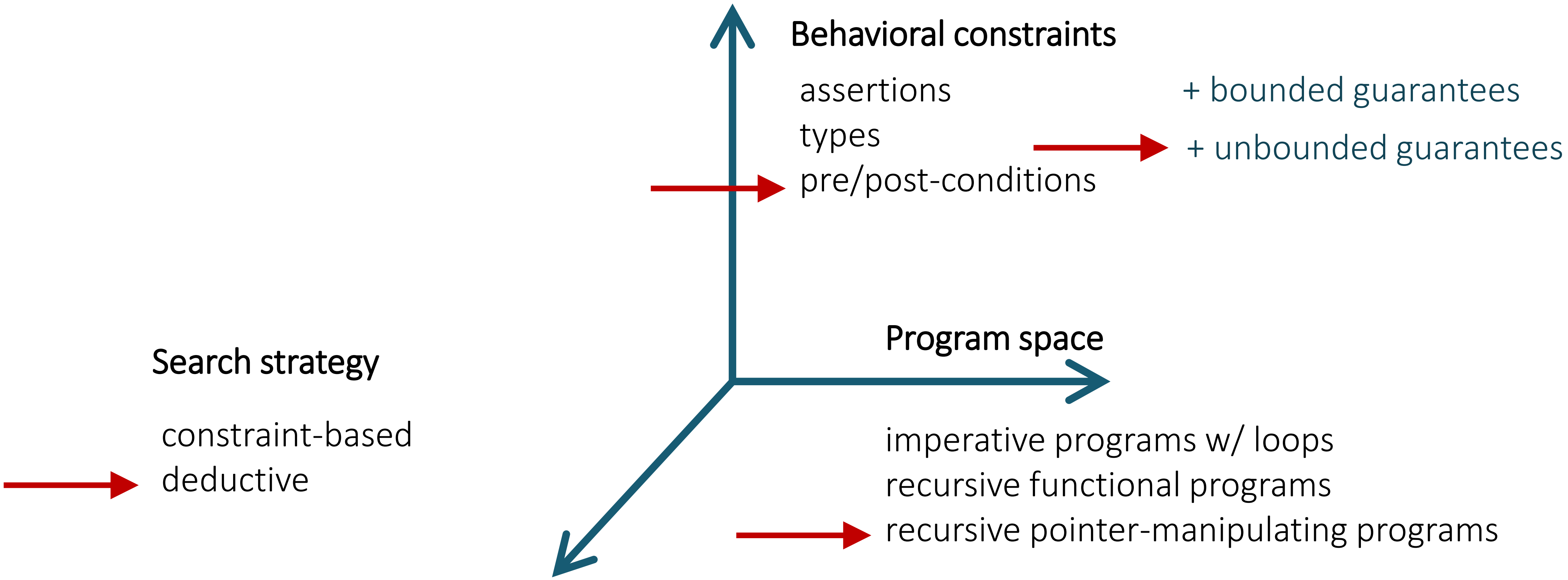
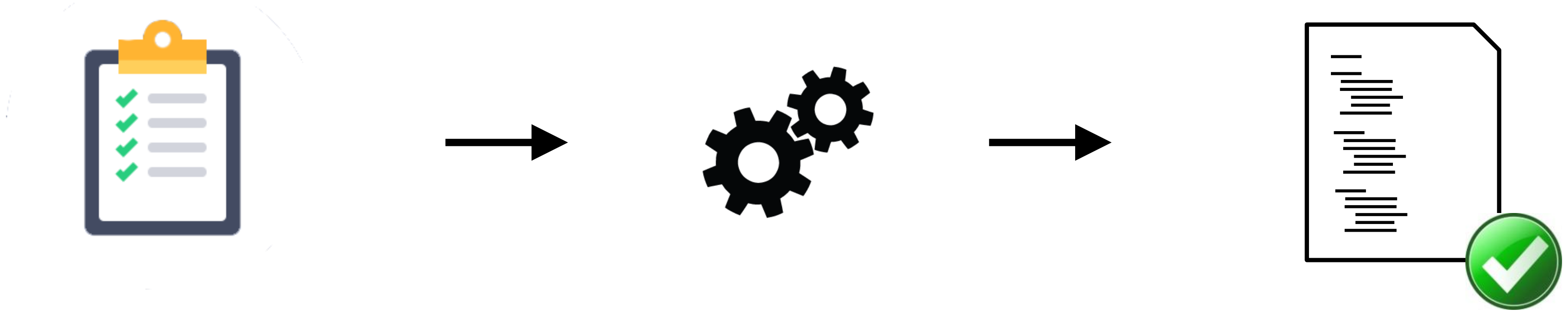
Lecture 13
Separation Logic and Deductive Synthesis

