SuSLik: synthesis of safe pointer-manipulating programs

Nadia Polikarpova

with Ilya Sergey (Yale-NUS), Shachar Itzhaky (Technion), Reuben Rowe (Royal Holloway), and Hila Peleg (UCSD)

UCSDCSE Computer Science and Engineering

pointer-manipulating programs

operating systems

network / security protocols

browsers

pointer-manipulating programs

operating systems

network / security protocols

browsers

☺ efficient

pointer-manipulating programs

operating systems

network / security protocols

browsers

☺ efficient

☹ hard to write

☹ memory safety bugs

pointer-manipulating programs

operating systems

network / security protocols

browsers

☺ efficient

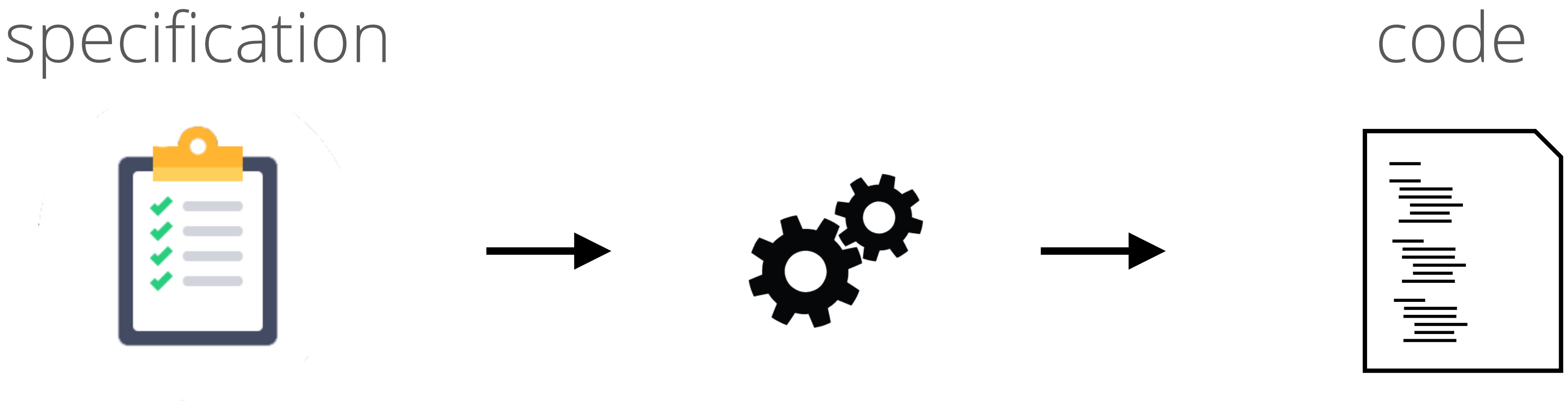
☹ hard to write

☹ memory safety bugs

program synthesis to the rescue

specification

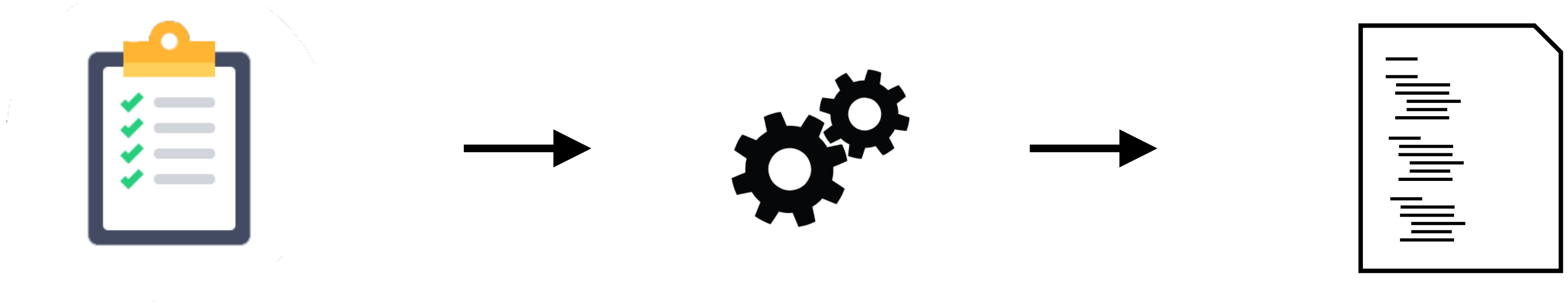
program synthesis to the rescue



program synthesis to the rescue

specification

code



☺ easy to write

program synthesis to the rescue

specification

code

☺ easy to write

☺ efficient
☺ backwards compatible
☺ provably memory-safe

program synthesis to the rescue

specification

code

☺ easy to write

☹ verbose
☹ unstructured
☹ pointers & aliasing

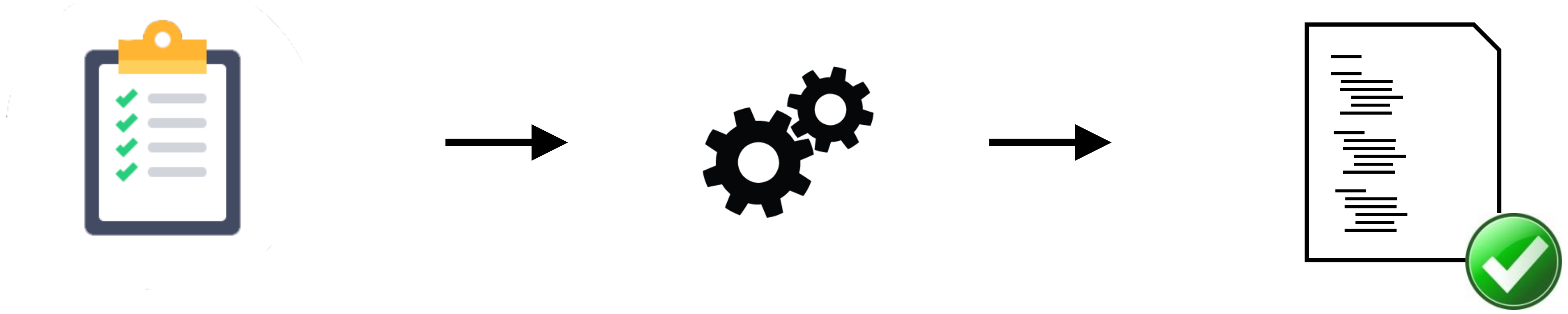
☺ efficient
☺ backwards compatible
☺ provably memory-safe

