Program Verification using Separation Logic

Lecture 3 : Frame Rule and Interprocedural Analysis

Hongseok Yang (Queen Mary, Univ. of London)

Proving about Procedure Calls

$$\{P\}f()\{Q\} \;\vdash\; \{P\}y{=}f()\{Q[y/\mathsf{ret}]\}$$

Proving about Procedure Calls

$$\{P\}f()\{Q\} \vdash \{P\}y{=}f()\{Q[y/\mathsf{ret}]\}$$

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

- Verification of the caller program:

{emp}

```
w=create_list();

x=create_list();

y=create_list();

z=create_list()
```

{ls (w,0) * ls (x,0) * ls(y,0) * ls(z,0)}

Proving about Procedure Calls

$$\{P\}f()\{Q\} \vdash \{P\}y{=}f()\{Q[y/\mathsf{ret}]\}$$

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

- Verification of the caller program:

{emp}

```
w=create_list();
```

{ls(w,0)}

```
x=create_list();

y=create_list();

z=create_list()
```

{ls (w,0) * ls (x,0) * ls(y,0) * ls(z,0)}

Proving about Procedure Calls

$$\{P\}f()\{Q\} \vdash \{P\}y{=}f()\{Q[y/\mathsf{ret}]\}$$

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

Preconditions do not match.

- Verification of the caller program:

{emp}

```
w=create_list();
```

{ls(w,0)}

```
x=create_list();

y=create_list();

z=create_list()
```

{ls (w,0) * ls (x,0) * ls(y,0) * ls(z,0)}

Proving about Procedure Calls

$$\{P\}f()\{Q\} \vdash \{P\}y{=}f()\{Q[y/\mathsf{ret}]\}$$

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

- Verification of the caller program:

{emp}

```
w=create_list();
```

{ls(w,0)}

```
x=create_list();
```

{ls (w,0) * ls (x,0)}

```
y=create_list();


z=create_list()
```

{ls (w,0) * ls (x,0) * ls(y,0) * ls(z,0)}

Proving about Procedure Calls

$$\{P\}f()\{Q\} \vdash \{P\}y{=}f()\{Q[y/\mathsf{ret}]\}$$

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

Preconditions do not match again!

- Verification of the caller program:

{emp}

```
w=create_list();
```

{ls(w,0)}

```
x=create_list();
```

{ls (w,0) * ls (x,0)}

```
y=create_list();


z=create_list()
```

{ls (w,0) * ls (x,0) * ls(y,0) * ls(z,0)}

Proving about Procedure Calls

$$\{P\}f()\{Q\} \vdash \{P\}y{=}f()\{Q[y/\mathsf{ret}]\}$$

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

- Verification of the caller program:

{emp}

```
w=create_list();
```

{ls(w,0)}

```
x=create_list();
```

{ls (w,0) * ls (x,0)}

```
y=create_list();
```

{ls (w,0) * ls (x,0) * ls(y,0)}

```
z=create_list()
```

{ls (w,0) * ls (x,0) * ls(y,0) * ls(z,0)}

Proving about Procedure Calls

$$\{P\}f()\{Q\} \vdash \{P\}y{=}f()\{Q[y/\mathrm{ret}]\}$$

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

Yes, the same problem. Third times!

- Verification of the caller program:

{emp}

```
w=create_list();
```

{ls(w,0)}

```
x=create_list();
```

{ls (w,0) * ls (x,0)}

```
y=create_list();
```

{ls (w,0) * ls (x,0) * ls(y,0)}

```
z=create_list()
```

{ls (w,0) * ls (x,0) * ls(y,0) * ls(z,0)}

Proving about Procedure Calls

$$\{P\}f()\{Q\} \vdash \{P\}y{=}f()\{Q[y/\mathsf{ret}]\}$$

- Spec for create_list:

{emp}

```
create_list()
```

{Is(ret,0)}

Yes, the same problem. Third times!

- Verification of the caller program:

{emp}

```
w=create_list();
```

{Is(w,0)}

```
x=create_list();
```

{Is (w,0) * Is (x,0)}

```
y=create_list();
```

{Is (w,0) Is (x,0) Is(y,0) Is(z,0)}

1) Need 4 Hoare triple for `create_list`.
2) A static analyzer should keep a table of size 4.
3) However, one triple should be enough.

Goal of Lecture 3

- Understand the frame rule and how the rule helps us to solve our problem.
- Can describe the basic ideas of an interprocedural analysis and the optimization based on the frame rule.

Frame Rule

Language with Simple Procedures

$$\begin{array}{rcl} \mathsf{ret} & \in & \mathsf{Vars} \\ C & ::= & \ldots \mid \texttt{local}\ x.C \mid x{=}f(\vec{E}) \mid \texttt{let}\ f(\vec{x}){=}C\ \texttt{in}\ C \end{array}$$

- Call-by-value procedures as in Java.
- Returning a value is done by the assignment to ret.
- Procedures do not access any global vars.

$$\begin{array}{l} \texttt{let alloc2}() = \\ \quad \texttt{local}\ t_1, t_2. \\ \quad t_1{=}\texttt{new}(1);\ {*}t_1{=}0; \\ \quad t_2{=}\texttt{new}(1);\ {*}t_2{=}t_1; \\ \quad \mathsf{ret}{=}t_2 \\ \texttt{in} \\ \quad x{=}\texttt{alloc2}(); \\ \quad y{=}\texttt{alloc2}() \end{array}$$

Language with Simple Procedures

ret $\in$ Vars

C ::= ... | `local` $x.C$ | $x{=}f(\vec{E})$ | `let` $f(\vec{x}){=}C$ `in` C

- Call-by-value procedures as in Java.
- Returning a value is done by the assignment to ret.
- Procedures do not access any global vars.

```
let alloc2() =
  local t1, t2.
  t1=new(1); *t1=0;
  t2=new(1); *t2=t1;
  ret=t2
in
  x=alloc2();
  y=alloc2()
```

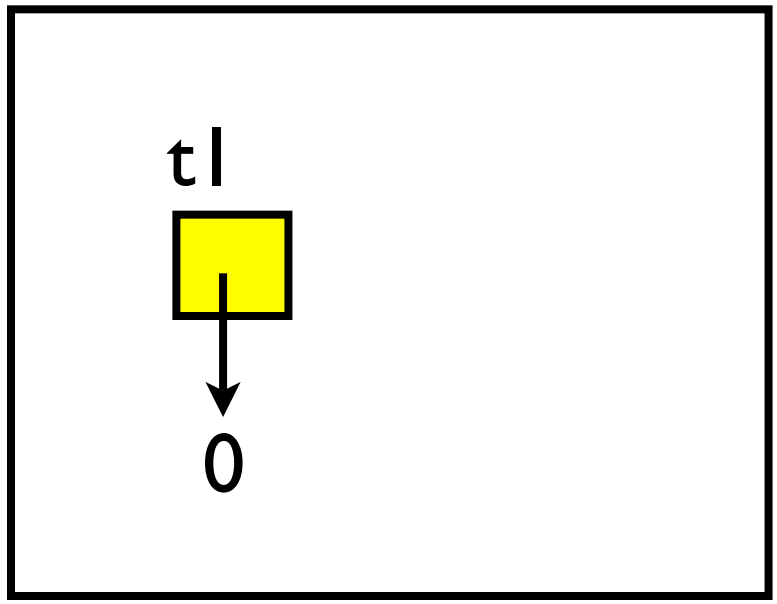
Language with Simple Procedures

ret $\in$ Vars

C ::= ... | `local` $x.C$ | $x{=}f(\vec{E})$ | `let` $f(\vec{x}){=}C$ `in` C

- Call-by-value procedures as in Java.
- Returning a value is done by the assignment to ret.
- Procedures do not access any global vars.

```
let alloc2() =
  local t1, t2.
  t1=new(1); *t1=0;
  t2=new(1); *t2=t1;
  ret=t2
in
  x=alloc2();
  y=alloc2()
```



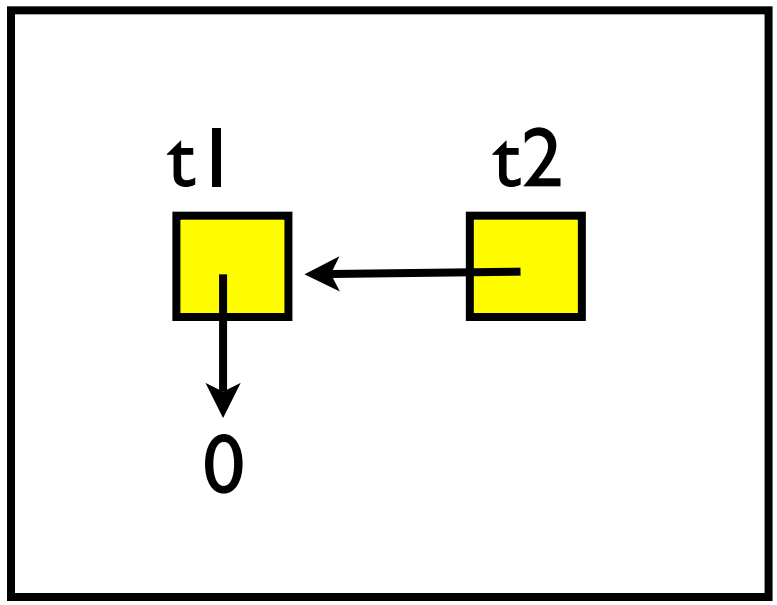
Language with Simple Procedures

ret $\in$ Vars

C ::= ... | `local` $x.C$ | $x{=}f(\vec{E})$ | `let` $f(\vec{x}){=}C$ `in` C

- Call-by-value procedures as in Java.
- Returning a value is done by the assignment to ret.
- Procedures do not access any global vars.

```
let alloc2() =
  local t1, t2.
  t1=new(1); *t1=0;
  t2=new(1); *t2=t1;
  ret=t2
in
  x=alloc2();
  y=alloc2()
```