Today

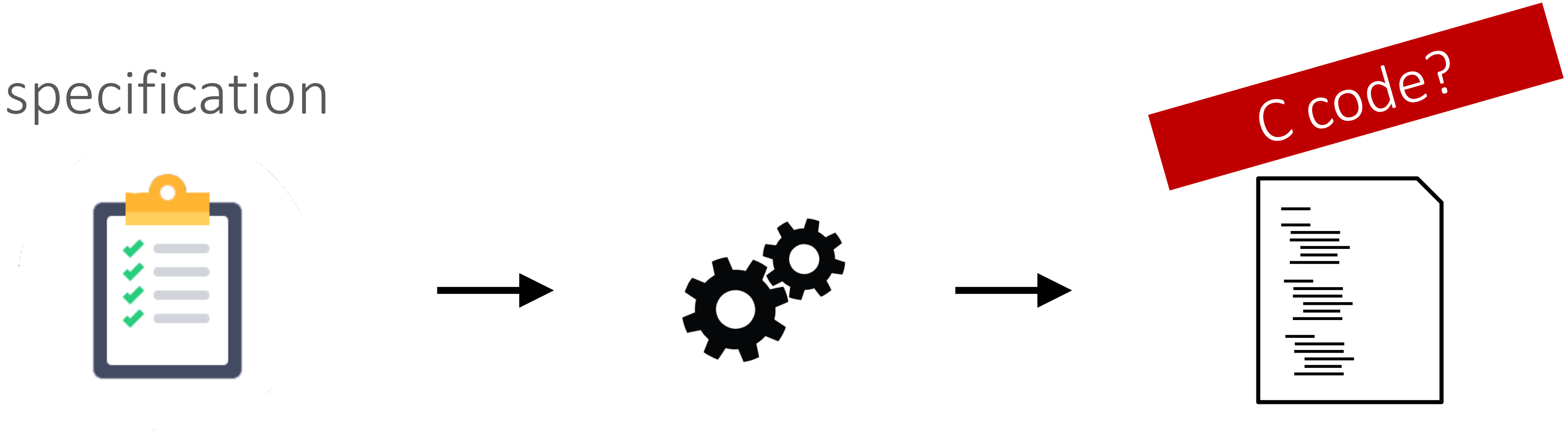

Behavioral constraints
assertions
types
pre/post-conditions
+ bounded guarantees
+ unbounded guarantees
Program space
imperative programs w/ loops
recursive functional programs
recursive pointer-manipulating programs
Search strategy
constraint-based
deductive

Program synthesis

specification code



Program synthesis



☹ verbose
☹ unstructured
☹ pointers

The trouble with pointers

Can we naively apply Hoare logic to programs with pointers?

$\{*x = 10 \wedge *y = 10\}$

$\Rightarrow$

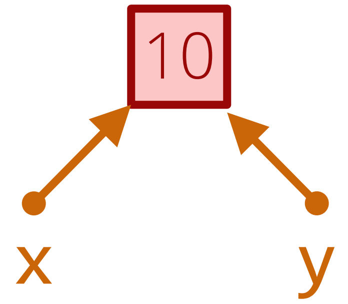
$\{(*x) + 5 = 15 \wedge (*y) - 5 = 5\}$

```
*x = *x + 5;
```

$\{*x = 15 \wedge (*y) - 5 = 5\}$

```
*y = *y - 5;
```

$\{*x = 15 \wedge *y = 5\}$



SuSLik

Synthesis Using Separation Logik

The SuSLik approach

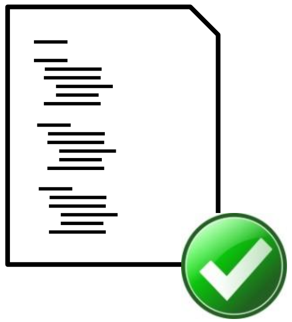
separation logic

☺ reasoning about pointers & aliasing

deductive synthesis

☺ uses specs to guide synthesis

code



Outline

example: swap

a taste of SuSLik

separation logic

specifying pointer-manipulating programs

deductive synthesis

from SL specifications to programs

Outline

example: swap

separation logic

deductive synthesis

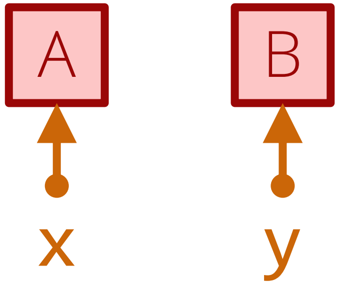
Example: swap

Swap values of two *distinct* pointers

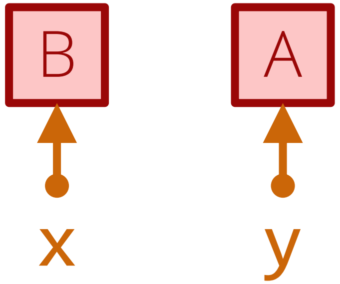
```
void swap(loc x, loc y)
```

Example: swap

start state:



end state:



in separation logic:

$\{ x \mapsto A \boxed{*} y \mapsto B \}$ precondition

separately

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

$\{ x \mapsto B \boxed{*} y \mapsto A \}$ postcondition

logical variables

Demo: swap

Swap values of two *distinct* pointers

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

$$\{\, \boxed{x \mapsto A} * y \mapsto B \,\}$$

??

$$\{\, x \mapsto B * y \mapsto A \,\}$$

```
let a1 = *x;
```

$\{ x \mapsto a1 * \boxed{y \mapsto B} \}$

??

$\{ x \mapsto B * y \mapsto a1 \}$

```
let a1 = *x;
let b1 = *y;
```

$\{ x \mapsto a1 * y \mapsto b1 \}$

??

$\{ \boxed{x \mapsto b1} * y \mapsto a1 \}$

```
let a1 = *x;
let b1 = *y;
*x = b1;
```

$\{ x \mapsto b1 * y \mapsto b1 \}$

??

$\{ x \mapsto b1 * \boxed{y \mapsto a1} \}$

```
let a1 = *x;
let b1 = *y;
*x = b1;
*y = a1;
```

$\{ x \mapsto b1 * y \mapsto a1 \}$

??