SuSLik

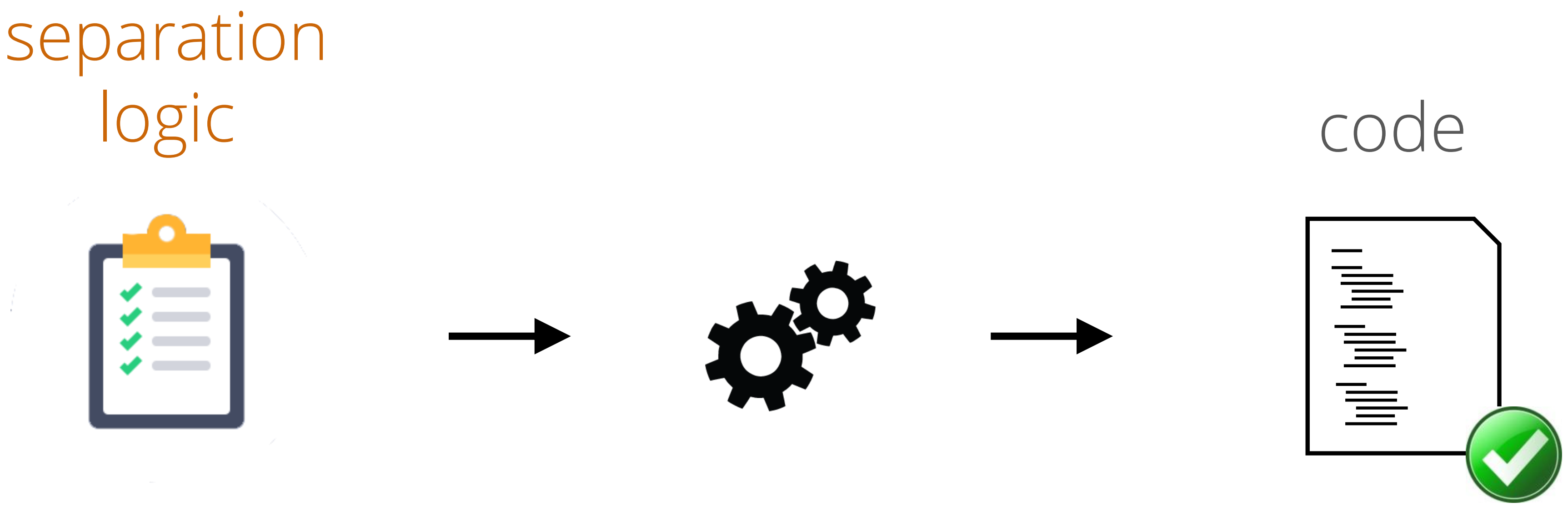
Synthesis Using Separation Logik

our solution

specification code



our solution



our solution

separation logic

code

☺ reasoning about pointers & aliasing

our solution

separation logic

deductive synthesis

code

☺ reasoning about pointers & aliasing

our solution

separation logic

☺ reasoning about pointers & aliasing

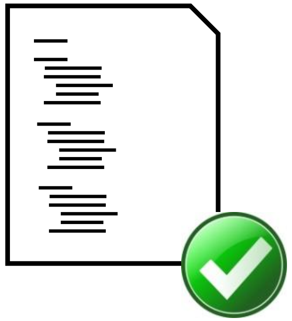
→

deductive synthesis

☺ uses specs to guide synthesis

→

code

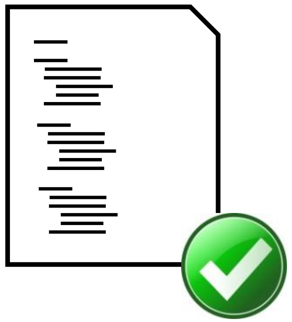


our solution

separation logic

deductive synthesis

code



☺ reasoning about pointers & aliasing

☺ uses specs to guide synthesis

examples

1. swap

examples

1. swap
2. recursive programs

examples

1. swap
2. recursive programs
3. auxiliary functions

examples

1. swap
2. recursive programs
3. auxiliary functions

Polikarpova, Sergey [POPL'19]

examples

1. swap
2. recursive programs
3. auxiliary functions

Polikarpova, Sergey [POPL'19]

in submission

examples

1. swap
2. recursive programs
3. auxiliary functions

swap

Swap values of two *distinct* pointers

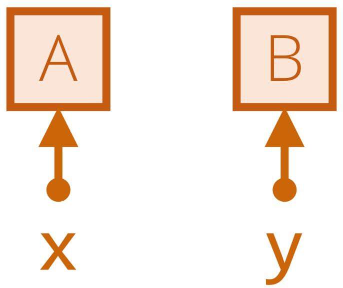
```
void swap(loc x, loc y)
```

swap

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

swap

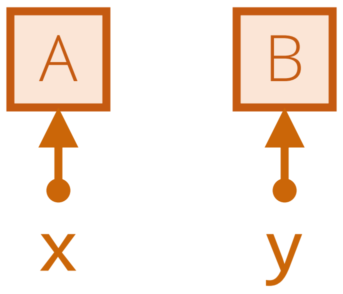
start state:



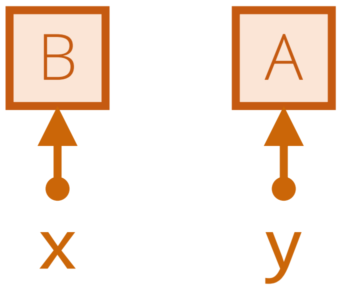

```
void swap(loc x, loc y)
```

swap

start state:



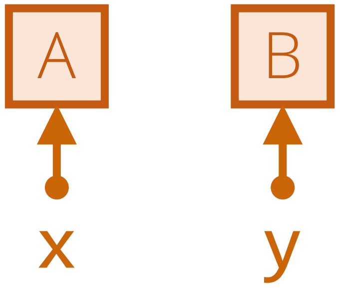
end state:



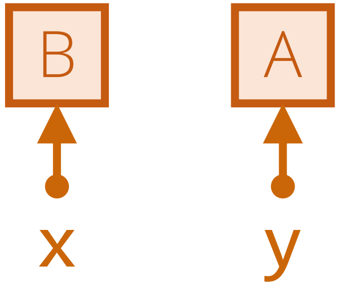

```
void swap(loc x, loc y)
```

swap

start state:



end state:



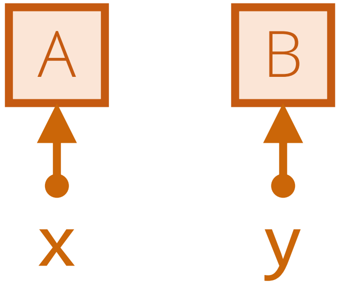
in separation logic:

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

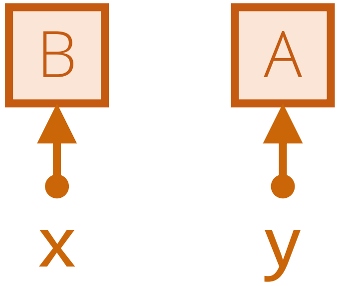
```
void swap(loc x, loc y)
```

swap

start state:



end state:



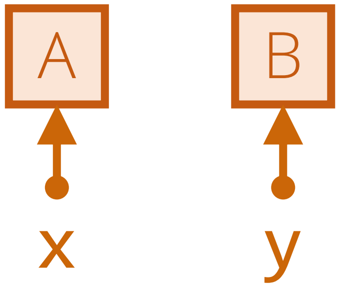
in separation logic:

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

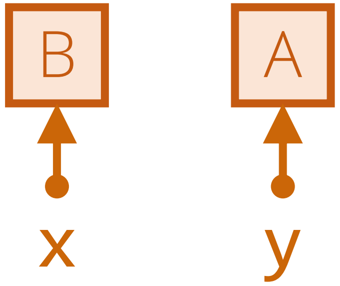
```
void swap(loc x, loc y)
```

swap

start state:



end state:



in separation logic:

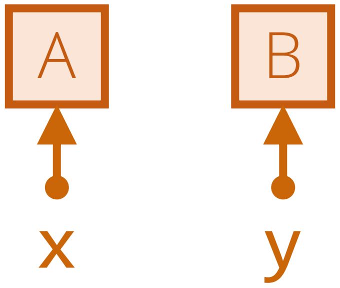
$\{ x \mapsto A * y \mapsto B \}$ precondition

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

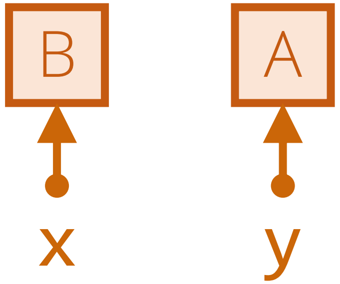
$\{ x \mapsto B * y \mapsto A \}$

swap

start state:



end state:



in separation logic:

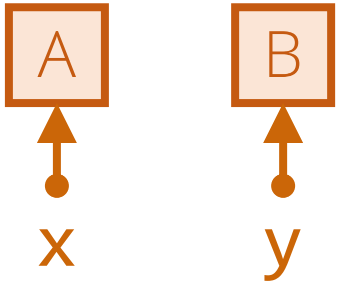
$\{ x \mapsto A * y \mapsto B \}$ precondition

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

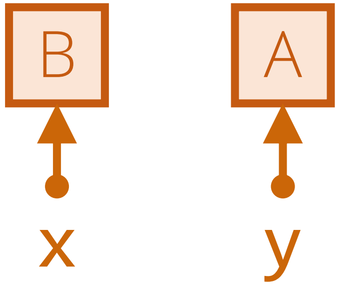
$\{ x \mapsto B * y \mapsto A \}$ postcondition

swap

start state:



end state:



in separation logic:

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

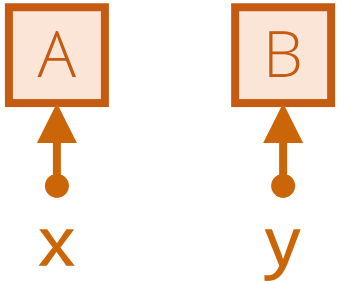
separately

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

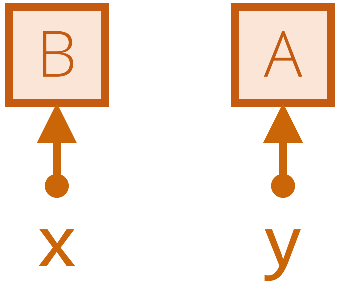
$\{ x \mapsto B \boxed{*} y \mapsto A \}$ postcondition

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

swap

specification

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

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

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

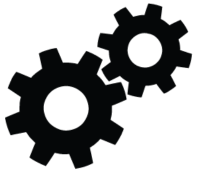
swap

specification

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

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

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



code

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

swap

specification

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

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

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

?

code

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

swap

specification

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

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

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

→ **?** →

code

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

insight: the spec tells us what to do!