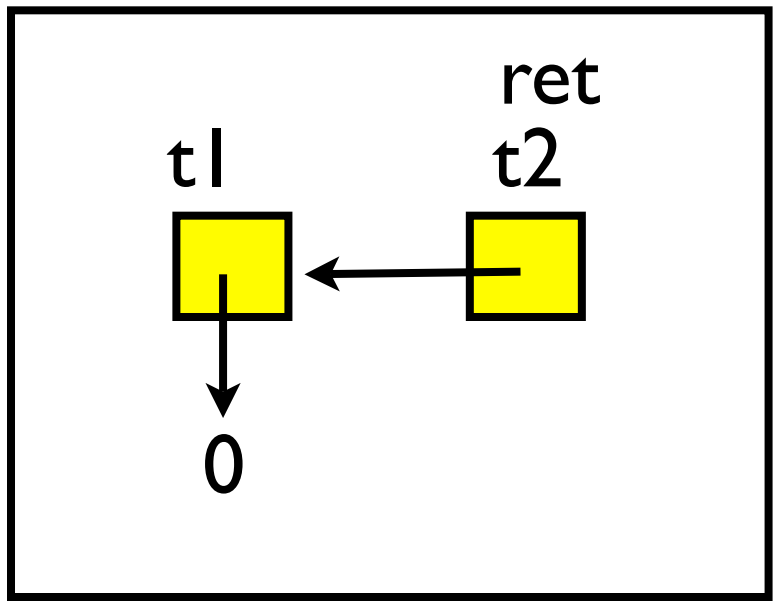
Language with Simple Procedures

ret $\in$ Vars

C ::= ... | `local` $x.C$ | $x{=}f(\vec{E})$ | `let` $f(\vec{x}){=}C$ `in` C

- Call-by-value procedures as in Java.
- Returning a value is done by the assignment to ret.
- Procedures do not access any global vars.

```
let alloc2() =
  local t1, t2.
  t1=new(1); *t1=0;
  t2=new(1); *t2=t1;
  ret=t2
in
  x=alloc2();
  y=alloc2()
```



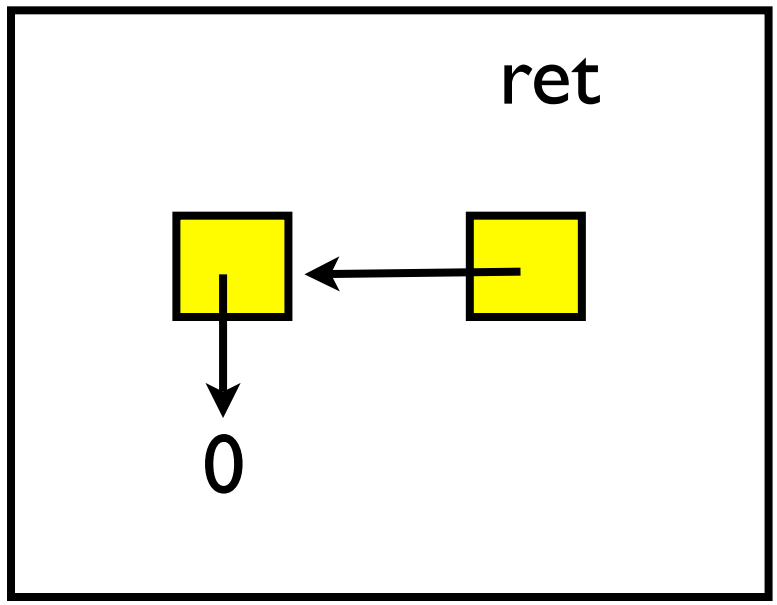
Language with Simple Procedures

ret $\in$ Vars

$C ::= \ldots \mid \texttt{local}\ x.C \mid x{=}f(\vec{E}) \mid \texttt{let}\ f(\vec{x}){=}C\ \texttt{in}\ C$

- Call-by-value procedures as in Java.
- Returning a value is done by the assignment to ret.
- Procedures do not access any global vars.

```
let alloc2() =
  local t1, t2.
  t1=new(1); *t1=0;
  t2=new(1); *t2=t1;
  ret=t2
in
  x=alloc2();
  y=alloc2()
```



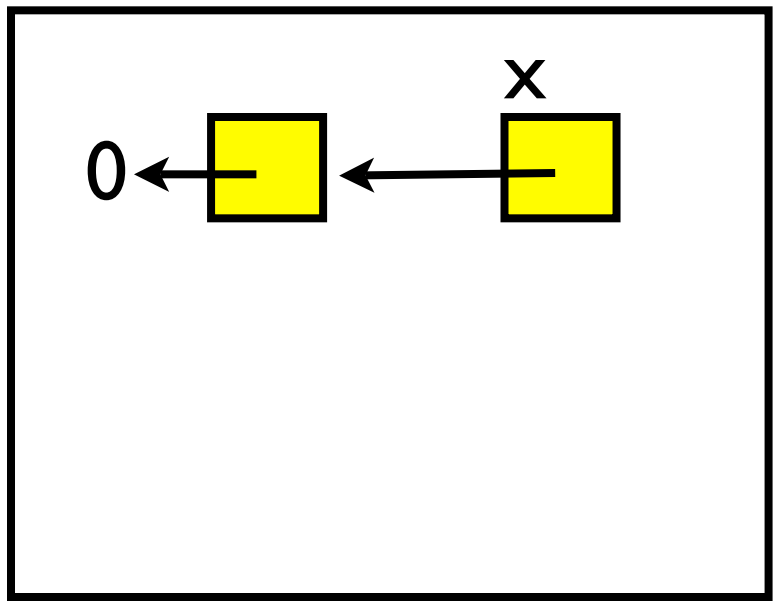
Language with Simple Procedures

ret $\in$ Vars

C ::= ... | `local` $x.C$ | $x{=}f(\vec{E})$ | `let` $f(\vec{x}){=}C$ `in` C

- Call-by-value procedures as in Java.
- Returning a value is done by the assignment to ret.
- Procedures do not access any global vars.

```
let alloc2() =
   local t1, t2.
   t1=new(1); *t1=0;
   t2=new(1); *t2=t1;
   ret=t2
in
   x=alloc2();
   y=alloc2()
```



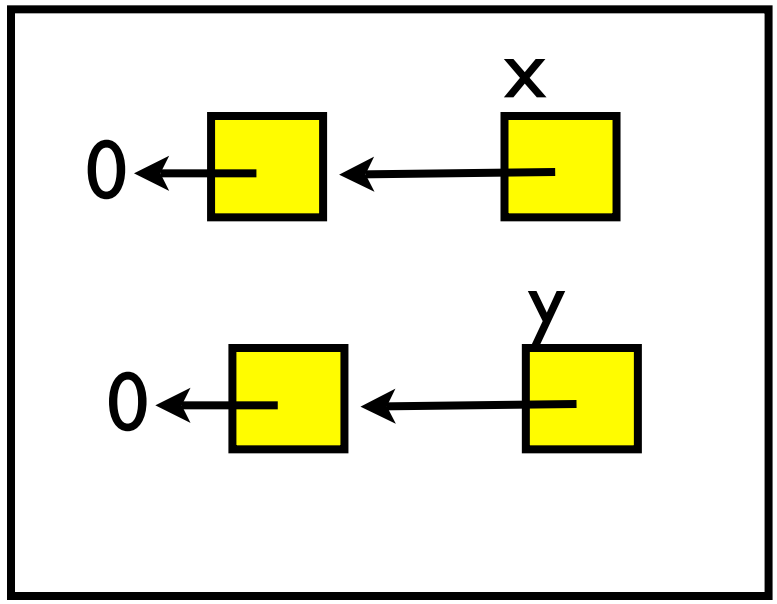
Language with Simple Procedures

ret $\in$ Vars

C ::= ... | `local` $x.C$ | $x{=}f(\vec{E})$ | `let` $f(\vec{x}){=}C$ `in` C

- Call-by-value procedures as in Java.
- Returning a value is done by the assignment to ret.
- Procedures do not access any global vars.

```
let alloc2() =
  local t1, t2.
  t1=new(1); *t1=0;
  t2=new(1); *t2=t1;
  ret=t2
in
  x=alloc2();
  y=alloc2()
```



Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

{emp}
let alloc2() =

  local t_1, t_2.

  t_1=new(1); $*t_1$=0;

  t_2=new(1); $*t_2$=t_1;

  ret=t_2

in

  x=alloc2();

  y=alloc2()

$\{\exists x'y'.\, x{\mapsto}x' * x'{\mapsto}0 * y{\mapsto}y' * y'{\mapsto}0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

```
{emp}
let alloc2() =
  {emp}
    local t1, t2.

    t1=new(1); *t1=0;

    t2=new(1); *t2=t1;

    ret=t2


  {∃t1. ret↦t1 * t1↦0}
in

    x=alloc2();

    y=alloc2()
{∃x'y'. x↦x' * x'↦0 * y↦y' * y'↦0}
```

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

```
{emp}
let alloc2() =
  {emp}
   local t1, t2.
  {emp}
   t1=new(1); *t1=0;

   t2=new(1); *t2=t1;

   ret=t2


  {∃t1. ret↦t1 * t1↦0}
in

   x=alloc2();

   y=alloc2()
{∃x'y'. x↦x' * x'↦0 * y↦y' * y'↦0}
```

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

$\{\mathsf{emp}\}$
$\texttt{let alloc2}() =$
  $\{\mathsf{emp}\}$
  $\texttt{local}\; t_1, t_2.$
  $\{\mathsf{emp}\}$
  $t_1{=}\texttt{new}(1);\; {*}t_1{=}0;$
  $\{t_1 \mapsto 0\}$
  $t_2{=}\texttt{new}(1);\; {*}t_2{=}t_1;$

  $\mathsf{ret}{=}t_2$

  $\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\texttt{in}$

  $x{=}\texttt{alloc2}();$

  $y{=}\texttt{alloc2}()$
$\{\exists x'y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

$\{\mathsf{emp}\}$
$\texttt{let alloc2}() =$
  $\{\mathsf{emp}\}$
  $\texttt{local}\; t_1, t_2.$
  $\{\mathsf{emp}\}$
  $t_1{=}\texttt{new}(1);\; {*}t_1{=}0;$
  $\{t_1 \mapsto 0\}$
  $t_2{=}\texttt{new}(1);\; {*}t_2{=}t_1;$
  $\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\mathsf{ret}{=}t_2$