$\{ x \mapsto b1 * y \mapsto a1 \}$

same

```
let a1 = *x;
let b1 = *y;
*x = b1;
*y = a1;
```

```
void swap(loc x, loc y) {
        let a1 = *x;
        let b1 = *y;
        *x = b1;
        *y = a1;
}
```

Outline

example: swap

separation logic

deductive synthesis

Separation logic (SL)

Hoare logic

“about the heap”

Separation logic (SL)

SL assertions

{P} c {Q}

program

starting in a state that satisfies P
program **c** will execute without memory errors,
and upon its termination the state will satisfy Q

Outline

example: swap

separation logic

2.1. programs
2.2. assertions
2.3. specifying data transformations

deductive synthesis

Separation logic (SL)

$$\{P\}\ \ c\ \ \{Q\}$$

program

Programs

do nothing	**skip**

Programs

offset (natural number)

do nothing	**skip**
read from heap	**let** y = *(x + n)

variables

Programs

do nothing	**skip**
read from heap	**let** y = *(x + n)
write to heap	*(x + n) = e

expression

(arithmetic, boolean)

Programs

do nothing	**skip**
read from heap	**let** y = *(x + n)
write to heap	*(x + n) = e
allocate block	**let** y = **malloc**(n)

Programs

do nothing	**skip**
read from heap	**let** y = *(x + n)
write to heap	*(x + n) = e
allocate block	**let** y = **malloc**(n)
free block	**free**(x)

Programs

do nothing	**skip**
read from heap	**let** y = *(x + n)
write to heap	*(x + n) = e
allocate block	**let** y = **malloc**(n)
free block	**free**(x)
procedure call	p(e_1, …, e_n)

Programs

do nothing	**skip**
read from heap	**let** y = *(x + n)
write to heap	*(x + n) = e
allocate block	**let** y = **malloc**(n)
free block	**free**(x)
procedure call	p(e_1, …, e_n)
~~assignment~~	only heap is mutable, not stack variables!

Programs

do nothing	**skip**
read from heap	**let** y = *(x + n)
write to heap	*(x + n) = e
allocate block	**let** y = **malloc**(n)
free block	**free**(x)
procedure call	p(e_1, …, e_n)
sequential composition	c_1 ; c_2
conditional	**if** (e) {c_1} else {c_2}

Outline

example: swap

separation logic

2.1. programs
2.2. assertions
2.3. specifying data transformations

deductive synthesis

Separation logic (SL)

SL assertions

{P} c {Q}

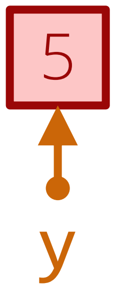
SL assertions

empty heap { emp }

SL assertions

empty heap $\{ \text{emp} \}$

singleton heap $\{ y \mapsto 5 \}$



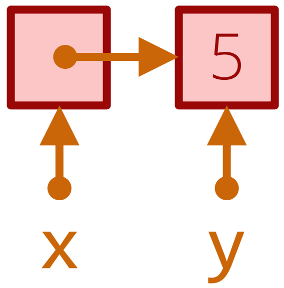
SL assertions

empty heap $\{ \mathrm{emp} \}$

singleton heap $\{ y \mapsto 5 \}$

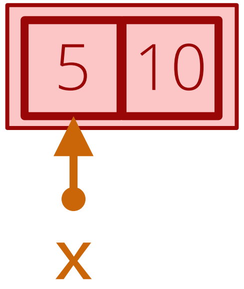
separating conjunction $\{ x \mapsto y * y \mapsto 5 \}$

heaplets



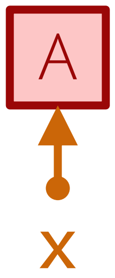
SL assertions

empty heap	{ emp }
singleton heap	{ $y \mapsto 5$ }
separating conjunction	{ $x \mapsto y * y \mapsto 5$ }
memory block	{ $[x, 2] * x \mapsto 5 * (x + 1) \mapsto 10$ }

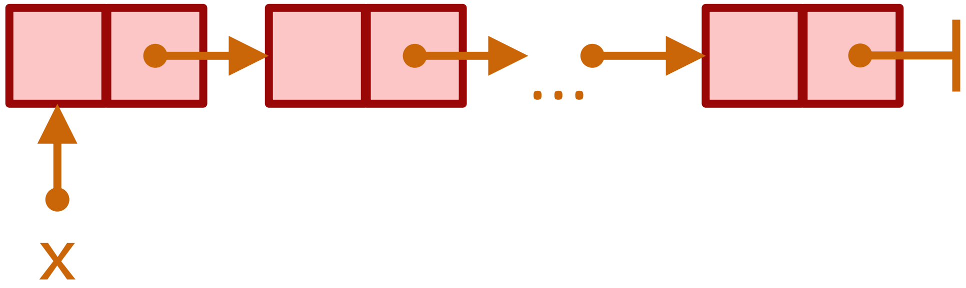


SL assertions

empty heap	$\{ \mathrm{emp} \}$
singleton heap	$\{ y \mapsto 5 \}$
separating conjunction	$\{ x \mapsto y * y \mapsto 5 \}$
memory block	$\{ [x, 2] * x \mapsto 5 * (x + 1) \mapsto 10 \}$
+ pure formula	$\{ A > 5 ; x \mapsto A \}$

