SuSLik: synthesis of safe pointer-manipulating programs

Nadia Polikarpova

joint work with Ilya Sergey (Yale-NUS)

UCSDCSE
Computer Science and Engineering

follow along

https://github.com/TyGuS/suslik-tutorial

pointer-manipulating programs

operating systems

network / security protocols

browsers

☺ efficient

☹ hard to write

☹ memory safety bugs

how to make them safe?

write in a high-level language

write in C, verify in Coq

☺ easy to write

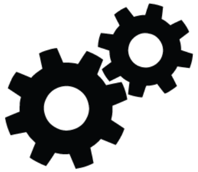
☹ have to rewrite everything

☹ hard to write

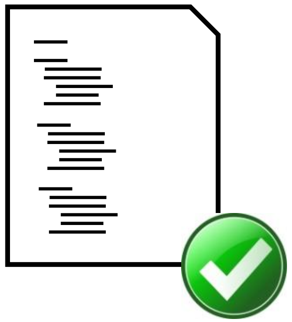
☺ backwards compatible

program synthesis to the rescue

specification



code



☺ easy to write

☹ verbose
☹ unstructured
☹ pointers & aliasing

☺ efficient
☺ backwards compatible
☺ provably memory-safe

SuSLik

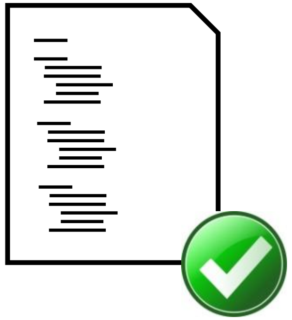
(**S**ynthesis **u**sing **S**eparation **L**ogi**k**)

the SuSLik approach

separation logic

deductive synthesis

code



☺ reasoning about pointers & aliasing

☺ uses specs to guide synthesis

this tutorial

1. example: swap
   a taste of SuSLik

2. intro to separation logic
   reasoning about pointer-manipulating programs

3. deductive synthesis
   from SL specifications to programs

this tutorial

1. example: swap
2. intro to separation logic
3. deductive synthesis

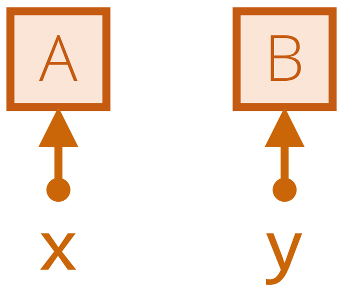
example: swap

Swap values of two *distinct* pointers

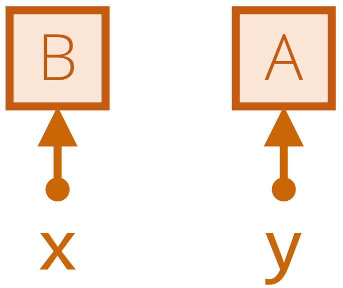
```
void swap(loc x, loc y)
```

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

ghost variables

demo 1: 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;
```

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

??

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

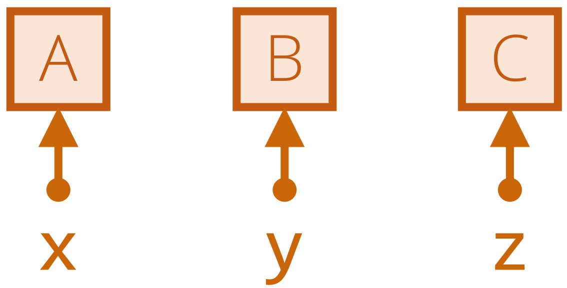
same

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

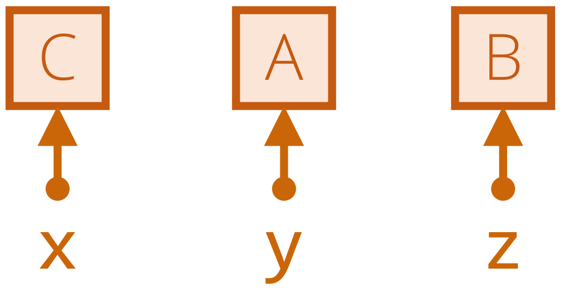
```
void swap(loc x, loc y) {
        let a1 = *x;
        let b1 = *y;
        *x = b1;
        *y = a1;
}
```

exercise 1: rotate three

start state:



end state:




```
void rotate(loc x, loc y, loc z)
```

this tutorial

1. example: swap
2. intro to separation logic
3. 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

this tutorial

1. example: swap
2. intro to separation logic
   2.1. programs and assertions
   2.2. inductive predicates
   2.3. specifying data transformations
3. 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	$p(e_1, \ldots, e_n)$
sequential composition	c_1 ; c_2
conditional	**if** (e) {c_1} else {c_2}

separation logic (SL)

SL assertions

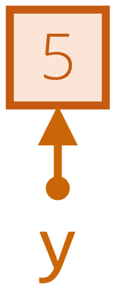
{P} c {Q}

SL assertions

empty heap { emp }

SL assertions

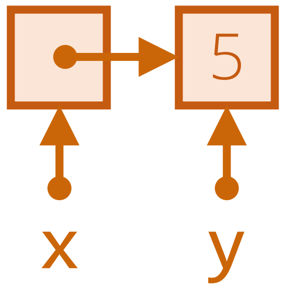
empty heap	{ emp }
singleton heap	{ $y \mapsto 5$ }



SL assertions

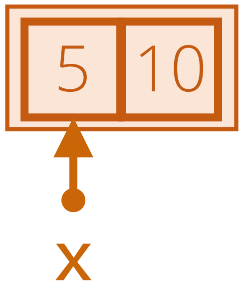
empty heap	{ emp }
singleton heap	{ $y \mapsto 5$ }
separating conjunction	{ $x \mapsto y$ * $y \mapsto 5$ }

heaplets



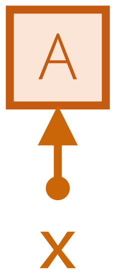
SL assertions

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



SL assertions

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



separation logic (SL)

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

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

example triples

$\{ x \mapsto A \}$	`*x = 5`	$\{ x \mapsto 5 \}$