  $\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\texttt{in}$

  $x{=}\texttt{alloc2}();$

  $y{=}\texttt{alloc2}()$
$\{\exists x'y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

$\{\mathsf{emp}\}$
$\texttt{let alloc2}() =$
  $\{\mathsf{emp}\}$
  $\texttt{local}\; t_1, t_2.$
  $\{\mathsf{emp}\}$
  $t_1{=}\texttt{new}(1);\; {*}t_1{=}0;$
  $\{t_1 \mapsto 0\}$
  $t_2{=}\texttt{new}(1);\; {*}t_2{=}t_1;$
  $\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\mathsf{ret}{=}t_2$
  $\{\mathsf{ret}{=}t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$

  $\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\texttt{in}$

  $x{=}\texttt{alloc2}();$

  $y{=}\texttt{alloc2}()$
$\{\exists x'y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

$\{\mathsf{emp}\}$
$\texttt{let alloc2}() =$
  $\{\mathsf{emp}\}$
  $\texttt{local}\; t_1, t_2.$
  $\{\mathsf{emp}\}$
  $t_1{=}\texttt{new}(1);\; {*}t_1{=}0;$
  $\{t_1 \mapsto 0\}$
  $t_2{=}\texttt{new}(1);\; {*}t_2{=}t_1;$
  $\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\mathsf{ret}{=}t_2$
  $\{\mathsf{ret}{=}t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\mathsf{ret}{=}t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\texttt{in}$

  $x{=}\texttt{alloc2}();$

  $y{=}\texttt{alloc2}()$
$\{\exists x'y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q)$$

$$\frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

$\{\mathsf{emp}\}$
$\texttt{let alloc2}() =$
$\{\mathsf{emp}\}$
$\texttt{local}\; t_1, t_2.$
$\{\mathsf{emp}\}$
$t_1{=}\texttt{new}(1);\; {*}t_1{=}0;$
$\{t_1 \mapsto 0\}$
$t_2{=}\texttt{new}(1);\; {*}t_2{=}t_1;$
$\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
$\mathsf{ret}{=}t_2$
$\{\mathsf{ret}{=}t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
$\{\mathsf{ret}{=}t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\texttt{in}$

$x{=}\texttt{alloc2}();$

$y{=}\texttt{alloc2}()$
$\{\exists x'y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

$\{\mathsf{emp}\}$
$\texttt{let alloc2}() =$
  $\{\mathsf{emp}\}$
  $\texttt{local}\; t_1, t_2.$
  $\{\mathsf{emp}\}$
  $t_1{=}\texttt{new}(1);\; {*}t_1{=}0;$
  $\{t_1 \mapsto 0\}$
  $t_2{=}\texttt{new}(1);\; {*}t_2{=}t_1;$
  $\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\mathsf{ret}{=}t_2$
  $\{\mathsf{ret}{=}t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\mathsf{ret}{=}t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\texttt{in}$

  $x{=}\texttt{alloc2}();$

  $y{=}\texttt{alloc2}()$
$\{\exists x'y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

$\{\mathsf{emp}\}$
$\texttt{let alloc2}() =$
  $\{\mathsf{emp}\}$
  $\texttt{local}\; t_1, t_2.$
  $\{\mathsf{emp}\}$
  $t_1{=}\texttt{new}(1);\; {*}t_1{=}0;$
  $\{t_1 \mapsto 0\}$
  $t_2{=}\texttt{new}(1);\; {*}t_2{=}t_1;$
  $\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\mathsf{ret}{=}t_2$
  $\{\mathsf{ret}{=}t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\mathsf{ret}{=}t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\texttt{in}$
$\{\mathsf{emp}\}$
  $x{=}\texttt{alloc2}();$

  $y{=}\texttt{alloc2}()$
$\{\exists x'y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

$\{\mathsf{emp}\}$
$\texttt{let alloc2}() =$
  $\{\mathsf{emp}\}$
  $\texttt{local}\; t_1, t_2.$
  $\{\mathsf{emp}\}$
  $t_1{=}\texttt{new}(1);\; {*}t_1{=}0;$
  $\{t_1 \mapsto 0\}$
  $t_2{=}\texttt{new}(1);\; {*}t_2{=}t_1;$
  $\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\mathsf{ret}{=}t_2$
  $\{\mathsf{ret}{=}t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\mathsf{ret}{=}t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\texttt{in}$
$\{\mathsf{emp}\}$
  $x{=}\texttt{alloc2}();$
$\{\exists x'.\, x \mapsto x' * x' \mapsto 0\}$
  $y{=}\texttt{alloc2}()$
$\{\exists x' y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q)$$

$$\frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

Cannot apply the rule!

$\{\mathsf{emp}\}$
$\texttt{let alloc2}() =$
  $\{\mathsf{emp}\}$
  $\texttt{local}\; t_1, t_2.$
  $\{\mathsf{emp}\}$
  $t_1{=}\texttt{new}(1);\; {*}t_1{=}0;$
  $\{t_1 \mapsto 0\}$
  $t_2{=}\texttt{new}(1);\; {*}t_2{=}t_1;$
  $\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\mathsf{ret}{=}t_2$
  $\{\mathsf{ret}{=}t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\mathsf{ret}{=}t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\texttt{in}$
$\{\mathsf{emp}\}$
  $x{=}\texttt{alloc2}();$
$\{\exists x'.\, x \mapsto x' * x' \mapsto 0\}$
  $y{=}\texttt{alloc2}()$
$\{\exists x'y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\, x.C\{Q\}}\; x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\}, \ldots, \{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\; f(x){=}C\; \texttt{in}\; D\{Q\}}\; x \notin \mathsf{FV}(Q_i)$$

- E.g.

$\{\mathsf{emp}\}$
$\texttt{let alloc2}() =$
  $\{\mathsf{emp}\}$
  $\texttt{local}\; t_1, t_2.$
  $\{\mathsf{emp}\}$
  $t_1{=}\texttt{new}(1);\; {*}t_1{=}0;$
  $\{t_1 \mapsto 0\}$
  $t_2{=}\texttt{new}(1);\; {*}t_2{=}t_1;$
  $\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\mathsf{ret}{=}t_2$
  $\{\mathsf{ret}{=}t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\mathsf{ret}{=}t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
  $\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\texttt{in}$
$\{\mathsf{emp}\}$
  $x{=}\texttt{alloc2}();$
$\{\exists x'.\, x \mapsto x' * x' \mapsto 0\}$
  $y{=}\texttt{alloc2}()$
$\{\exists x' y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

$\{\exists x'.\, x \mapsto x' * x' \mapsto 0\}$

$\{\exists x' t_1.\, x \mapsto x' * x' \mapsto 0 * \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$

Proof Rules for Local Vars. and Proc.

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P\}\texttt{local}\ x.C\{Q\}}\ x \notin \mathsf{FV}(P,Q) \qquad \frac{}{\Gamma, \{P\}f(x)\{Q\} \vdash \{P[E/x]\}y{=}f(E)\{Q[y/\mathsf{ret}]\}}$$

$$\frac{\Gamma \vdash \{P_i\}C\{Q_i\} \quad \Gamma, \{P_0\}f(x)\{Q_0\},\ldots,\{P_n\}f(x)\{Q_n\} \vdash \{P\}D\{Q\}}{\Gamma \vdash \{P\}\texttt{let}\ f(x){=}C\ \texttt{in}\ D\{Q\}}\ x \notin \mathsf{FV}(Q_i)$$

- E.g.

```
{emp}
let alloc2() =
  {emp}                                   {∃x'. x↦x' ∗ x'↦0}
  local t1, t2.
  {emp}
  t1=new(1); ∗t1=0;
  {t1↦0}
  t2=new(1); ∗t2=t1;
  {t2↦t1 ∗ t1↦0}
  ret=t2
  {ret=t2 ∧ t2↦t1 ∗ t1↦0}
  {ret=t2 ∧ ret↦t1 ∗ t1↦0}
  {∃t1. ret↦t1 ∗ t1↦0}                    {∃x't1. x↦x' ∗ x'↦0 ∗ ret↦t1 ∗ t1↦0}
in
{emp}
  x=alloc2();
{∃x'. x↦x' ∗ x'↦0}
  y=alloc2()
{∃x'y'. x↦x' ∗ x'↦0 ∗ y↦y' ∗ y'↦0}
```

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \ \mathsf{Modifies}(C) \cap \mathsf{FV}(R) = \emptyset$$

- Means that R can be added as an invariant.
- * and err-avoiding triple take care of the heap access of C, and the side cond. takes care of the stack access.

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \quad \mathsf{Modifies}(C) \cap \mathsf{FV}(R) = \emptyset$$

- Means that R can be added as an invariant
- * and e ccess of C, an access.

$\{\mathsf{emp}\}$
$\mathtt{let}\ \mathtt{alloc2}() =$
$\{\mathsf{emp}\}$
$\mathtt{local}\ t_1, t_2.$
$\{\mathsf{emp}\}$
$t_1 = \mathtt{new}(1);\ *t_1 = 0;$
$\{t_1 \mapsto 0\}$
$t_2 = \mathtt{new}(1);\ *t_2 = t_1;$
$\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
$\mathsf{ret} = t_2$
$\{\mathsf{ret} = t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
$\{\mathsf{ret} = t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\{\exists t_1.\ \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\mathtt{in}$
$\{\mathsf{emp}\}$
$x = \mathtt{alloc2}();$
$\{\exists x'.\ x \mapsto x' * x' \mapsto 0\}$
$y = \mathtt{alloc2}()$
$\{\exists x' y'.\ x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \quad \mathsf{Modifies}(C) \cap \mathsf{FV}(R) = \emptyset$$

- Means that R can be added as an invariant
- * and e… ccess of C, an… access.