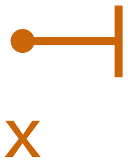


SL assertions: linked structures



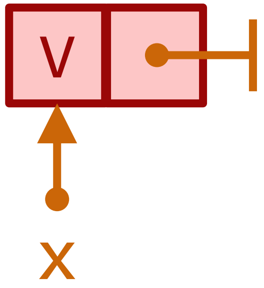
SL assertions: linked structures

linked list { x = 0 ; emp }



SL assertions: linked structures

linked list $\{ [x, 2] * x \mapsto V * (x + 1) \mapsto 0 \}$

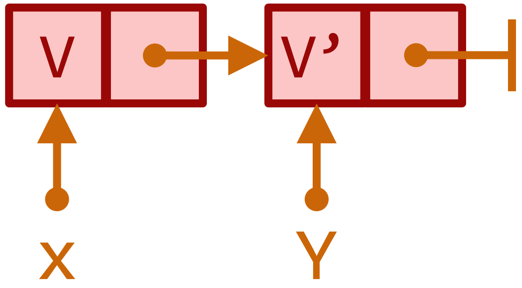


SL assertions: linked structures

linked list $\{\ [x, 2] * x \mapsto V * (x + 1) \mapsto Y *$

$[Y, 2] * Y \mapsto V' * (Y + 1) \mapsto 0$

$\}$



SL assertions: linked structures

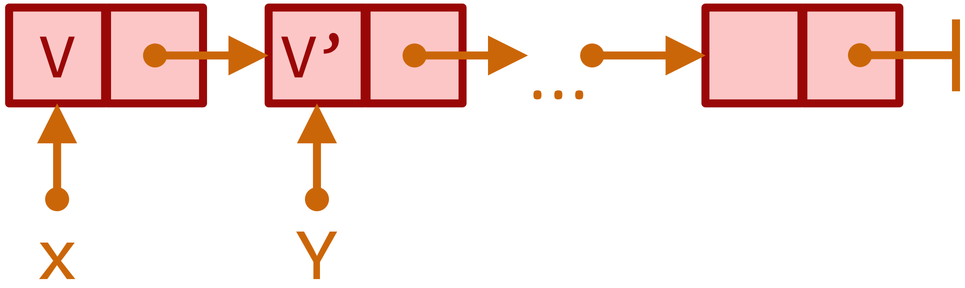
linked list

$\{\ [x, 2] * x \mapsto V * (x + 1) \mapsto Y *$

$[Y, 2] * Y \mapsto V' * (Y + 1) \mapsto Y' *$

…

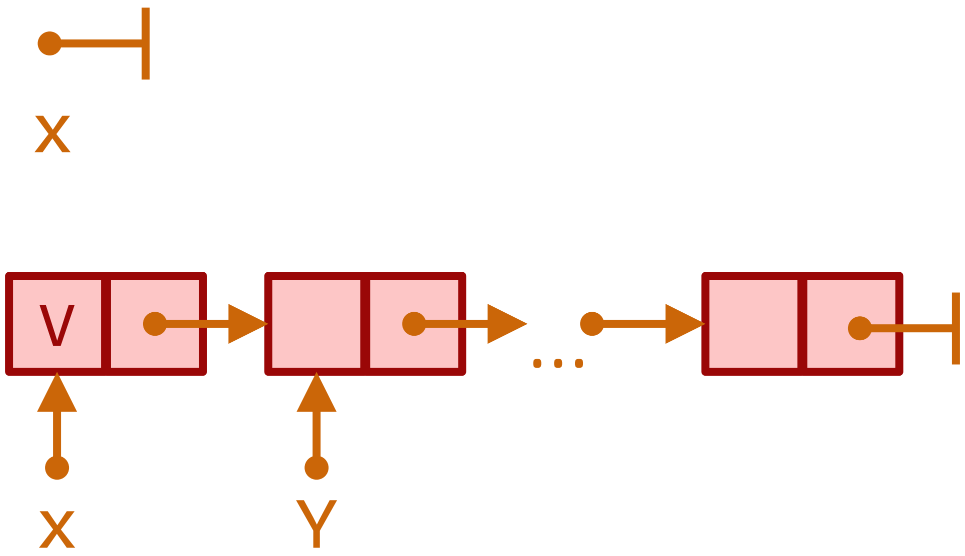
$\}$



inductive predicates to the rescue!

The linked list predicate

```
predicate list (loc x) {
   |  x = 0  =>  { emp }
   |  x ≠ 0  =>  { [x, 2]
                   ∗ x ↦ V
                   ∗ (x + 1) ↦ Y
                   ∗  list(Y)
                 }
}
```