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

??

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

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

??

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

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

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

??

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

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

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

??

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

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

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

??

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

```
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;
```

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

??

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

```
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;
```

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

??

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

```
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;
}
```

examples

1. swap
2. recursive functions
3. auxiliary functions

dispose a list

{ list(x) }

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

{ emp }

dispose a list

{ list(x) }

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

{ emp }

→

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

singly- to doubly-linked list

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

```
void sll_to_dll(loc r)
```

$\{ r \mapsto Y * \mathrm{dlist}(Y, S) \}$

singly- to doubly-linked list

$\{\ r \mapsto x * \mathrm{list}(x, S)\ \}$ singly-linked list at x with set of elements S

```
void sll_to_dll(loc r)
```

$\{\ r \mapsto Y * \mathrm{dlist}(Y, S)\ \}$

singly- to doubly-linked list

{ r ↦ x * list(x, S) } singly-linked list at x with set of elements S

```
void sll_to_dll(loc r)
```

{ r ↦ Y * dlist(Y, S) } doubly-linked list at Y with set of elements S

singly- to doubly-linked list

{ r ↦ x * list(x, S) } singly-linked list at x with set of elements S

```
void sll_to_dll(loc r)
```

{ r ↦ Y * dlist(Y, S) } doubly-linked list at Y with set of elements S

return location

singly- to doubly-linked list

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

```
void sll_to_dll(loc r)
```

{ $r \mapsto Y$ * dlist(Y, S) }

singly- to doubly-linked list

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

```
void sll_to_dll(loc r)
```

$\{ r \mapsto Y * dlist(Y, S) \}$

$\longrightarrow$

```
void sll_to_dll(loc r) {
  let x = *r;
  if (x != 0) {
    let v = *x;
    let n = *(x + 1);
    *r = n;
    sll_to_dll(r);
    let y1 = *r;
    let y = malloc(3);
    free(x);
    *r = y;
    *(y + 2) = 0;
    *y = v;
    if (y1 == 0) {
      *(y + 1) = 0;
    } else {
      *(y1 + 2) = y;
      *(y + 1) = y1;
    }}}
```

examples

1. swap
2. recursive functions
3. auxiliary functions

flatten a free

$\{ \text{ret} \mapsto t * \text{tree}(t) \}$

```
void flatten(loc ret)
```

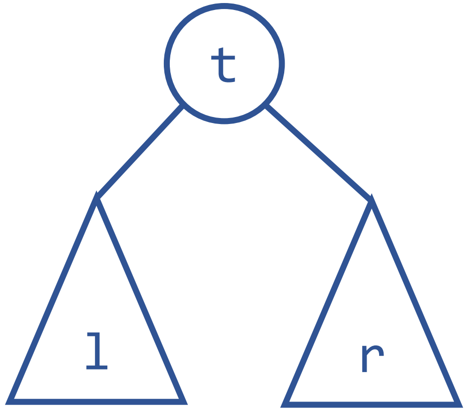
$\{ \text{ret} \mapsto X * \text{list}(X) \}$