$\{ x \mapsto A \}$	`*(x + 1) = 5`	❌
$\{ x \mapsto A \}$	`let y = *x`	$\{ x \mapsto y \}$
$\{ \mathrm{emp} \}$	`let y = malloc(2)`	$\{ [y, 2] * y \mapsto A * (y + 1) \mapsto B \}$
$\{ [x, 2] * x \mapsto 5 * (x + 1) \mapsto 7 \}$	`free(x + 1)`	❌

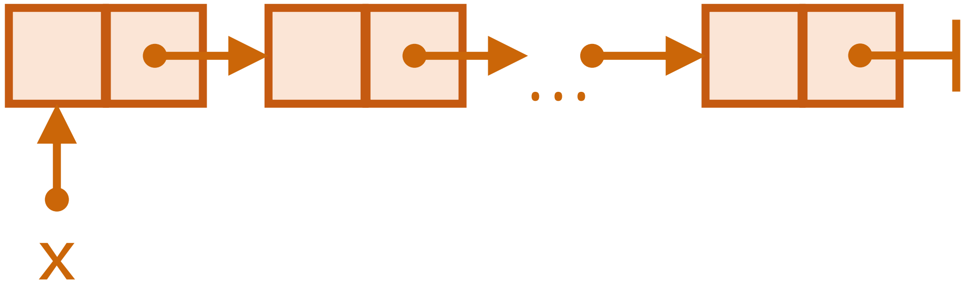
this tutorial

1. example: swap

2. intro to separation logic
   2.1. programs and assertions
   2.2. inductive predicates
   2.3. specifying data transformations

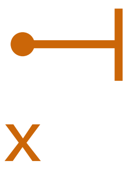
3. deductive synthesis

SL assertions: linked structures



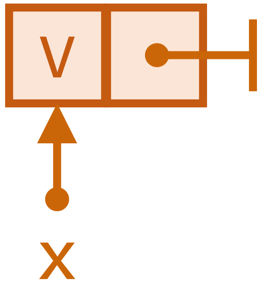
SL assertions: linked structures

linked list { x = 0 ; emp }



SL assertions: linked structures

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

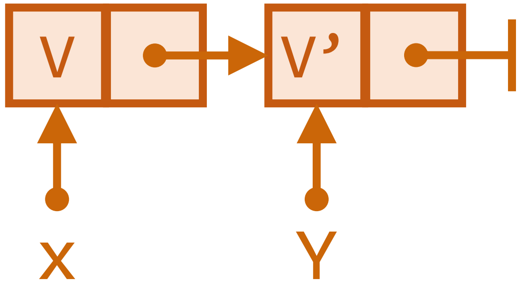


SL assertions: linked structures

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

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

$\}$



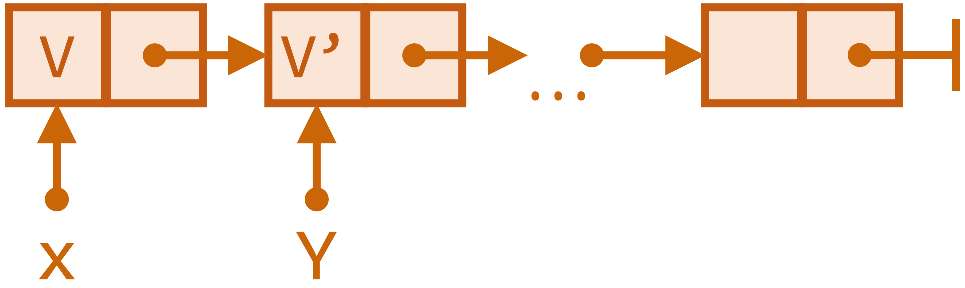
SL assertions: linked structures

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

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

…

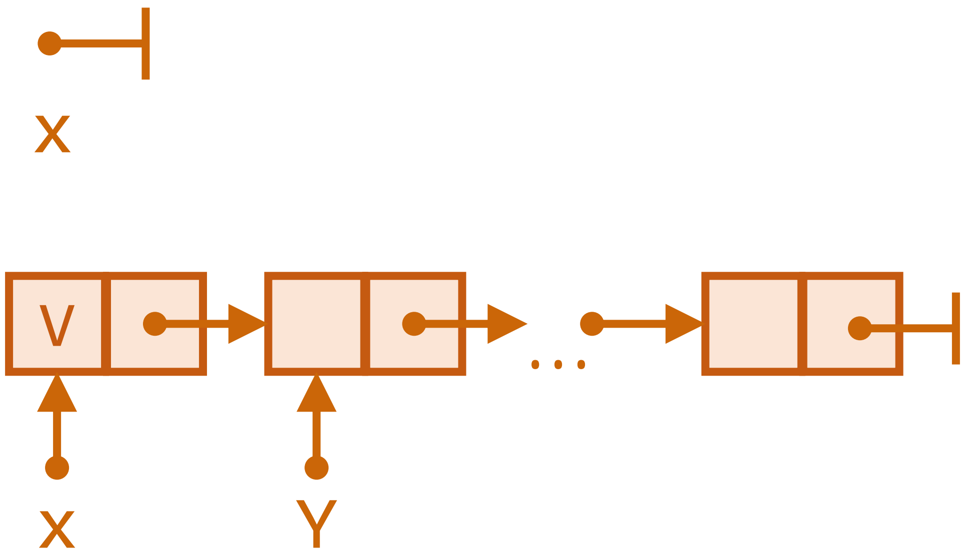
$\}$



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)
                  }
}
```