Outline

example: swap

separation logic

2.1. programs

2.2. assertions

2.3. specifying data transformations

deductive synthesis

Demo: dispose a list

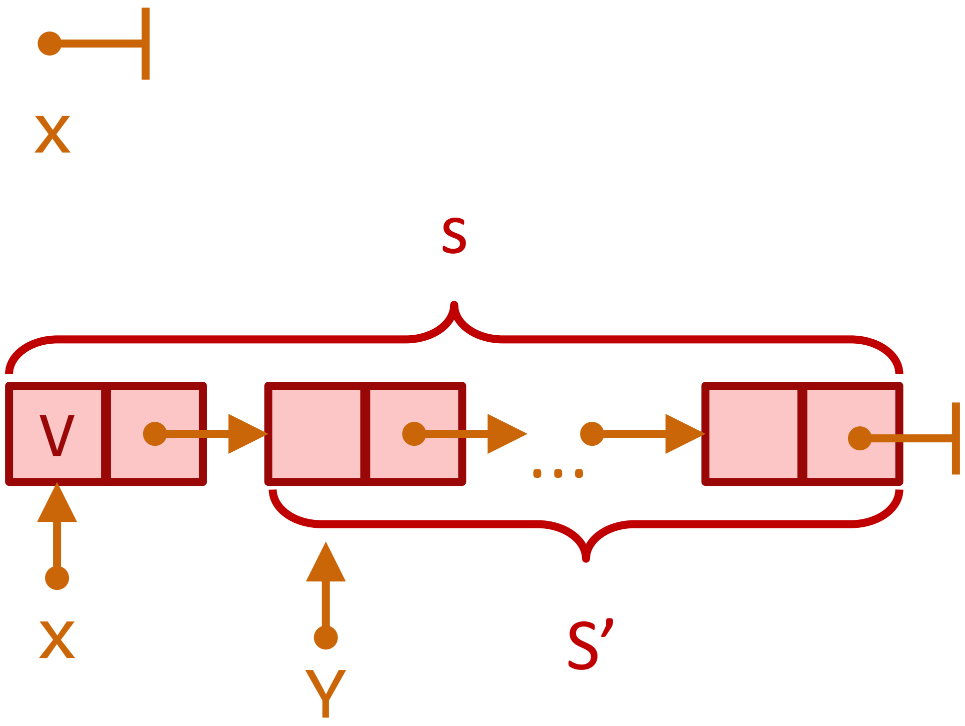
```
void dispose(loc x)
```

{ list(x) }

{ emp }

Linked list with elements

```
predicate list (loc x, set s) {
   |  x = 0  =>  { s = Ø ; emp }
   |  x ≠ 0  =>  { s = {V} + S' ;
                   [x, 2]
                   ∗ x ↦ V ∗ (x + 1) ↦ Y
                   ∗  list(Y, S') }
}
```



Demo: copy a list

```
void copy(loc x, loc ret)
```

{ list(x, S) * ret ↦ _ }

{ list(x, S) * ret ↦ Y * list(Y, S) }

return location

Outline

example: swap

the logic

deductive synthesis

Deductive synthesis

synthesis as proof search

this talk

example: swap

the logic

deductive synthesis

3.1. proof system

3.2. proof search

transforming entailment

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

a state that satisfies P
can be transformed into a state that satisfies Q
using a program **c**

Synthetic separation logic (SSL)

proof system for

transforming entailment

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

(Emp)

$$\{\mathrm{emp}\} \rightsquigarrow \{\mathrm{emp}\} \mid \texttt{skip}$$

(Frame)

$$\frac{\{\, P \,\} \rightsquigarrow \{\, Q \,\} \mid c}{\{\, P * R \,\} \rightsquigarrow \{\, Q * R \,\} \mid ??}$$

(Write)

$$\frac{\{\, x \mapsto e * P \,\} \rightsquigarrow \{\, x \mapsto e * Q \,\} \mid c}{\{\, x \mapsto \_ * P \,\} \rightsquigarrow \{\, x \mapsto e * Q \,\} \mid ??}$$

(Read)

$$\frac{[y/A]\{\, x \mapsto A * P \,\} \rightsquigarrow [y/A]\{\, Q \,\} \mid c}{\{\, x \mapsto A * P \,\} \rightsquigarrow \{\, Q \,\} \mid ??}$$

SSL: basic rules

(Emp)

$$\{emp\} \rightsquigarrow \{emp\} \mid \textbf{skip}$$

(Read)

$$\frac{[y/A]\{\, x \mapsto A * P \,\} \rightsquigarrow [y/A]\{\, Q \,\} \mid c}{\{\, x \mapsto A * P \,\} \rightsquigarrow \{\, Q \,\} \mid \texttt{\textbf{let} y = *x; c}}$$

(Frame)

$$\frac{\{\, P \,\} \rightsquigarrow \{\, Q \,\} \mid c}{\{\, P * R \,\} \rightsquigarrow \{\, Q * R \,\} \mid c}$$

(Write)

$$\frac{\{\, x \mapsto e * P \,\} \rightsquigarrow \{\, x \mapsto e * Q \,\} \mid c}{\{\, x \mapsto \_ * P \,\} \rightsquigarrow \{\, x \mapsto e * Q \,\} \mid \texttt{*x = e; c}}$$

Example: swap

$$\{ x \mapsto A * y \mapsto B \} \rightsquigarrow \{ x \mapsto B * y \mapsto A \} \mid \quad ??$$

$$\{ x \mapsto A * y \mapsto B \} \rightsquigarrow \{ x \mapsto B * y \mapsto A \} \quad | \quad ??$$

$$\dfrac{\{\, x \mapsto a1 * y \mapsto B \,\} \rightsquigarrow \{\, x \mapsto B * y \mapsto a1 \,\} \mid \texttt{??}}{\{\, x \mapsto A * y \mapsto B \,\} \rightsquigarrow \{\, x \mapsto B * y \mapsto A \,\} \mid \texttt{let a1 = *x; ??}} \text{(Read)}$$

$$\cfrac{\cfrac{\{x \mapsto a1 * y \mapsto b1\} \rightsquigarrow \{x \mapsto b1 * y \mapsto a1\} \mid \texttt{??}}{\{x \mapsto a1 * y \mapsto B\} \rightsquigarrow \{x \mapsto B * y \mapsto a1\} \mid \texttt{let b1 = *y; ??}}\ (\text{Read})}{\{x \mapsto A * y \mapsto B\} \rightsquigarrow \{x \mapsto B * y \mapsto A\} \mid \texttt{let a1 = *x; ??}}\ (\text{Read})$$