{emp}
let alloc2() =
{emp}
local t_1, t_2.
{emp}
t_1=new(1); $*t_1$=0;
$\{t_1 \mapsto 0\}$
t_2=new(1); $*t_2$=t_1;
$\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
ret=t_2
$\{\mathsf{ret}=t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
$\{\mathsf{ret}=t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\{\exists t_1.\, \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
in
{emp}
x=alloc2();
$\{\exists x'.\, x \mapsto x' * x' \mapsto 0\}$
y=alloc2()
$\{\exists x' y'.\, x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

{emp}
y=alloc2()
$\{\exists y'.\, y \mapsto y' * y' \mapsto 0\}$

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \quad \mathsf{Modifies}(C) \cap \mathsf{FV}(R) = \emptyset$$

- Means that R can be added as an invariant
- * and e… ccess of C, an… access.

$\{\mathsf{emp}\}$
$\mathtt{let}\ \mathtt{alloc2}() =$
$\{\mathsf{emp}\}$
$\mathtt{local}\ t_1, t_2.$
$\{\mathsf{emp}\}$
$t_1 = \mathtt{new}(1);\ *t_1 = 0;$
$\{t_1 \mapsto 0\}$
$t_2 = \mathtt{new}(1);\ *t_2 = t_1;$
$\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
$\mathsf{ret} = t_2$
$\{\mathsf{ret} = t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
$\{\mathsf{ret} = t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\{\exists t_1.\ \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\mathtt{in}$
$\{\mathsf{emp}\}$
$x = \mathtt{alloc2}();$
$\{\exists x'.\ x \mapsto x' * x' \mapsto 0\}$
$y = \mathtt{alloc2}()$
$\{\exists x' y'.\ x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

$\{\mathsf{emp} * (\exists x'.\ x \mapsto x' * x' \mapsto 0)\}$
$\{\mathsf{emp}\}$
$y = \mathtt{alloc2}()$
$\{\exists y'.\ y \mapsto y' * y' \mapsto 0\}$
$\{(\exists y'.\ y \mapsto y' * y' \mapsto 0) * (\exists x'.\ x \mapsto x' * x' \mapsto 0)\}$

Frame Rule

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \quad \mathsf{Modifies}(C) \cap \mathsf{FV}(R) = \emptyset$$

- Means that R can be added as an invariant
- * and e… ccess of C, an… access.

$\{\mathsf{emp}\}$
$\mathtt{let}\ \mathtt{alloc2}() =$
$\{\mathsf{emp}\}$
$\mathtt{local}\ t_1, t_2.$
$\{\mathsf{emp}\}$
$t_1 = \mathtt{new}(1);\ *t_1 = 0;$
$\{t_1 \mapsto 0\}$
$t_2 = \mathtt{new}(1);\ *t_2 = t_1;$
$\{t_2 \mapsto t_1 * t_1 \mapsto 0\}$
$\mathsf{ret} = t_2$
$\{\mathsf{ret} = t_2 \wedge t_2 \mapsto t_1 * t_1 \mapsto 0\}$
$\{\mathsf{ret} = t_2 \wedge \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\{\exists t_1.\ \mathsf{ret} \mapsto t_1 * t_1 \mapsto 0\}$
$\mathtt{in}$
$\{\mathsf{emp}\}$
$x = \mathtt{alloc2}();$
$\{\exists x'.\ x \mapsto x' * x' \mapsto 0\}$
$y = \mathtt{alloc2}()$
$\{\exists x' y'.\ x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

$\{\exists x'.\ x \mapsto x' * x' \mapsto 0\}$
$\{\mathsf{emp} * (\exists x'.\ x \mapsto x' * x' \mapsto 0)\}$
$\{\mathsf{emp}\}$
$y = \mathtt{alloc2}()$
$\{\exists y'.\ y \mapsto y' * y' \mapsto 0\}$
$\{(\exists y'.\ y \mapsto y' * y' \mapsto 0) * (\exists x'.\ x \mapsto x' * x' \mapsto 0)\}$
$\{\exists x' y'.\ x \mapsto x' * x' \mapsto 0 * y \mapsto y' * y' \mapsto 0\}$

Frame Rule and Local Reasoning

$$\frac{\{P\}C\{Q\}}{\{P * R\}C\{Q * R\}} \ \mathsf{Modifies}(C) \cap \mathsf{FV}(R) = \emptyset$$

- Local reasoning means that the verification of a prog. fragment should focus on what's accessed by the frag.
- The frame rule supports local reasoning.
- This support of local reasoning is the reason that sep. logic has been successful.

Example Again

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

- Verification of the caller program:

{emp}

```
w=create_list();
```

{ls(w,0)}

```
x=create_list();

y=create_list();

z=create_list()
```

Example Again

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

- Verification of the caller program:

{emp}

```
w=create_list();
```

{ls(w,0)}

```
x=create_list();
```

{ls (w,0) * ls (x,0)}

```
y=create_list();


z=create_list()
```

Example Again

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

- Verification of the caller program:

{emp}

```
w=create_list();
```

{ls(w,0)}

```
x=create_list();
```

{ls (w,0) * ls (x,0)}

```
y=create_list();
```

{ls (w,0) * ls (x,0) * ls(y,0)}

```
z=create_list()
```

Example Again

- Spec for create_list:

{emp}

```
create_list()
```

{ls(ret,0)}

- Verification of the caller program:

{emp}

```
w=create_list();
```

{ls(w,0)}

```
x=create_list();
```

{ls (w,0) * ls (x,0)}

```
y=create_list();
```

{ls (w,0) * ls (x,0) * ls(y,0)}

```
z=create_list()
```

{ls (w,0) * ls (x,0) * ls(y,0) * ls(z,0)}

Frame Rule and Interprocedural Analysis

Review: Analysis Algorithm

- Run a given program with symbolic heaps.
- Collect all the obtained sym. heaps at each program point.
- Repeat until we cannot get any new sym. heaps.

Interprocedural Analysis

- For each procedure, create a table that records all the past analysis results.
- Table for create_list():

Precond.	Postcondition
emp	ret=0 $\wedge$ emp, ret$\mapsto$0, ls ret 0
...	...

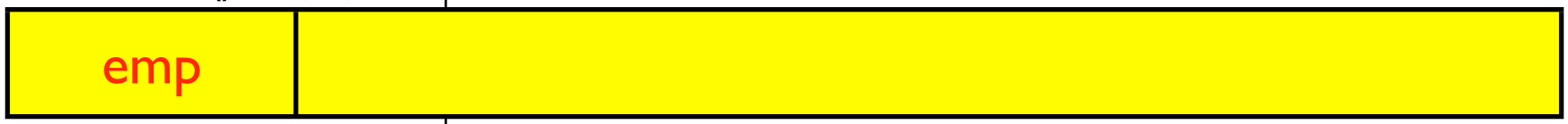
- Use the table whenever possible.
- Assume non-recursive procedures in this lecture.

Creation of 2 Lists

```
let create_list() ={...}




in

        emp
 x=create_list();


 y=create_list();
```

Creation of 2 Lists

let create_list() ={...}



in

emp

x=create_list();

y=create_list();

Creation of 2 Lists

let create_list() ={...}

emp	ret=0 ∧ emp,	ret↦0,	ls(ret,0)

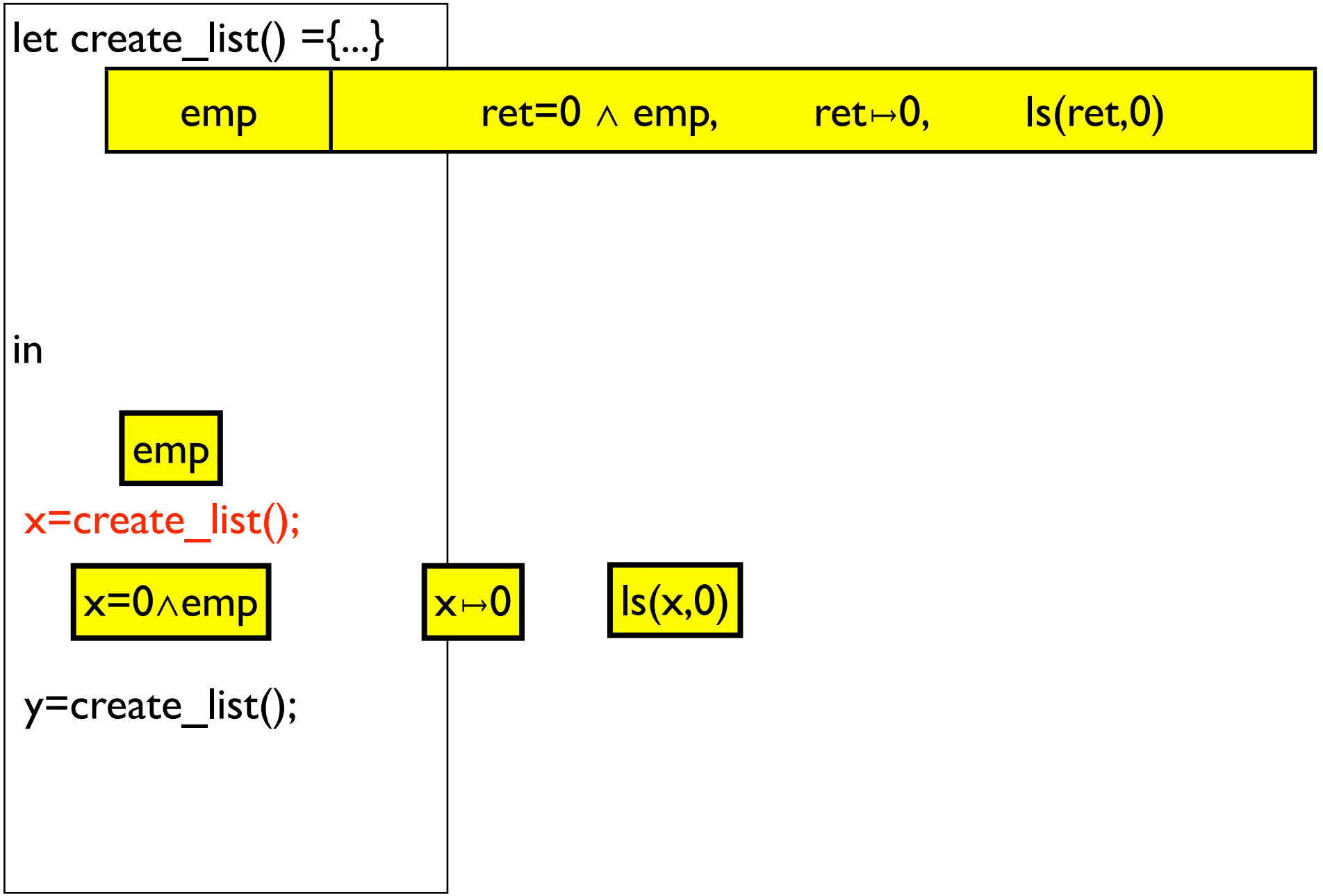
in

emp

x=create_list();

y=create_list();

