focusing* in Proof Theory), phased search, “Early Failure” rules

Limitations

- Specifications have to be *inductive*
- Only *structural* recursion *wrt.* inductive predicates (*i.e.*, no QuickSort)
- Unfolding of predicates up to a *fixed depth*
- Limitations of used decision procedures for the *pure* logic fragment

Implementation

SuSLik

(**S**ynthesis **u**sing **S**eparation **L**og**ik**)

- GitHub repository: https://github.com/TyGuS/suslik
- Online Demo: http://comcom.csail.mit.edu/comcom/#SuSLik

Demo?

Group	*Description*	*Code*	*Code/Spec*	*Time*	*T-phase*	*T-inv*	*T-fail*	*T-com*	*T-all*	*T-IS*
Integers	swap two	12	0.9x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	min of two[2]	10	0.7x	0.1	0.1	0.1	< 0.1	0.1	0.2	
Linked List	length[1,2]	21	1.2x	0.4	0.9	0.5	0.4	0.6	1.4	29x
	max[1]	27	1.7x	0.6	0.8	0.5	0.4	0.4	0.8	20x
	min[1]	27	1.7x	0.5	0.9	0.5	0.4	0.5	1.2	49x
	singleton[2]	11	0.8x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	dispose	11	2.8x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	initialize	13	1.4x	< 0.1	0.1	0.1	< 0.1	0.1	< 0.1	
	copy[3]	35	2.5x	0.2	0.3	0.3	0.1	0.2	-	
	append[3]	19	1.1x	0.2	0.3	0.3	0.2	0.3	0.7	
	delete[3]	44	2.6x	0.7	0.5	0.3	0.2	0.3	0.7	
Sorted list	prepend[1]	11	0.3x	0.2	1.4	83.5	0.1	0.1	-	48x
	insert[1]	58	1.2x	4.8	-	-	-	5.0	-	6x
	insertion sort[1]	28	1.3x	1.1	1.8	1.3	1.2	1.2	74.2	82x
Tree	size	38	2.7x	0.2	0.3	0.2	0.2	0.2	0.3	
	dispose	16	4.0x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	copy	55	3.9x	0.4	49.8	-	0.8	1.4	-	
	flatten w/append	48	4.0x	0.4	0.6	0.5	0.4	0.4	0.6	
	flatten w/acc	35	1.9x	0.6	1.7	0.7	0.5	0.6	-	
BST	insert[1]	58	1.2x	31.9	-	-	-	-	-	11x
	rotate left[1]	15	0.1x	37.7	-	-	-	-	-	0.5x
	rotate right[1]	15	0.1x	17.2	-	-	-	-	-	0.8x

[1] From (Qiu and Solar-Lezama 2017) [2] From (Leino and Milicevic 2012) [3] From (Qiu et al. 2013)

Group	*Description*	*Code*	*Code/Spec*	*Time*	*T-phase*	*T-inv*	*T-fail*	*T-com*	*T-all*	*T-IS*
Integers	swap two	12	0.9x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	min of two[2]	10	0.7x	0.1	0.1	0.1	< 0.1	0.1	0.2	
Linked List	length[1,2]	21	1.2x	0.4	0.9	0.5	0.4	0.6	1.4	29x
	max[1]	27	1.7x	0.6	0.8	0.5	0.4	0.4	0.8	20x
	min[1]	27	1.7x	0.5	0.9	0.5	0.4	0.5	1.2	49x
	singleton[2]	11	0.8x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	dispose	11	2.8x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	initialize	13	1.4x	< 0.1	0.1	0.1	< 0.1	0.1	< 0.1	
	copy[3]	35	2.5x	0.2	0.3	0.3	0.1	0.2	-	
	append[3]	19	1.1x	0.2	0.3	0.3	0.2	0.3	0.7	
	delete[3]	44	2.6x	0.7	0.5	0.3	0.2	0.3	0.7	
Sorted list	prepend[1]	11	0.3x	0.2	1.4	83.5	0.1	0.1	-	48x
	insert[1]	58	1.2x	4.8	-	-	-	5.0	-	6x
	insertion sort[1]	28	1.3x	1.1	1.8	1.3	1.2	1.2	74.2	82x
Tree	size	38	2.7x	0.2	0.3	0.2	0.2	0.2	0.3	
	dispose	16	4.0x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	copy	55	3.9x	0.4	49.8	-	0.8	1.4	-	
	flatten w/append	48	4.0x	0.4	0.6	0.5	0.4	0.4	0.6	
	flatten w/acc	35	1.9x	0.6	1.7	0.7	0.5	0.6	-	
BST	insert[1]	58	1.2x	31.9	-	-	-	-	-	11x
	rotate left[1]	15	0.1x	37.7	-	-	-	-	-	0.5x
	rotate right[1]	15	0.1x	17.2	-	-	-	-	-	0.8x

[1] From (Qiu and Solar-Lezama 2017) [2] From (Leino and Milicevic 2012) [3] From (Qiu et al. 2013)

Group	*Description*	*Code*	*Code/Spec*	*Time*	*T-phase*	*T-inv*	*T-fail*	*T-com*	*T-all*	*T-IS*
Integers	swap two	12	0.9x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	min of two[2]	10	0.7x	0.1	0.1	0.1	< 0.1	0.1	0.2	
Linked List	length[1,2]	21	1.2x	0.4	0.9	0.5	0.4	0.6	1.4	29x
	max[1]	27	1.7x	0.6	0.8	0.5	0.4	0.4	0.8	20x
	min[1]	27	1.7x	0.5	0.9	0.5	0.4	0.5	1.2	49x
	singleton[2]	11	0.8x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	dispose	11	2.8x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	initialize	13	1.4x	< 0.1	0.1	0.1	< 0.1	0.1	< 0.1	
	copy[3]	35	2.5x	0.2	0.3	0.3	0.1	0.2	-	
	append[3]	19	1.1x	0.2	0.3	0.3	0.2	0.3	0.7	
	delete[3]	44	2.6x	0.7	0.5	0.3	0.2	0.3	0.7	
Sorted list	prepend[1]	11	0.3x	0.2	1.4	83.5	0.1	0.1	-	48x
	insert[1]	58	1.2x	4.8	-	-	-	5.0	-	6x
	insertion sort[1]	28	1.3x	1.1	1.8	1.3	1.2	1.2	74.2	82x
Tree	size	38	2.7x	0.2	0.3	0.2	0.2	0.2	0.3	
	dispose	16	4.0x	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	< 0.1	
	copy	55	3.9x	0.4	49.8	-	0.8	1.4	-	
	flatten w/append	48	4.0x	0.4	0.6	0.5	0.4	0.4	0.6	
	flatten w/acc	35	1.9x	0.6	1.7	0.7	0.5	0.6	-	
BST	insert[1]	58	1.2x	31.9	-	-	-	-	-	11x
	rotate left[1]	15	0.1x	37.7	-	-	-	-	-	0.5x
	rotate right[1]	15	0.1x	17.2	-	-	-	-	-	0.8x

[1] From (Qiu and Solar-Lezama 2017) [2] From (Leino and Milicevic 2012) [3] From (Qiu et al. 2013)

To Take Away

- **Separation Logic (SL)** is a **Proof System** for heap-manipulating programs.
- **Synthetic Separation Logic (SSL)** expresses **program synthesis** as algorithmic **proof search** for SL-style specifications.
- **SuSLik** is a *deductive synthesis tool* implementing fast proof search in SSL.

Thanks!