flatten a free

{ ret ↦ t * tree(t) }

```
void flatten(loc ret)
```

{ ret ↦ X * list(X) }

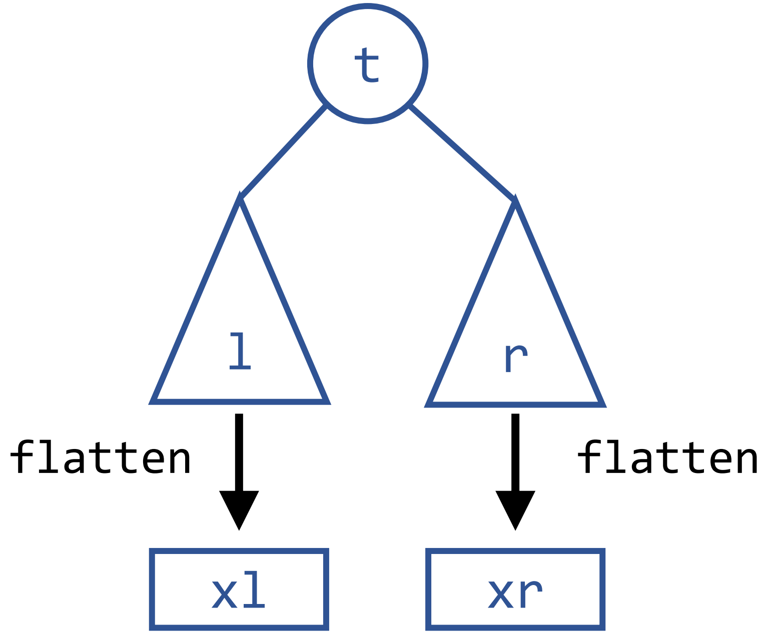


flatten a free

{ ret ↦ t * tree(t) }

```
void flatten(loc ret)
```

{ ret ↦ X * list(X) }

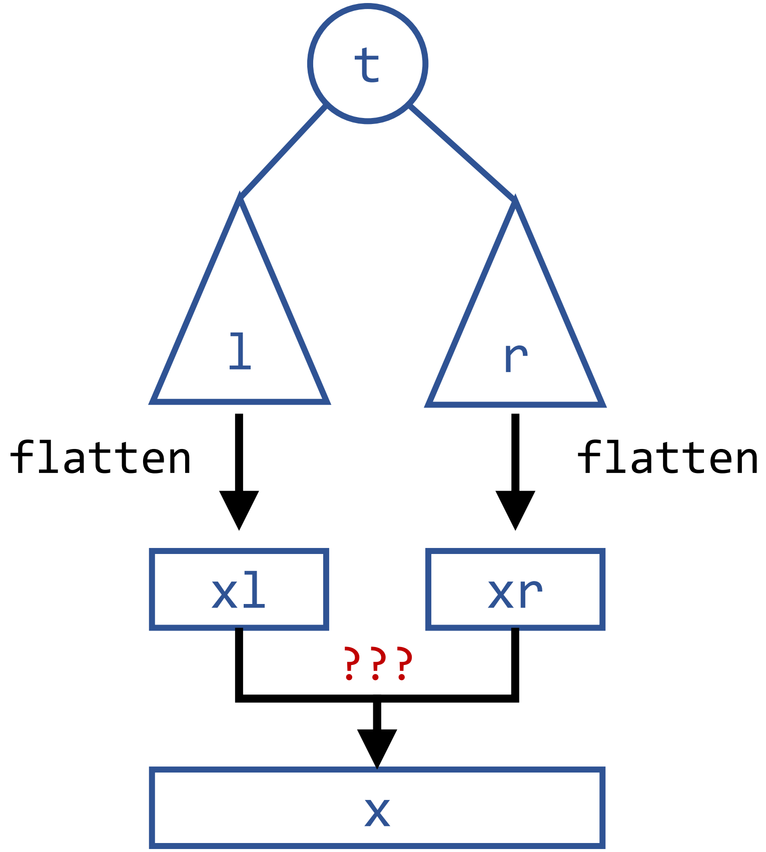


flatten a free

{ ret ↦ t * tree(t) }

```
void flatten(loc ret)
```

{ ret ↦ X * list(X) }



flatten a tree

$\{ret \mapsto t * tree(t)\}$ `flatten(ret)` $\{ret \mapsto x * list(x)\}$

flatten a tree

{ret $\mapsto$ t * tree(t) } `flatten(ret)` {ret $\mapsto$ x * list(x) }

```
void flatten (loc ret, loc t) {
  if (t != 0) {
    let v = *t;
    let l = *(t + 1);
    let r = *(t + 2);
    flatten(ret, l);
    aux(r, v, ret, t);
  }
}
```

flatten a tree

{ret ↦ t * tree(t) } `flatten(ret)` {ret ↦ x * list(x) }

```
void flatten (loc ret, loc t) {
  if (t != 0) {
    let v = *t;
    let l = *(t + 1);
    let r = *(t + 2);
    flatten(ret, l);
    aux(r, v, ret, t);
  }
}
```

```
// cons v to flattened r
// and append result to list pointed by ret
void aux (loc r, int v, loc ret, loc t) {
  let x1 = *ret;
  if (x1 == 0) {
    flatten(ret, r);
    let x2 = *ret;
    let x = malloc(2); *x = v; *(x + 1) = x2;
    *ret = x;
    free(t);
  } else {
    let n = *(x1 + 1);
    *ret = n;
    aux(r, v, ret, t);
    let x = *ret;
    *(x1 + 1) = x;
    *ret = x1;
  }
}
```

cyclic synthesis

SSL + cyclic proofs

cyclic synthesis

SSL + cyclic proofs

Brotherston, Bornat, Calcagno: *Cyclic proofs of program termination in separation logic.* [POPL'08]

Rowe, Brotherston: *Automatic cyclic termination proofs for recursive procedures in separation logic.* [CPP'17]

dispose revisited

{ list(x) }

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

{ emp }

dispose: cyclic proof

{ list(x) } `dispose(x)` { emp }

$\{ \text{list}(x) \} \rightsquigarrow \{ \text{emp} \}$

dispose: cyclic proof

{ list(x) } `dispose(x)` { emp }

$$\frac{}{\{\ \mathrm{list}(x)\ \}\ \rightsquigarrow\ \{\ \mathrm{emp}\ \}}\ (\text{Open})$$

dispose: cyclic proof

{ list(x) } `dispose(x)` { emp }

$$\frac{\{\ \mathrm{emp}\ \} \rightsquigarrow \{\ \mathrm{emp}\ \}}{\{\ \mathrm{list}(x)\ \} \rightsquigarrow \{\ \mathrm{emp}\ \}}\ (\text{Open})$$

dispose: cyclic proof

$\{\, \mathrm{list}(x) \,\}$ `dispose(x)` $\{\, \mathrm{emp} \,\}$

$$\dfrac{\{\, \mathrm{emp} \,\} \rightsquigarrow \{\, \mathrm{emp} \,\} \qquad \{[x, 2] * x \mapsto v * (x+1) \mapsto x1 * \mathrm{list}(x1)\} \rightsquigarrow \{\, \mathrm{emp} \,\}}{\{\, \mathrm{list}(x) \,\} \rightsquigarrow \{\, \mathrm{emp} \,\}}\ (\mathrm{Open})$$