Creation of 2 Lists



Creation of 2 Lists

let create_list() ={...}

emp	ret=0 ∧ emp, ret↦0, ls(ret,0)
x↦0	
ls x 0	

in

emp

x=create_list();

x=0∧emp x↦0 ls(x,0)

y=create_list();

Creation of 2 Lists

let create_list() ={...}

emp	ret=0 ∧ emp, ret↦0, ls(ret,0)
x↦0	ret=0 ∧ x↦0, x↦0 * ret↦0, x↦0 * ls(ret,0)
ls x 0	ret=0 ∧ ls(x,0), ls(x,0) * ret↦0, ls(x,0) * ls(ret,0)

in

emp

x=create_list();

x=0∧emp x↦0 ls(x,0)

y=create_list();

Creation of 2 Lists

let create_list() ={...}

emp	ret=0 ∧ emp, ret↦0, ls(ret,0)
x↦0	ret=0 ∧ x↦0, x↦0 * ret↦0, x↦0 * ls(ret,0)
ls x 0	ret=0 ∧ ls(x,0), ls(x,0) * ret↦0, ls(x,0) * ls(ret,0)

in

emp

x=create_list();

x=0∧emp x↦0 ls(x,0)

y=create_list();

x=0∧y=0∧emp x=0∧y↦0 ..... x↦0*y↦0 ..... ls(x,0)*ls(y,0)

Creation of 2 Lists

let create_list() ={...}

emp	ret=0 ∧ emp, ret↦0, ls(ret,0)
x↦0	ret=0 ∧ x↦0, x↦0 * ret↦0, x↦0 * ls(ret,0)
ls x 0	ret=0 ∧ ls(x,0), ls(x,0) * ret↦0, ls(x,0) * ls(ret,0)

3 entries, 9 results

in

emp

x=create_list();

x=0∧emp x↦0 ls(x,0)

y=create_list();

x=0∧y=0∧emp x=0∧y↦0 ..... x↦0*y↦0 ..... ls(x,0)*ls(y,0)

Creation of 2 Lists

let create_list() ={...}

emp	ret=0 ∧ emp, ret↦0, ls(ret,0)
x↦0	ret=0 ∧ x↦0, x↦0 * ret↦0, x↦0 * ls(ret,0)
ls x 0	ret=0 ∧ ls(x,0), ls(x,0) * ret↦0, ls(x,0) * ls(ret,0)

3 entries, 9 results

in

emp

The verifier constructs proofs of two Hoare triples for create_list unnecessarily.

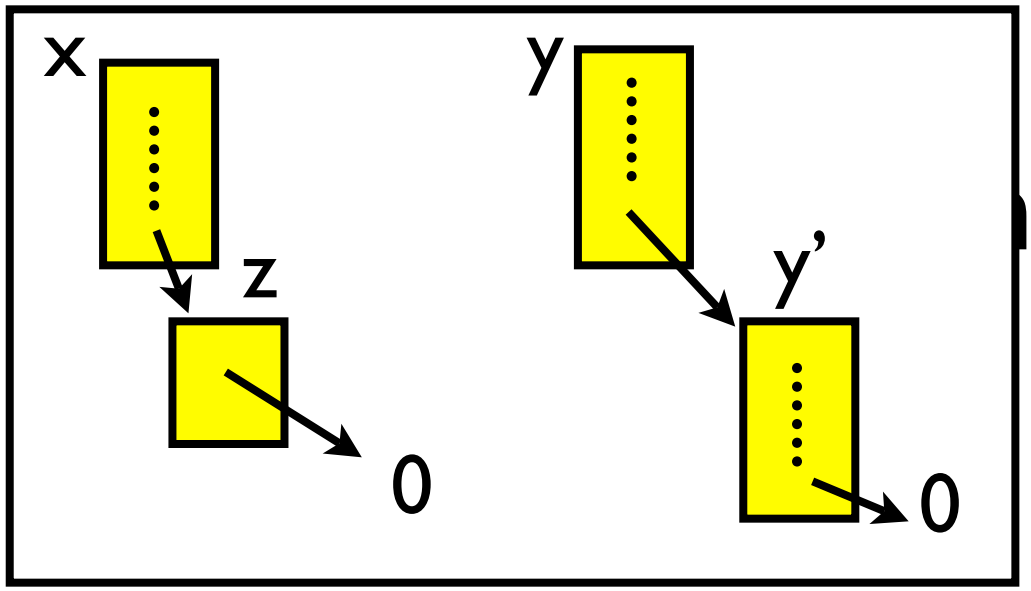
y=create_list();

x=0∧y=0∧emp x=0∧y↦0 ..... x↦0*y↦0 ..... ls(x,0)*ls(y,0)

Optimization by the Frame Rule

- Pass & change only the part of a symbolic heap, that is reachable from the parameters. [Rinetzky et al, O'Hearn et al, Gotsman et al]
- E.g.

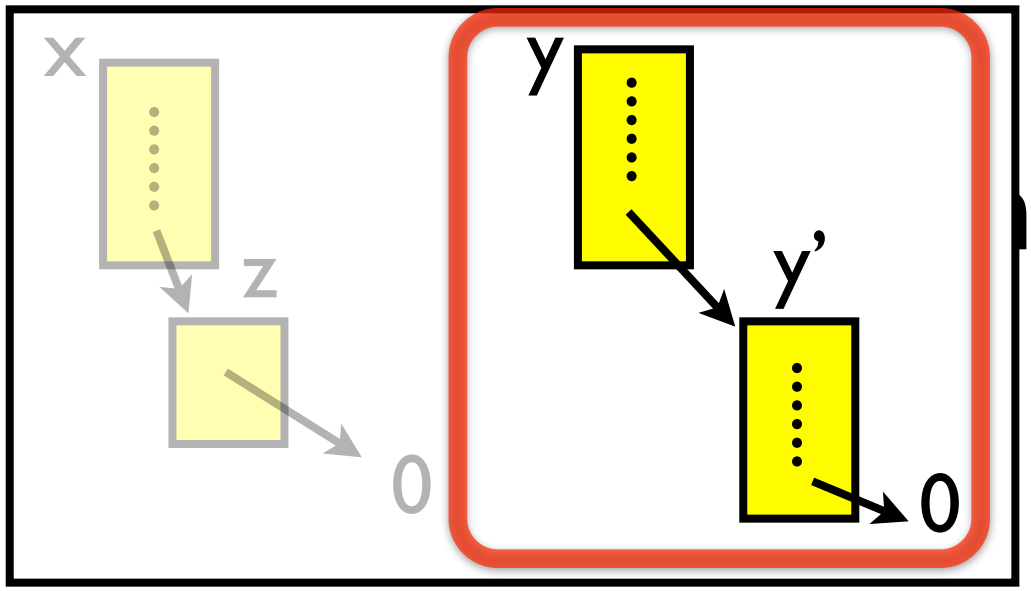
  $\{\exists y'.\ ls(x,z) * z \mapsto 0 * ls(y,y') * ls(y',0)\ \}$
  dispose_list(y)



e Frame Rule

- Pass & change only the part of a symbolic heap, that is reachable from the parameters. [Rinetzky et al, O'Hearn et al, Gotsman et al]
- E.g.

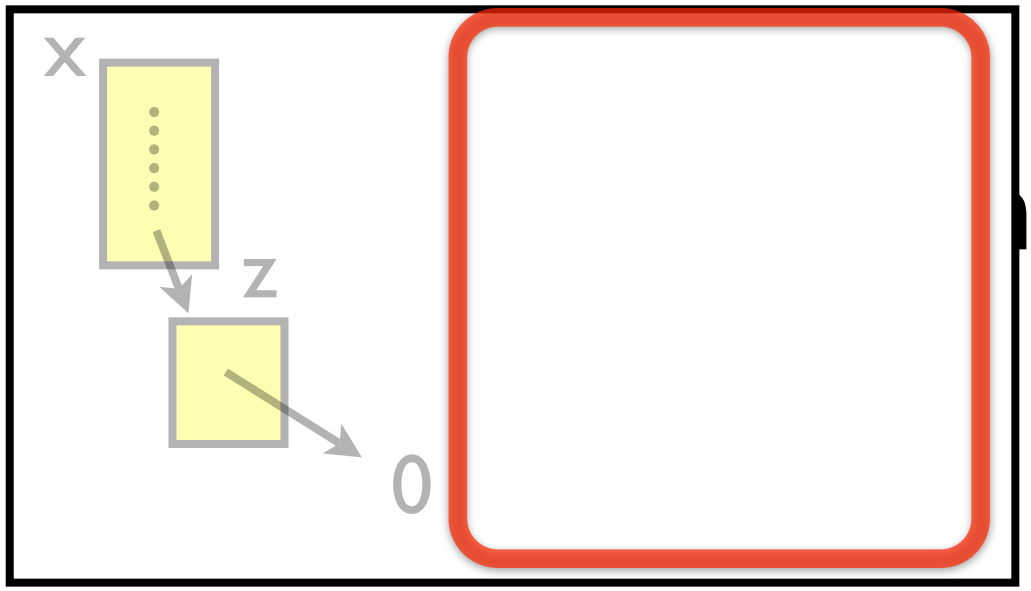
  {∃y'. ls(x,z) * z↦0 * ls(y,y') * ls(y',0) }
  dispose_list(y)



e Frame Rule

- Pass & change only the part of a symbolic heap, that is reachable from the parameters. [Rinetzky et al, O'Hearn et al, Gotsman et al]
- E.g.