$$
\dfrac{\dfrac{\dfrac{\{\, x \mapsto b1 * y \mapsto b1 \,\} \rightsquigarrow \{\, x \mapsto b1 * y \mapsto a1 \,\} \mid \texttt{??}}{\{\, x \mapsto a1 * y \mapsto b1 \,\} \rightsquigarrow \{\, x \mapsto b1 * y \mapsto a1 \,\} \mid \texttt{*x = b1; ??}}\;(\text{Write})}{\{\, x \mapsto a1 * y \mapsto B \,\} \rightsquigarrow \{\, x \mapsto B * y \mapsto a1 \,\} \mid \texttt{let b1 = *y; ??}}\;(\text{Read})}{\{\, x \mapsto A * y \mapsto B \,\} \rightsquigarrow \{\, x \mapsto B * y \mapsto A \,\} \mid \texttt{let a1 = *x; ??}}\;(\text{Read})
$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\{\, y \mapsto b1 \,\} \rightsquigarrow \{\, y \mapsto a1 \,\} \mid \texttt{??}}{\{\, x \mapsto b1 * y \mapsto b1 \,\} \rightsquigarrow \{\, x \mapsto b1 * y \mapsto a1 \,\} \mid \texttt{??}}\ (\text{Frame})}{\{\, x \mapsto a1 * y \mapsto b1 \,\} \rightsquigarrow \{\, x \mapsto b1 * y \mapsto a1 \,\} \mid \texttt{*x = b1; ??}}\ (\text{Write})}{\{\, x \mapsto a1 * y \mapsto B \,\} \rightsquigarrow \{\, x \mapsto B * y \mapsto a1 \,\} \mid \texttt{let b1 = *y; ??}}\ (\text{Read})}{\{\, x \mapsto A * y \mapsto B \,\} \rightsquigarrow \{\, x \mapsto B * y \mapsto A \,\} \mid \texttt{let a1 = *x; ??}}\ (\text{Read})$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{ y \mapsto a1 \} \leadsto \{ y \mapsto a1 \} \mid \texttt{??}}{\{ y \mapsto b1 \} \leadsto \{ y \mapsto a1 \} \mid \texttt{*y = a1; ??}}\ (\text{Write})}{\{ x \mapsto b1 * y \mapsto b1 \} \leadsto \{ x \mapsto b1 * y \mapsto a1 \} \mid \texttt{??}}\ (\text{Frame})}{\{ x \mapsto a1 * y \mapsto b1 \} \leadsto \{ x \mapsto b1 * y \mapsto a1 \} \mid \texttt{*x = b1; ??}}\ (\text{Write})}{\{ x \mapsto a1 * y \mapsto B \} \leadsto \{ x \mapsto B * y \mapsto a1 \} \mid \texttt{let b1 = *y; ??}}\ (\text{Read})}{\{ x \mapsto A * y \mapsto B \} \leadsto \{ x \mapsto B * y \mapsto A \} \mid \texttt{let a1 = *x; ??}}\ (\text{Read})
$$

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\{\,\mathrm{emp}\,\} \leadsto \{\,\mathrm{emp}\,\} \mid \texttt{??}}{\{\,y \mapsto a1\,\} \leadsto \{\,y \mapsto a1\,\} \mid \texttt{??}}\,(\text{Frame})}{\{\,y \mapsto b1\,\} \leadsto \{\,y \mapsto a1\,\} \mid \texttt{*y = a1; ??}}\,(\text{Write})}{\{\,x \mapsto b1 \ast y \mapsto b1\,\} \leadsto \{\,x \mapsto b1 \ast y \mapsto a1\,\} \mid \texttt{??}}\,(\text{Frame})}{\{\,x \mapsto a1 \ast y \mapsto b1\,\} \leadsto \{\,x \mapsto b1 \ast y \mapsto a1\,\} \mid \texttt{*x = b1; ??}}\,(\text{Write})}{\{\,x \mapsto a1 \ast y \mapsto B\,\} \leadsto \{\,x \mapsto B \ast y \mapsto a1\,\} \mid \texttt{let b1 = *y; ??}}\,(\text{Read})}{\{\,x \mapsto A \ast y \mapsto B\,\} \leadsto \{\,x \mapsto B \ast y \mapsto A\,\} \mid \texttt{let a1 = *x; ??}}\,(\text{Read})
$$

$$\frac{}{\{\,\mathsf{emp}\,\} \rightsquigarrow \{\,\mathsf{emp}\,\} \mid \boxed{\texttt{skip}}}\;(\text{Emp})$$

$$\frac{\qquad}{\{\,y \mapsto a1\,\} \rightsquigarrow \{\,y \mapsto a1\,\} \mid \texttt{??}}\;(\text{Frame})$$

$$\frac{\qquad}{\{\,y \mapsto b1\,\} \rightsquigarrow \{\,y \mapsto a1\,\} \mid \boxed{\texttt{*y = a1;}}\,\texttt{??}}\;(\text{Write})$$

$$\frac{\qquad}{\{\,x \mapsto b1 * y \mapsto b1\,\} \rightsquigarrow \{\,x \mapsto b1 * y \mapsto a1\,\} \mid \texttt{??}}\;(\text{Frame})$$

$$\frac{\qquad}{\{\,x \mapsto a1 * y \mapsto b1\,\} \rightsquigarrow \{\,x \mapsto b1 * y \mapsto a1\,\} \mid \boxed{\texttt{*x = b1;}}\,\texttt{??}}\;(\text{Write})$$

$$\frac{\qquad}{\{\,x \mapsto a1 * y \mapsto B\,\} \rightsquigarrow \{\,x \mapsto B * y \mapsto a1\,\} \mid \boxed{\texttt{let b1 = *y;}}\,\texttt{??}}\;(\text{Read})$$

$$\frac{\qquad}{\{\,x \mapsto A * y \mapsto B\,\} \rightsquigarrow \{\,x \mapsto B * y \mapsto A\,\} \mid \boxed{\texttt{let a1 = *x;}}\,\texttt{??}}\;(\text{Read})$$