dispose: cyclic proof

$\{\,\mathsf{list}(x)\,\}$ `dispose(x)` $\{\,\mathsf{emp}\,\}$

$$
\dfrac{\dfrac{}{\{\,\mathsf{emp}\,\} \rightsquigarrow \{\,\mathsf{emp}\,\}}\,(\mathrm{Emp}) \qquad \{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \mathsf{list}(x1)\} \rightsquigarrow \{\,\mathsf{emp}\,\}}{\{\,\mathsf{list}(x)\,\} \rightsquigarrow \{\,\mathsf{emp}\,\}}\,(\mathrm{Open})
$$

dispose: cyclic proof

$\{ \mathrm{list}(x) \}$ `dispose(x)` $\{ \mathrm{emp} \}$

$$\dfrac{\dfrac{\text{(Emp)}}{\{ \mathrm{emp} \} \rightsquigarrow \{ \mathrm{emp} \}} \qquad \dfrac{}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \mathrm{list}(x1)\} \rightsquigarrow \{ \mathrm{emp} \}}\text{(Free)}}{\{ \mathrm{list}(x) \} \rightsquigarrow \{ \mathrm{emp} \}}\text{(Open)}$$

dispose: cyclic proof

$\{ \mathrm{list}(x) \}$ `dispose(x)` $\{ \mathrm{emp} \}$

$$\dfrac{\dfrac{\text{(Emp)}}{\{ \mathrm{emp} \} \rightsquigarrow \{ \mathrm{emp} \}} \qquad \dfrac{\{ \mathrm{list}(x1) \} \rightsquigarrow \{ \mathrm{emp} \}}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \mathrm{list}(x1)\} \rightsquigarrow \{ \mathrm{emp} \}}\text{(Free)}}{\{ \mathrm{list}(x) \} \rightsquigarrow \{ \mathrm{emp} \}}\text{(Open)}$$

dispose: cyclic proof

{ list(x) } `dispose(x)` { emp }

$$\dfrac{\dfrac{}{\{\,\mathrm{emp}\,\} \rightsquigarrow \{\,\mathrm{emp}\,\}}\,(\mathrm{Emp}) \qquad \dfrac{\{\,\mathrm{list}(x1)\,\} \rightsquigarrow \{\,\mathrm{emp}\,\}}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \mathrm{list}(x1)\} \rightsquigarrow \{\,\mathrm{emp}\,\}}\,(\mathrm{Free})}{\{\,\mathrm{list}(x)\,\} \rightsquigarrow \{\,\mathrm{emp}\,\}}\,(\mathrm{Open})$$

dispose: cyclic proof

{ list(x) } `dispose(x)` { emp }

cycle generates a recursive call!

$$\dfrac{\dfrac{(\text{Emp})}{\{\ \mathrm{emp}\ \} \leadsto \{\ \mathrm{emp}\ \}} \qquad \dfrac{\{\ \mathrm{list}(x1)\ \} \leadsto \{\ \mathrm{emp}\ \} \ \Big|\ \texttt{dispose(x1)}}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \mathrm{list}(x1)\} \leadsto \{\ \mathrm{emp}\ \}}\,(\text{Free})}{\{\ \mathrm{list}(x)\ \} \leadsto \{\ \mathrm{emp}\ \}}\,(\text{Open})$$

dispose two

{ list(x) * list(y) }

```
void dispose2(loc x, loc y)
```

{ emp }

dispose two

$\{ \text{list}(x) * \text{list}(y) \}$ `dispose2(x, y)` $\{ \text{emp} \}$

$\{ \text{list}(x) * \text{list}(y) \} \rightsquigarrow \{ \text{emp} \}$

dispose two

{ list(x) * list(y) } `dispose2(x, y)` { emp }

$$\frac{}{\{\ \mathrm{list}(x) * \mathrm{list}(y)\ \} \rightsquigarrow \{\ \mathrm{emp}\ \}}\ (\mathrm{Open})$$

dispose two

$\{ \text{list}(x) * \text{list}(y) \}$ `dispose2(x, y)` $\{ \text{emp} \}$

$$\dfrac{\{ \text{list}(y) \} \rightsquigarrow \{ \text{emp} \} \qquad \{[x, 2] * x \mapsto v * (x+1) \mapsto x1 * \text{list}(x1) * \text{list}(y)\} \rightsquigarrow \{ \text{emp} \}}{\{ \text{list}(x) * \text{list}(y) \} \rightsquigarrow \{ \text{emp} \}} \text{(Open)}$$

dispose two

$\{\,\mathrm{list}(x) * \mathrm{list}(y)\,\}$ `dispose2(x, y)` $\{\,\mathrm{emp}\,\}$

$$\dfrac{\{\,\mathrm{list}(y)\,\} \rightsquigarrow \{\,\mathrm{emp}\,\} \qquad \dfrac{}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \mathrm{list}(x1) * \mathrm{list}(y)\} \rightsquigarrow \{\,\mathrm{emp}\,\}}\;(\mathrm{Free})}{\{\,\mathrm{list}(x) * \mathrm{list}(y)\,\} \rightsquigarrow \{\,\mathrm{emp}\,\}}\;(\mathrm{Open})$$

dispose two

$\{\, \mathrm{list}(x) * \mathrm{list}(y) \,\}$ `dispose2(x, y)` $\{\, \mathrm{emp} \,\}$

$$\dfrac{\{\, \mathrm{list}(y) \,\} \leadsto \{\, \mathrm{emp} \,\} \qquad \dfrac{\{\, \mathrm{list}(x1) * \mathrm{list}(y) \,\} \leadsto \{\, \mathrm{emp} \,\}}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \mathrm{list}(x1) * \mathrm{list}(y)\} \leadsto \{\, \mathrm{emp} \,\}}\ (\mathrm{Free})}{\{\, \mathrm{list}(x) * \mathrm{list}(y) \,\} \leadsto \{\, \mathrm{emp} \,\}}\ (\mathrm{Open})$$