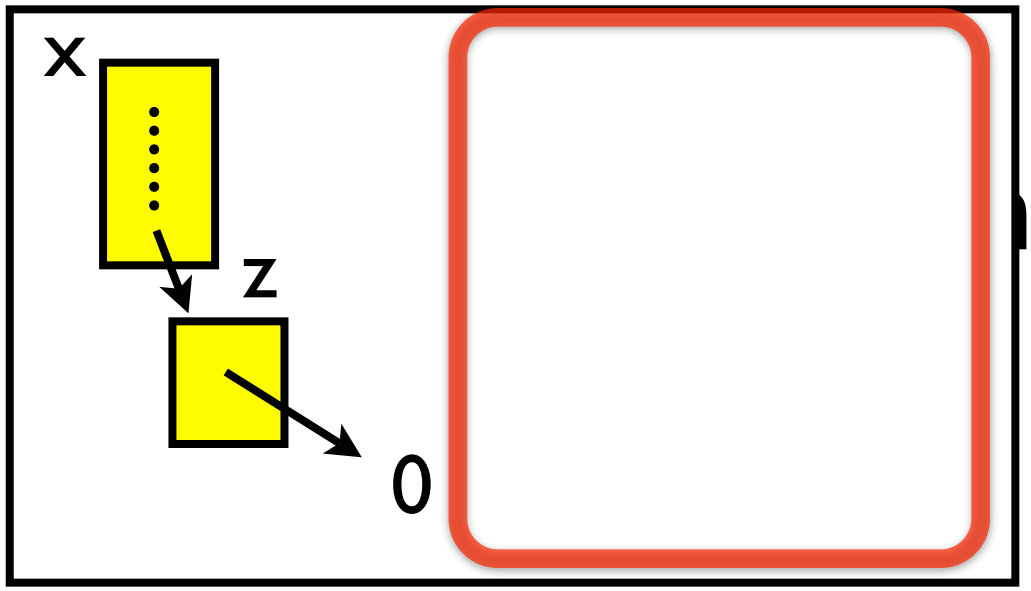
  {∃y'. ls(x,z) * z↦0 * ls(y,y') * ls(y',0) }
  dispose_list(y)



e Frame Rule

- Pass & change only the part of a symbolic heap, that is reachable from the parameters. [Rinetzky et al, O'Hearn et al, Gotsman et al]
- E.g.

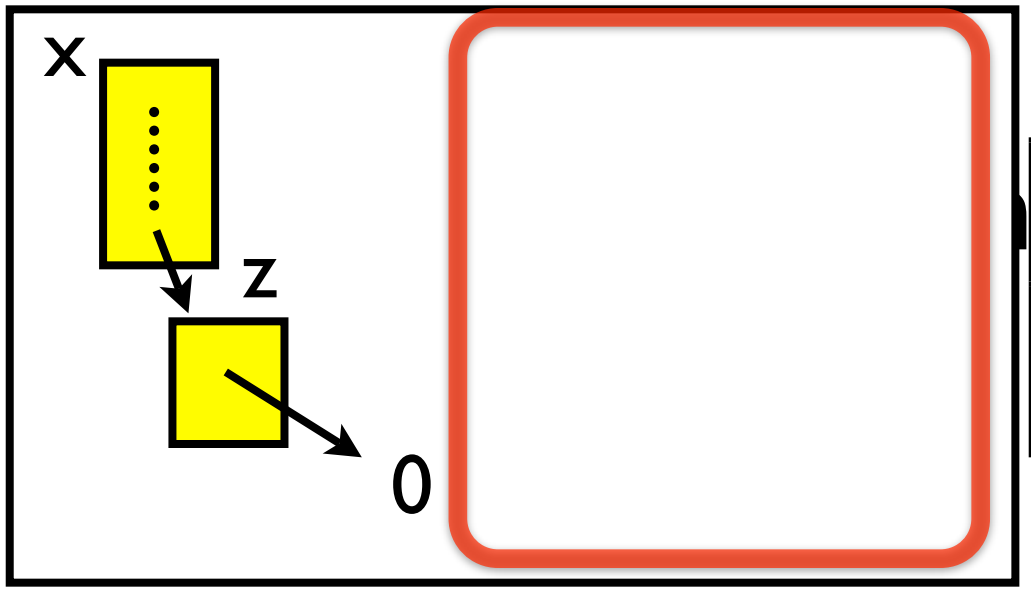
  {∃y'. ls(x,z) * z↦0 * ls(y,y') * ls(y',0) }
  dispose_list(y)



e Frame Rule

- Pass & change only the part of a symbolic heap, that is reachable from the parameters. [Rinetzky et al, O'Hearn et al, Gotsman et al]
- E.g.

  {∃y'. ls(x,z) * z↦0 * ls(y,y') * ls(y',0) }
  dispose_list(y)



Pre	Post
∃y'.ls(y, y') * ls(y',0)	emp

- Pass & change only the part of a symbolic heap, that is reachable from the parameters [Rinetzky et al.
- E

Uses Frame Rule and Proc. Call

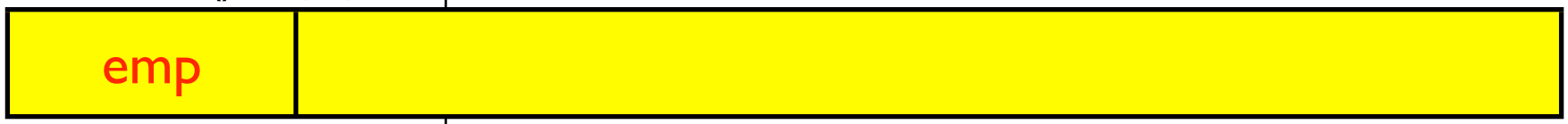
$$\frac{\{P\}f()\{Q\} \vdash \{P\}x{=}f()\{Q[x/\mathsf{ret}]\}}{\{P\}f()\{Q\} \vdash \{P * R\}x{=}f()\{Q[x/\mathsf{ret}] * R\}}\ \begin{matrix}\text{Call}\\ \text{Frame}\end{matrix}$$

Creation of 2 Lists

```
let create_list() ={...}




in

        emp
 x=create_list();


 y=create_list();
```

Creation of 2 Lists

let create_list() ={...}



in

emp

x=create_list();

y=create_list();

Creation of 2 Lists

let create_list() ={...}

emp	ret=0 ∧ emp,	ret↦0,	ls(ret,0)