$\{ x \mapsto A * y \mapsto B \}$

```
let a1 = *x;  let b1 = *y;  *x = b1;  *y = a1;  skip
```

$\{ x \mapsto B * y \mapsto A \}$

Synthetic separation logic (SSL)

basic rules

(Emp), (Read), (Write), (Frame)

(Alloc), (Free)

pure reasoning and unification

inductive predicates and recursion

Synthetic separation logic (SSL)

basic rules

(Emp), (Read), (Write), (Frame)

(Alloc), (Free)

pure reasoning and unification

inductive predicates and recursion

Example: dispose a list

```
void dispose(loc x)
```

{ list(x) }

{ emp }

$\{\boxed{\text{list}^1(x)}\}$ **void** dispose(**loc** x) $\{ \text{emp} \}$

$\{\boxed{\text{list}^0(x)}\}$

?? (Induction)

$\{ \text{emp} \}$

predicate list (loc x) {
  | $\boxed{x = 0}$ => { emp }
  | $\boxed{x \neq 0}$ => { [x, 2] $*$ x $\mapsto$ V $*$ (x + 1) $\mapsto$ Y $*$ list(Y) }
}

{ $list^1$ (x) } **void** dispose(**loc** x) { emp }

{ $list^0(x)$ }

??

(Open)

{ emp }

predicate list (loc x) {
| x = 0 => $\boxed{\{ \text{emp} \}}$
| x ≠ 0 => $\boxed{\{ [x, 2] * x \mapsto V * (x + 1) \mapsto Y * \text{list}(Y) \}}$
}

$\{ \text{list}^1 (x) \}$ **void** dispose(**loc** x) { emp }

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

$\{ x = 0 \,;\ \text{emp} \}$

```
    ??
```

$\{ \text{emp} \}$

```
} else {
```

$\{ x \neq 0 \,;\ [x, 2] * x \mapsto V * (x + 1) \mapsto Y * \text{list}^1 (Y) \}$

```
    ??
```

$\{ \text{emp} \}$

```
}
```

$\{\ list^1\ (x)\ \}$ **void** dispose(**loc** x) $\{\ emp\ \}$

predicate list (loc x) {
  | $x = 0$ => $\{\ emp\ \}$
  | $x \neq 0$ => $\{\ [x, 2] * x \mapsto V * (x + 1) \mapsto Y * \ list(Y)\ \}$
}

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

$\{\ x = 0\ ;\ emp\ \}$

?? (Emp)

$\{\ emp\ \}$

```
} else {
```

$\{\ x \neq 0\ ;\ [x, 2] * x \mapsto V * (x + 1) \mapsto Y * list^1\ (Y)\ \}$

??

$\{\ emp\ \}$

}

predicate list (loc x) {
| $x = 0$ => { emp }
| $x \neq 0$ => { $[x, 2] * x \mapsto V * (x+1) \mapsto Y *$ list(Y) }
}

{ list^1 (x) } **void** dispose(**loc** x) { emp }

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

{ $x = 0$; emp }

```
    skip
```

{ emp }

```
} else {
```

{ $x \neq 0$; $[x, 2] * x \mapsto V * (x+1) \mapsto Y * \text{list}^1(Y)$ }

```
    ??
```

{ emp }

```
}
```

predicate list (loc x) {
  | $x = 0$ => { emp }
  | $x \neq 0$ => { $[x, 2] * x \mapsto V * (x + 1) \mapsto Y *$ list(Y) }
}

{ list^1 (x) } **void** dispose(**loc** x) { emp }

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

{ $x \neq 0$; $[x, 2] * x \mapsto V * (x + 1) \mapsto Y *$ list^1 (Y) }

```
    ??
```

{ emp }

}

predicate list (loc x) {
| $x = 0$ => { emp }
| $x \neq 0$ => { $[x, 2] * x \mapsto V * (x+1) \mapsto Y *$ list(Y) }
}

{ list¹ (x) } **void** dispose(**loc** x) { emp }

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

{ $x \neq 0$; $[x, 2] * x \mapsto V * (x+1) \mapsto Y *$ list¹ (Y) }

(Read)

```
    ??
```

{ emp }

}

predicate list (loc x) {
  | $x = 0$ => { emp }
  | $x \neq 0$ => { $[x, 2] * x \mapsto V * (x + 1) \mapsto Y *$ list(Y) }
}

{ list^1 (x) } **void** dispose(**loc** x) { emp }

```
if (x == 0) { skip } else {
    let y1 = *(x + 1);
```

{ $x \neq 0$; $[x, 2] * x \mapsto V * (x + 1) \mapsto y1 *$ list^1 (y1) }

```
    ??
```

{ emp }

```
}
```

predicate list (loc x) {
  | $x = 0$ => { emp }
  | $x \neq 0$ => { $[x, 2] * x \mapsto V * (x + 1) \mapsto Y *$ list(Y) }
}

{ $list^1(x)$ } **void** dispose(**loc** x) { emp }

```
if (x == 0) { skip } else {
    let y1 = *(x + 1);
```

{ $x \neq 0$; $[x, 2] * x \mapsto V * (x + 1) \mapsto y1 * list^1(y1)$ }

```
    ??
```

(Free)

{ emp }

```
}
```

predicate list (loc x) {
| $x = 0$ => { emp }
| $x \neq 0$ => { $[x, 2] * x \mapsto V * (x + 1) \mapsto Y *$ list(Y) }
}