this tutorial

1. example: swap

2. intro to separation logic
   2.1. programs and assertions
   2.2. inductive predicates
   2.3. specifying data transformations

3. deductive synthesis

demo 2: dispose a list

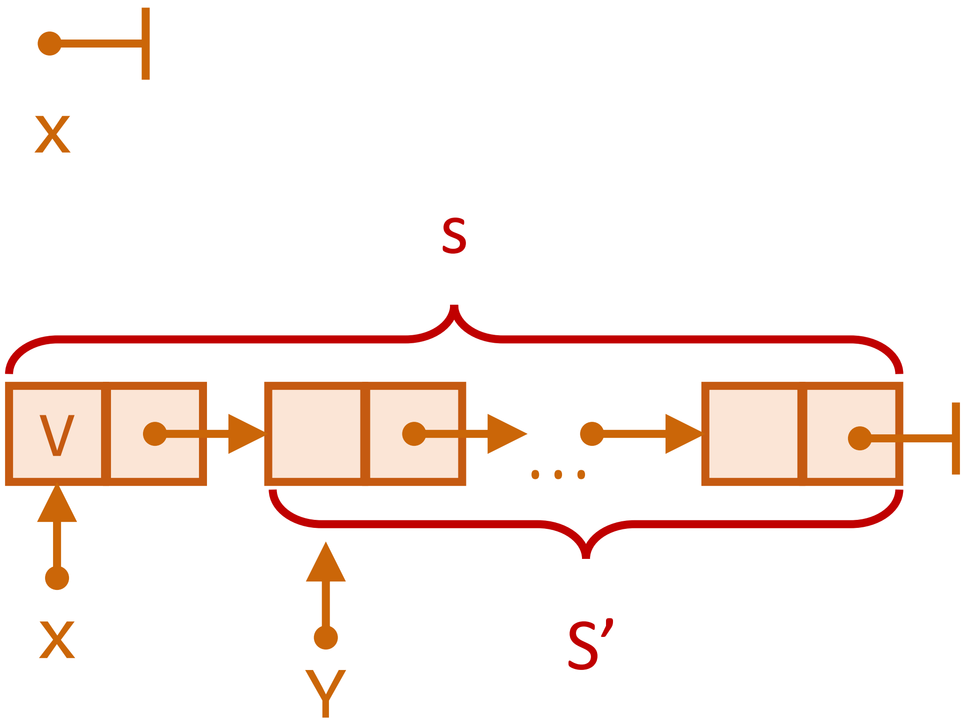
```
void dispose(loc x)
```

{ list(x) }

{ emp }

linked list with elements

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



demo 3: copy a list

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

{ list(x, S) * r ↦ _ }

{ list(x, S) * r ↦ Y * list(Y, S) }

return location

exercise 2: append two lists

```
void append( ??? )
{ ??? }
{ ???}
```

exercise 3: schema migration

```
void single_to_double(loc x)
{ ??? }
{ ??? }
```

singly-linked list
with reserved space in each node

doubly-linked list
at the same address

this tutorial

1. example: swap
2. intro to separation logic
3. deductive synthesis

deductive synthesis

synthesis as proof search

this tutorial

1. example: swap
2. intro to separation logic
3. deductive synthesis
    3.1. proof system
    3.2. proof search

transforming entailment

$$P \rightsquigarrow Q \mid c$$

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

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

(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)

$$\{\mathsf{emp}\} \leadsto \{\mathsf{emp}\} \mid \texttt{skip}$$

(Read)

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

(Frame)

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

(Write)

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

example: swap

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

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

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

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

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

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

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

$$\{\, \mathsf{emp} \,\} \rightsquigarrow \{\, \mathsf{emp} \,\} \mid \texttt{??}$$

$$\rule{8cm}{0.4pt}\ \text{(Frame)}$$

$$\{\, y \mapsto a1 \,\} \rightsquigarrow \{\, y \mapsto a1 \,\} \mid \texttt{??}$$

$$\rule{9cm}{0.4pt}\ \text{(Write)}$$

$$\{\, y \mapsto b1 \,\} \rightsquigarrow \{\, y \mapsto a1 \,\} \mid \texttt{*y = a1; ??}$$

$$\rule{11cm}{0.4pt}\ \text{(Frame)}$$

$$\{\, x \mapsto b1 * y \mapsto b1 \,\} \rightsquigarrow \{\, x \mapsto b1 * y \mapsto a1 \,\} \mid \texttt{??}$$