dispose two

$\{ \text{list}(x) * \text{list}(y) \}$ `dispose2(x, y)` $\{ \text{emp} \}$

$$\dfrac{\{ \text{list}(y) \} \leadsto \{ \text{emp} \} \qquad \dfrac{\{ \text{list}(x1) * \text{list}(y) \} \leadsto \{ \text{emp} \}}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \text{list}(x1) * \text{list}(y)\} \leadsto \{ \text{emp} \}}\,\text{(Free)}}{\{ \text{list}(x) * \text{list}(y) \} \leadsto \{ \text{emp} \}}\,\text{(Open)}$$

dispose two

$\{\,\mathrm{list}(x) * \mathrm{list}(y)\,\}$ `dispose2(x, y)` $\{\,\mathrm{emp}\,\}$

$$\dfrac{\{\,\mathrm{list}(y)\,\} \rightsquigarrow \{\,\mathrm{emp}\,\} \qquad \dfrac{\{\,\mathrm{list}(x1) * \mathrm{list}(y)\,\} \rightsquigarrow \{\,\mathrm{emp}\,\} \mid \texttt{dispose2(x1, y)}}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \mathrm{list}(x1) * \mathrm{list}(y)\} \rightsquigarrow \{\,\mathrm{emp}\,\}}\;(\mathrm{Free})}{\{\,\mathrm{list}(x) * \mathrm{list}(y)\,\} \rightsquigarrow \{\,\mathrm{emp}\,\}}\;(\mathrm{Open})$$

dispose two

$\{ \mathrm{list}(x) * \mathrm{list}(y) \}$ `dispose2(x, y)` $\{ \mathrm{emp} \}$

$$\dfrac{\dfrac{}{\{ \mathrm{list}(y) \} \rightsquigarrow \{ \mathrm{emp} \}}\,(\mathrm{Open}) \qquad \dfrac{\{ \mathrm{list}(x1) * \mathrm{list}(y) \} \rightsquigarrow \{ \mathrm{emp} \} \mid \texttt{dispose2(x1, y)}}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \mathrm{list}(x1) * \mathrm{list}(y)\} \rightsquigarrow \{ \mathrm{emp} \}}\,(\mathrm{Free})}{\{ \mathrm{list}(x) * \mathrm{list}(y) \} \rightsquigarrow \{ \mathrm{emp} \}}\,(\mathrm{Open})$$

dispose two

$$\{\, \mathrm{list}(x) * \mathrm{list}(y) \,\}\ \texttt{dispose2(x, y)}\ \{\, \mathrm{emp} \,\}$$

$$\{\, \mathrm{emp} \,\} \leadsto \{\, \mathrm{emp} \,\} \qquad \{[y, 2] * y \mapsto u * (y+1) \mapsto y1 * \mathrm{list}(y1)\} \leadsto \{\, \mathrm{emp} \,\}$$

$$\dfrac{\dfrac{}{\{\, \mathrm{list}(y) \,\} \leadsto \{\, \mathrm{emp} \,\}}\ (\mathrm{Open}) \qquad \dfrac{\{\, \mathrm{list}(x1) * \mathrm{list}(y) \,\} \leadsto \{\, \mathrm{emp} \,\} \mid \texttt{dispose2(x1, y)}}{\{[x, 2] * x \mapsto v * (x+1) \mapsto x1 * \mathrm{list}(x1) * \mathrm{list}(y)\} \leadsto \{\, \mathrm{emp} \,\}}\ (\mathrm{Free})}{\{\, \mathrm{list}(x) * \mathrm{list}(y) \,\} \leadsto \{\, \mathrm{emp} \,\}}\ (\mathrm{Open})$$

dispose two

$\{ \mathsf{list}(x) * \mathsf{list}(y) \}$ `dispose2(x, y)` $\{ \mathsf{emp} \}$

$$\frac{}{\{ \mathsf{emp} \} \rightsquigarrow \{ \mathsf{emp} \}}\ (\mathrm{Emp}) \qquad \{[y, 2] * y \mapsto u * (y+1) \mapsto y1 * \mathsf{list}(y1)\} \rightsquigarrow \{ \mathsf{emp} \}$$

$$\dfrac{(\mathrm{Open})\ \dfrac{}{\{ \mathsf{list}(y) \} \rightsquigarrow \{ \mathsf{emp} \}} \qquad \dfrac{\{ \mathsf{list}(x1) * \mathsf{list}(y) \} \rightsquigarrow \{ \mathsf{emp} \} \mid \texttt{dispose2(x1, y)}}{\{[x, 2] * x \mapsto v * (x+1) \mapsto x1 * \mathsf{list}(x1) * \mathsf{list}(y)\} \rightsquigarrow \{ \mathsf{emp} \}}\ (\mathrm{Free})}{\{ \mathsf{list}(x) * \mathsf{list}(y) \} \rightsquigarrow \{ \mathsf{emp} \}}\ (\mathrm{Open})$$

dispose two

$\{\ \mathrm{list}(x) * \mathrm{list}(y)\ \}$ `dispose2(x, y)` $\{\ \mathrm{emp}\ \}$

$$\dfrac{}{\{\ \mathrm{emp}\ \} \rightsquigarrow \{\ \mathrm{emp}\ \}}\ (\mathrm{Emp}) \qquad \dfrac{}{\{[y, 2] * y \mapsto u * (y+1) \mapsto y1 * \mathrm{list}(y1)\} \rightsquigarrow \{\ \mathrm{emp}\ \}}\ (\mathrm{Free})$$