{ list^1 (x) } **void** dispose(**loc** x) { emp }

```
if (x == 0) { skip } else {
    let y1 = *(x + 1);
    free x;
```

{ $x \neq 0$; list^1 (y1) }

```
    ??
```

{ emp }

```
}
```

predicate list (loc x) {
  | $x = 0$ => { emp }
  | $x \neq 0$ => { [x, 2] $*$ $x \mapsto V * (x + 1) \mapsto Y *$ list(Y) }
}

{ list[1] (x) } **void** dispose(**loc** x) { emp }

```
if (x == 0) { skip } else {
    let y1 = *(x + 1);
    free x;
```

{ $x \neq 0$; list[1] (y1) }

(Call)

```
    ??
```

{ emp }

}

predicate list (loc x) {
  | $x = 0$ => { emp }
  | $x \neq 0$ => { $[x, 2] * x \mapsto V * (x + 1) \mapsto Y *$ list(Y) }
}

{ list[1] (x) } **void** dispose(**loc** x) { emp }

```
if (x == 0) { skip } else {
    let y1 = *(x + 1);
    free x;
    dispose(y1);
```

{ $x \neq 0$; emp }

??

{ emp }

```
}
```

predicate list (loc x) {
  | $x = 0$ => { emp }
  | $x \neq 0$ => { $[x, 2] * x \mapsto V * (x + 1) \mapsto Y *$ list(Y) }
}

{ list^1 (x) } **void** dispose(**loc** x) { emp }

```
if (x == 0) { skip } else {
    let y1 = *(x + 1);
    free x;
    dispose(y1);
```

{ $x \neq 0$; emp }

??

{ emp }

```
}
```

(Emp)

predicate list (loc x) {
  | $x = 0$ => { emp }
  | $x \neq 0$ => { $[x, 2] * x \mapsto V * (x + 1) \mapsto Y *$ list(Y) }
}

{ list^1 (x) } **void** dispose(**loc** x) { emp }

```
if (x == 0) { skip } else {
    let y1 = *(x + 1);
    free x;
    dispose(y1);
    skip



}
```

```
void dispose(loc x) {
  if (x == 0) { }
  else {
    let y1 = *(x + 1);
    free x;
    dispose(y1)
  }
}
```

Synthetic separation logic (SSL)

basic rules

(Emp), (Read), (Write), (Frame), (Alloc), (Free)

pure reasoning and unification

inductive predicates and recursion

(Open), (Close), (Induction), (Call)

Outline

example: swap

the logic

deductive synthesis

3.1. proof system

3.2. proof search

SuSLik

best-first top-down search in SSL

+ optimizations

Optimizations

invertible rules

early failure

multi-phase search

goal memoization

Optimization: invertible rules

invertible rules do not restrict the set of derivable programs

idea: invertible rules need not be backtracked

(Read)

$$\frac{[y/A]\{\, x \mapsto A * P \,\} \rightsquigarrow [y/A]\{\, Q \,\} \mid \texttt{c}}{\{\, x \mapsto A * P \,\} \rightsquigarrow \{\, Q \,\} \mid \texttt{let y = *x; c}}$$

Optimization: early failure

idea: sometimes you know that a goal is unsatisfiable

(Post-Inconsistent)

$$\frac{\psi \neq \bot \qquad \vdash \phi \land \psi \Rightarrow \bot \qquad \{\, \mathrm{emp} \,\} \rightsquigarrow \{\, \bot \,;\, \mathrm{emp} \,\} \mid c}{\{\, \phi \,;\, P \,\} \rightsquigarrow \{\, \psi \,; Q \,\} \mid c}$$

Optimization: multi-phase search

unfolding phase: eliminates inductive predicates

(Open), (Close), (Call), (Unify) , (Frame)

block phase: eliminates blocks

(Alloc), (Free), (Unify), (Frame)

pointer phase: aligns all pointers

(Unify), (Frame)

pure phase: deals with the content of pointers

(Write), (Unify), (Pure)

idea: if a phase fails, don’t start the next one

SuSLik: contributions

Synthesis with unbounded guarantees for recursive heap-manipulating programs

Does not require sketches

Optimizations from proof search

SuSLik: limitations

Only structurally recursive programs
No auxiliary functions

- These two has been fixed-ish in follow-up work [PLDI'21]

Requires the spec to be inductive

Does not support loops

- But we can turn recursion into loops (?)

Requires SL expertise

Incompleteness due to bounded unfolding

No notion of cost/efficiency

SuSLik: questions

Behavioral constraints? Structural constraints? Search strategy?

- SL specs
- Set of components + built-in language constraints
- Deductive search

SuSLik: questions

Why is Frame rule unsound in the presence of assignments?

$$\text{(Frame)}\quad \frac{\{\, P \,\} \rightsquigarrow \{\, Q \,\} \mid c}{\{\, P * R \,\} \rightsquigarrow \{\, Q * R \,\} \mid c}\quad fv(R) \cap mod(c) = \emptyset$$

$\{\, x = 0;\ y \mapsto x \,\}$

$x := 1$

$\{\, x = 1;\ y \mapsto x \,\}$

provable with this frame rule, but incorrect!

which side condition should we add to Frame?

SuSLik: questions

Why is Frame rule not invertible?

- What other rule can it prevent from applying?

$$\text{(Frame)}\ \frac{\{\, P \,\} \rightsquigarrow \{\, Q \,\} \mid c}{\{\, P * R \,\} \rightsquigarrow \{\, Q * R \,\} \mid c}$$