$$\rule{13cm}{0.4pt}\ \text{(Write)}$$

$$\{\, x \mapsto a1 * y \mapsto b1 \,\} \rightsquigarrow \{\, x \mapsto b1 * y \mapsto a1 \,\} \mid \texttt{*x = b1; ??}$$

$$\rule{14cm}{0.4pt}\ \text{(Read)}$$

$$\{\, x \mapsto a1 * y \mapsto B \,\} \rightsquigarrow \{\, x \mapsto B * y \mapsto a1 \,\} \mid \texttt{let b1 = *y; ??}$$

$$\rule{14cm}{0.4pt}\ \text{(Read)}$$

$$\{\, x \mapsto A * y \mapsto B \,\} \rightsquigarrow \{\, x \mapsto B * y \mapsto A \,\} \mid \texttt{let a1 = *x; ??}$$

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

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

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

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

demo 4: tracing swap

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 }

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

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

??

(Induction)

{ emp }

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

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

{ list^0(x) }

??

(Open)

{ 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 }

```
    ??
```

{ emp }

```
} else {
```

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

```
    ??
```

{ emp }

```
}
```

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

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

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

{ $x = 0$; emp }

```
    ??
```

(Emp)

{ 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) {
```

{ $x = 0$; emp }

```
    skip
```

{ emp }

```
} else {
```

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

```
    ??
```

{ emp }

```
}
```

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

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

```
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) }
}

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

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

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

(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);
```

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

```
      ??
```

{ 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) }
}

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

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

{ $x \neq 0$; list¹ (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)

this tutorial

1. example: swap
2. intro to separation logic
3. deductive synthesis
   3.1. proof system
   3.2. proof search

SuSLik

backtracking search in SSL

+ optimizations

demo 5: backtracking

optimizations

- invertible rules
- early failure
- multi-phase search
- symmetry reduction

optimization: invertible rules

- invertible rules do not restrict the set of derivable programs
- idea: invertible rules need not be backtracked

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

optimization: early failure

- idea: sometimes you know that a goal is unsatisfiable

(Post-Inconsistent)

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

optimization: multi-phase search

- unfolding phase: deals with inductive predicates
  (Open), (Close), (Call), (Frame)
- flat phase: deals with points-to and blocks
  (Write), (Call), (Alloc), (Free), (Frame)
- idea: if unfolding phase cannot eliminate all predicates, give up

optimization: symmetry reduction

- sometimes rule applications commute

$$\dfrac{\dfrac{\{ a \mapsto 0 \} \rightsquigarrow \{ b \mapsto 0 \} \mid ??}{\{ y \mapsto b * a \mapsto 0 \} \rightsquigarrow \{ y \mapsto b * b \mapsto 0 \} \mid ??}\ \text{(Frame)}}{\{ x \mapsto a * y \mapsto b * a \mapsto 0 \} \rightsquigarrow \{ x \mapsto a * y \mapsto b * b \mapsto 0 \} \mid ??}\ \text{(Frame)}$$

- idea: only allow one order of commuting applications

limitations & future work

- pure synthesis

  $\{ r \mapsto \_ \} \rightsquigarrow \{ r \geq x \land r \geq y ; r \mapsto M \}$ | `c`

  use an off-the-shelf pure synthesizer
- reasoning about inductive predicates

  $\{ P \} \rightsquigarrow \{ Q \}$ | `skip`

  identify decidable fragment
- goal must be inductive

  cyclic proofs to the rescue!

$\{ \mathrm{tree}(t, S) * r \mapsto \_ \}$

```
void flatten(loc t, loc r)
```

$\{ r \mapsto X * \mathrm{list}(X, S) \}$

deductive synthesis with SuSLik

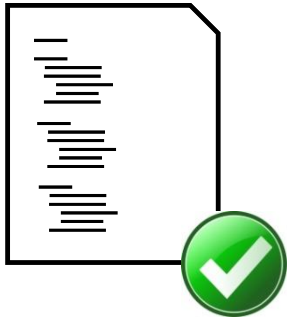
separation logic

☺ reasoning about pointers & aliasing

deductive synthesis

☺ uses specs to guide synthesis

code



☺ provably memory-safe