$$\dfrac{(\mathrm{Open})\ \dfrac{}{\{\ \mathrm{list}(y)\ \} \rightsquigarrow \{\ \mathrm{emp}\ \}} \qquad \dfrac{\{\ \mathrm{list}(x1) * \mathrm{list}(y)\ \} \rightsquigarrow \{\ \mathrm{emp}\ \} \mid \texttt{dispose2(x1, y)}}{\{[x, 2] * x \mapsto v * (x+1) \mapsto x1 * \mathrm{list}(x1) * \mathrm{list}(y)\} \rightsquigarrow \{\ \mathrm{emp}\ \}}\ (\mathrm{Free})}{\{\ \mathrm{list}(x) * \mathrm{list}(y)\ \} \rightsquigarrow \{\ \mathrm{emp}\ \}}\ (\mathrm{Open})$$

dispose two

$\{\ \mathsf{list}(x) * \mathsf{list}(y)\ \}$ `dispose2(x, y)` $\{\ \mathsf{emp}\ \}$

$$\dfrac{\text{(Emp)}}{\{\ \mathsf{emp}\ \} \rightsquigarrow \{\ \mathsf{emp}\ \}} \qquad \dfrac{\{\ \mathsf{list}(y1)\ \} \rightsquigarrow \{\ \mathsf{emp}\ \}}{\{[y, 2] * y \mapsto u * (y + 1) \mapsto y1 * \mathsf{list}(y1)\} \rightsquigarrow \{\ \mathsf{emp}\ \}}\ \text{(Free)}$$

$$\dfrac{\text{(Open)}\ \dfrac{}{\{\ \mathsf{list}(y)\ \} \rightsquigarrow \{\ \mathsf{emp}\ \}} \qquad \dfrac{\{\ \mathsf{list}(x1) * \mathsf{list}(y)\ \} \rightsquigarrow \{\ \mathsf{emp}\ \} \mid \texttt{dispose2(x1, y)}}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * \mathsf{list}(x1) * \mathsf{list}(y)\} \rightsquigarrow \{\ \mathsf{emp}\ \}}\ \text{(Free)}}{\{\ \mathsf{list}(x) * \mathsf{list}(y)\ \} \rightsquigarrow \{\ \mathsf{emp}\ \}}\ \text{(Open)}$$

dispose two

$\{\ \mathrm{list}(x) * \mathrm{list}(y)\ \}$ `dispose2(x, y)` $\{\ \mathrm{emp}\ \}$

$$\frac{}{\{\ \mathrm{emp}\ \} \leadsto \{\ \mathrm{emp}\ \}}\ (\mathrm{Emp}) \qquad \frac{\{\ \mathrm{list}(y1)\ \} \leadsto \{\ \mathrm{emp}\ \}}{\{[y, 2] * y \mapsto u * (y+1) \mapsto y1 * \mathrm{list}(y1)\} \leadsto \{\ \mathrm{emp}\ \}}\ (\mathrm{Free})$$

$$\frac{(\mathrm{Open})\ \dfrac{}{\{\ \mathrm{list}(y)\ \} \leadsto \{\ \mathrm{emp}\ \}} \qquad \dfrac{\{\ \mathrm{list}(x1) * \mathrm{list}(y)\ \} \leadsto \{\ \mathrm{emp}\ \} \mid \texttt{dispose2(x1, y)}}{\{[x, 2] * x \mapsto v * (x+1) \mapsto x1 * \mathrm{list}(x1) * \mathrm{list}(y)\} \leadsto \{\ \mathrm{emp}\ \}}\ (\mathrm{Free})}{\{\ \mathrm{list}(x) * \mathrm{list}(y)\ \} \leadsto \{\ \mathrm{emp}\ \}}\ (\mathrm{Open})$$

dispose two

$\{\ list(x) * list(y)\ \}$ `dispose2(x, y)` $\{\ emp\ \}$

internal cycle: extract subtree into auxiliary function!

$$
\dfrac{
  \dfrac{\dfrac{}{\{\ emp\ \} \rightsquigarrow \{\ emp\ \}}\,(\text{Emp})
  \qquad
  \dfrac{\{\ list(y1)\ \} \rightsquigarrow \{\ emp\ \} \mid \texttt{dispose(y1)}}{\{[y, 2] * y \mapsto u * (y + 1) \mapsto y1 * list(y1)\} \rightsquigarrow \{\ emp\ \}}\,(\text{Free})}
  {\{\ list(y)\ \} \rightsquigarrow \{\ emp\ \}}\,(\text{Open})
  \qquad
  \dfrac{\{\ list(x1) * list(y)\ \} \rightsquigarrow \{\ emp\ \} \mid \texttt{dispose2(x1, y)}}{\{[x, 2] * x \mapsto v * (x + 1) \mapsto x1 * list(x1) * list(y)\} \rightsquigarrow \{\ emp\ \}}\,(\text{Free})
}{
  \{\ list(x) * list(y)\ \} \rightsquigarrow \{\ emp\ \}
}\,(\text{Open})
$$

synthesis with auxiliary functions

```
void dispose2(loc x, loc y) {
  if (x == 0) {
    dispose(y)
  } else {
    let x1 = *(x + 1);
    free x;
    dispose(x1, y)
  }
}
```

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

deductive synthesis with SuSLik

separation logic

☺ reasoning about pointers & aliasing

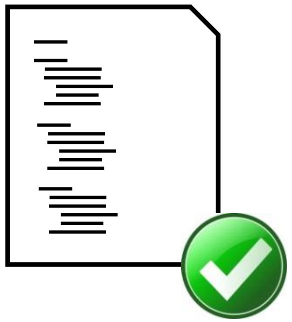
→

deductive synthesis

☺ uses specs to guide synthesis

→

code



☺ provably memory-safe