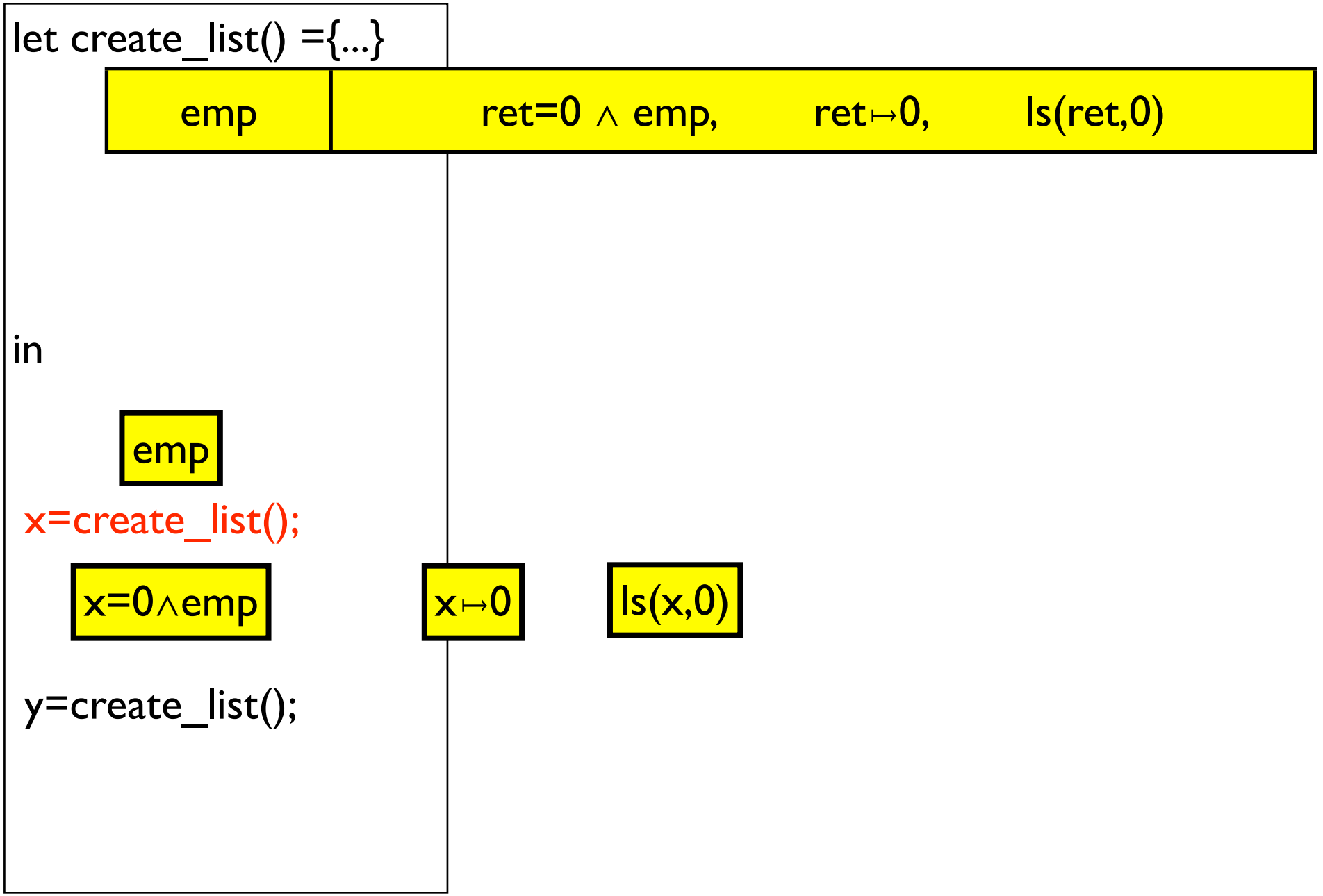
in

emp

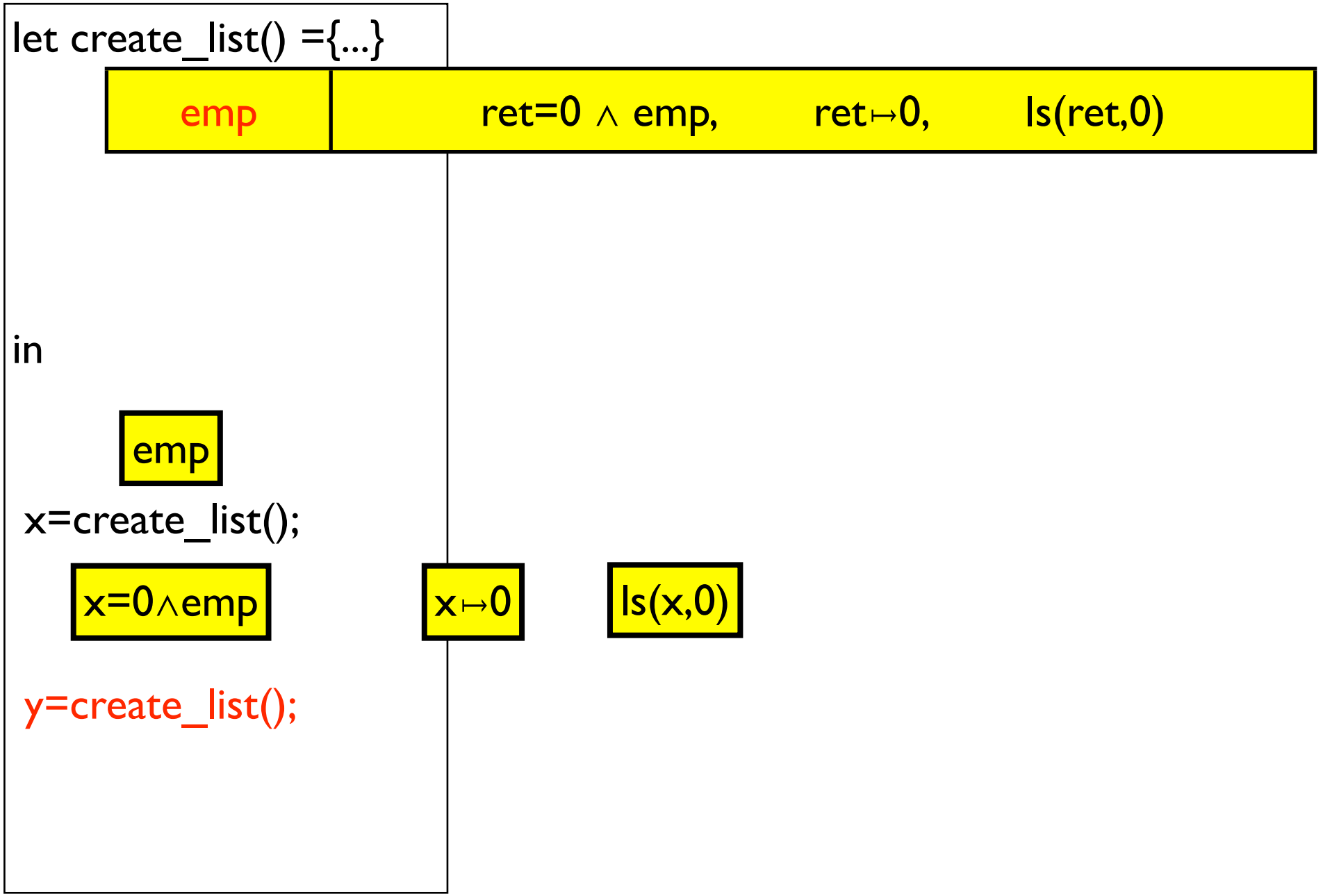
x=create_list();

y=create_list();

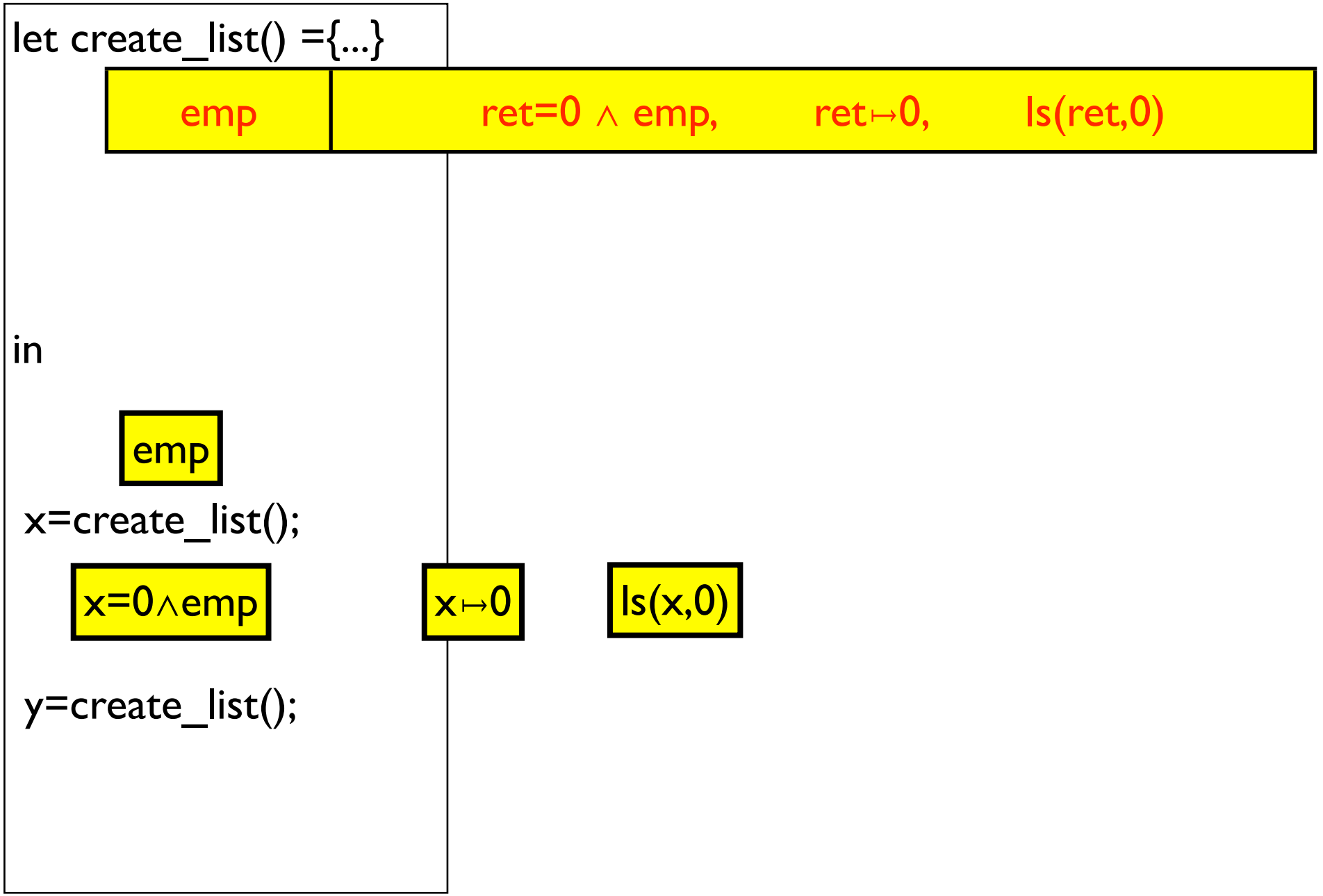
Creation of 2 Lists



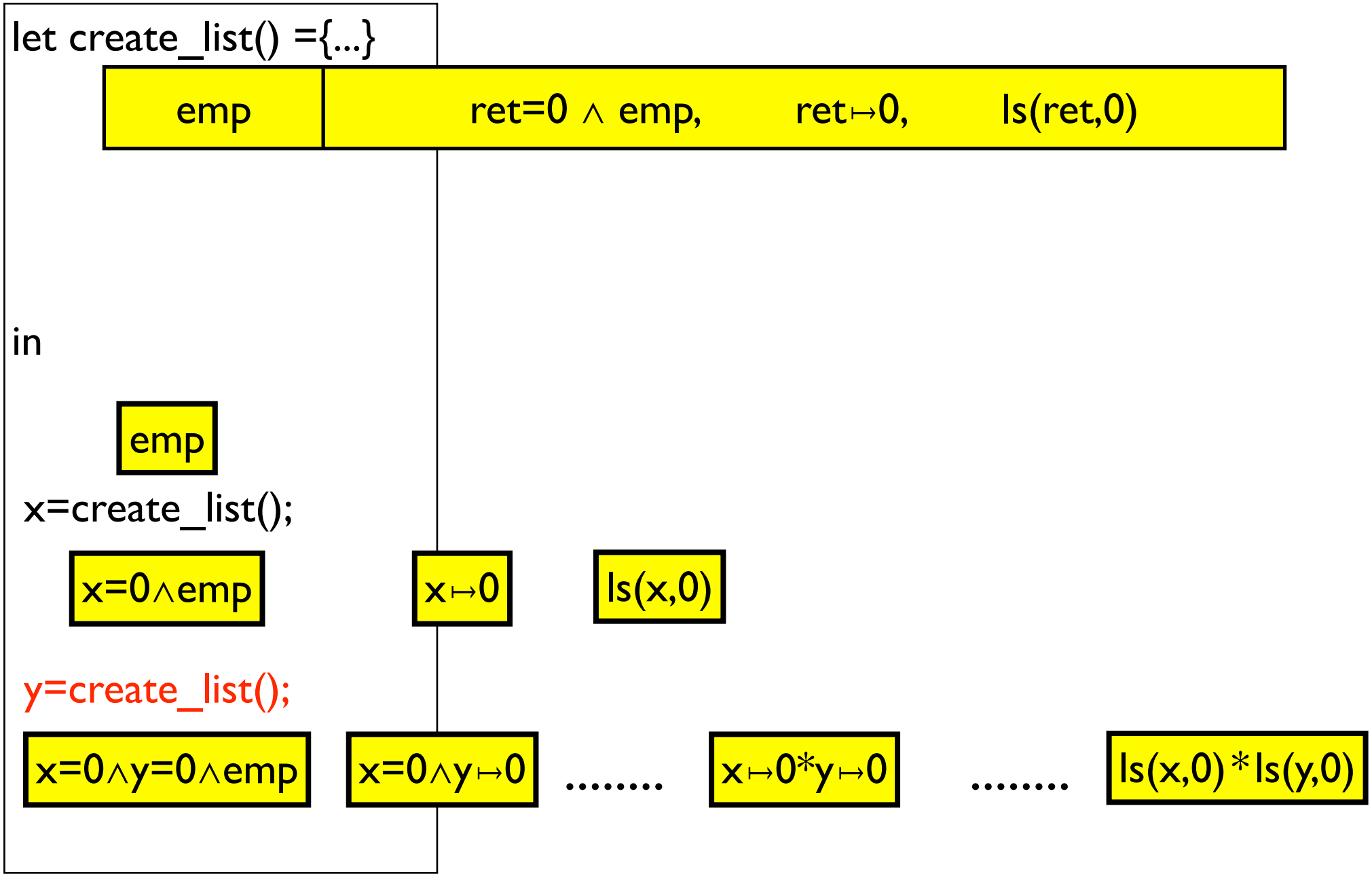
Creation of 2 Lists



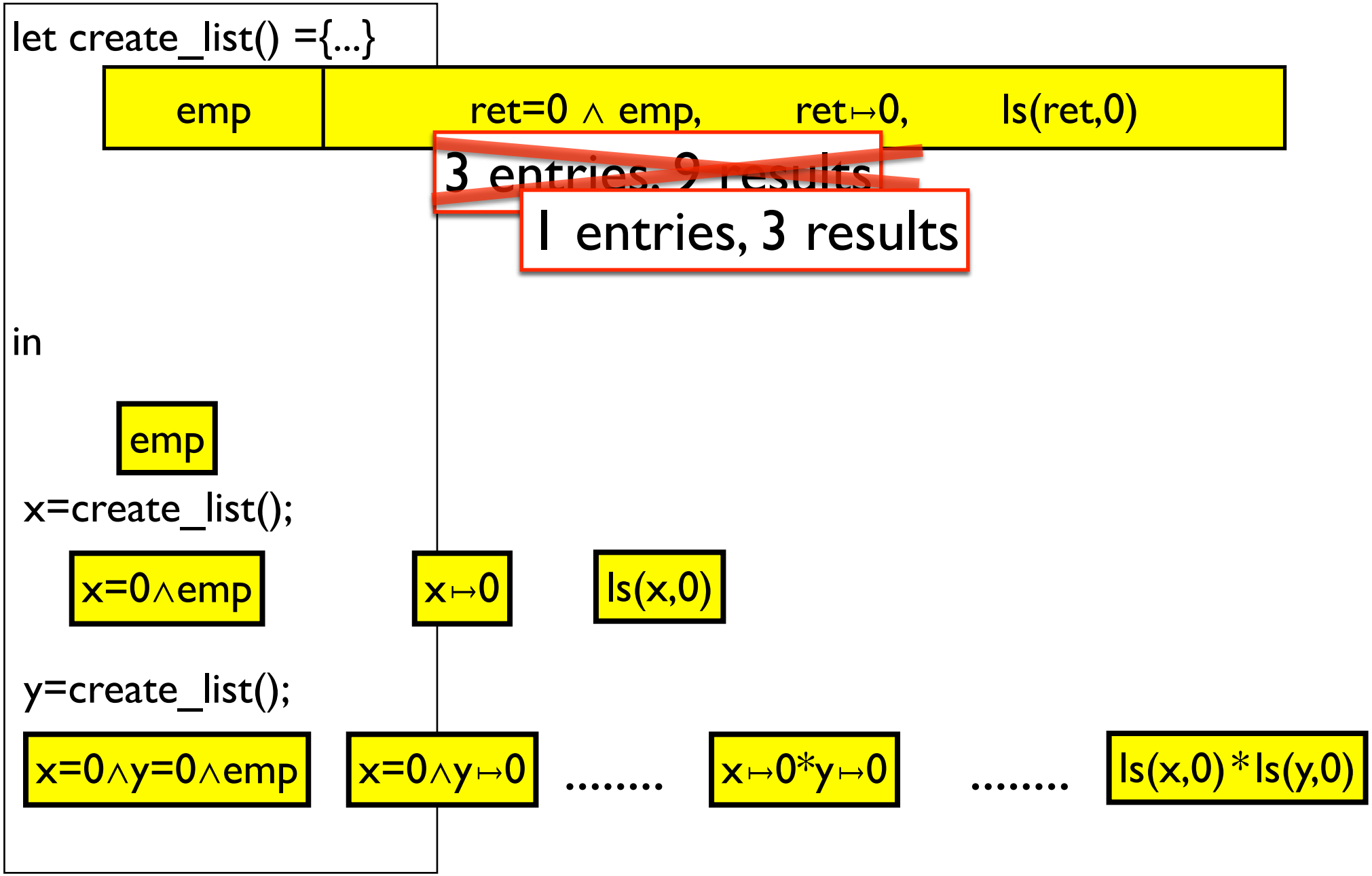
Creation of 2 Lists



Creation of 2 Lists



Creation of 2 Lists



Soundness of Frame Rule and Locality Properties

Soundness of the Frame Rule

$$\frac{\Gamma \vdash \{P\}C\{Q\}}{\Gamma \vdash \{P * R\}C\{Q * R\}} \text{ Modifies}(C) \cap \text{FV}(C) = \emptyset$$

- Soundness of the frame rule relies on specific properties of implementable commands.
- The rule does not hold for the following command:
  - C = "InitializeAllAllocatedCellsToZero".
  - {emp}C{emp}, so {1 ↦20}C{1 ↦20}.
  - But, {1 ↦20}C{1 ↦0}.
- But, such commands are not implementable.

Locality Properties

$$[\![C]\!] \quad : \quad \mathsf{States} \to \mathcal{P}(\mathsf{States} \cup \{\mathsf{err}\})$$

- Safety monotonicity:

$$h \# h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \implies \mathsf{err} \notin [\![C]\!](s, h * h_0)$$

- Frame property:

$$h \# h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \wedge h_1' \in [\![C]\!](s, h * h_0)$$
$$\implies \exists h'.\, h_1' = h' * h_0 \wedge h' \in [\![C]\!](s,h).$$

Locality Properties

$$[\![C]\!] \quad : \quad \mathsf{States} \to \mathcal{P}(\mathsf{States} \cup \{\mathsf{err}\})$$

- Safety monotonicity:

$$h \# h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \implies \mathsf{err} \notin [\![C]\!](s, h * h_0)$$

- Frame property:

$$h \# h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \wedge h_1' \in [\![C]\!](s, h * h_0)$$
$$\implies \exists h'.\, h_1' = h' * h_0 \wedge h' \in [\![C]\!](s,h).$$

Locality Properties

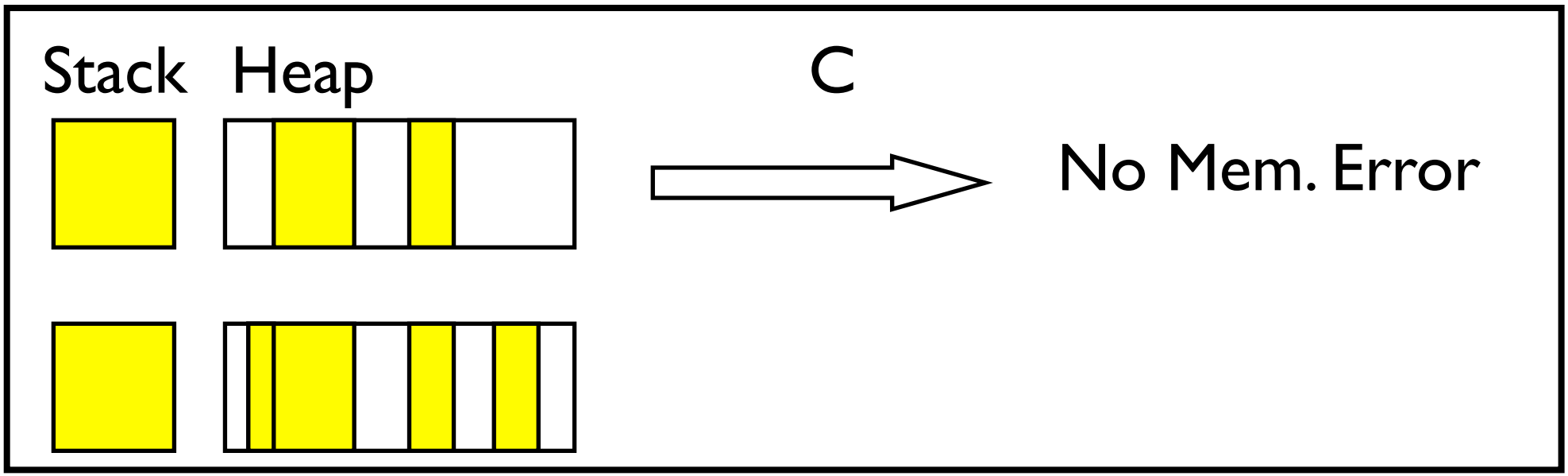
$$[\![C]\!] \;:\; \text{States} \to \mathcal{P}(\text{States} \cup \{\text{err}\})$$

- Safety monotonicity:

$$h \# h_0 \wedge \text{err} \notin [\![C]\!](s,h) \implies \text{err} \notin [\![C]\!](s, h * h_0)$$

- Frame property:

$$h \# h_0 \wedge \text{err} \notin [\![C]\!](s,h) \wedge h_1' \in [\![C]\!](s, h * h_0)$$
$$\implies \exists h'.\, h_1' = h' * h_0 \wedge h' \in [\![C]\!](s,h).$$



Locality Properties

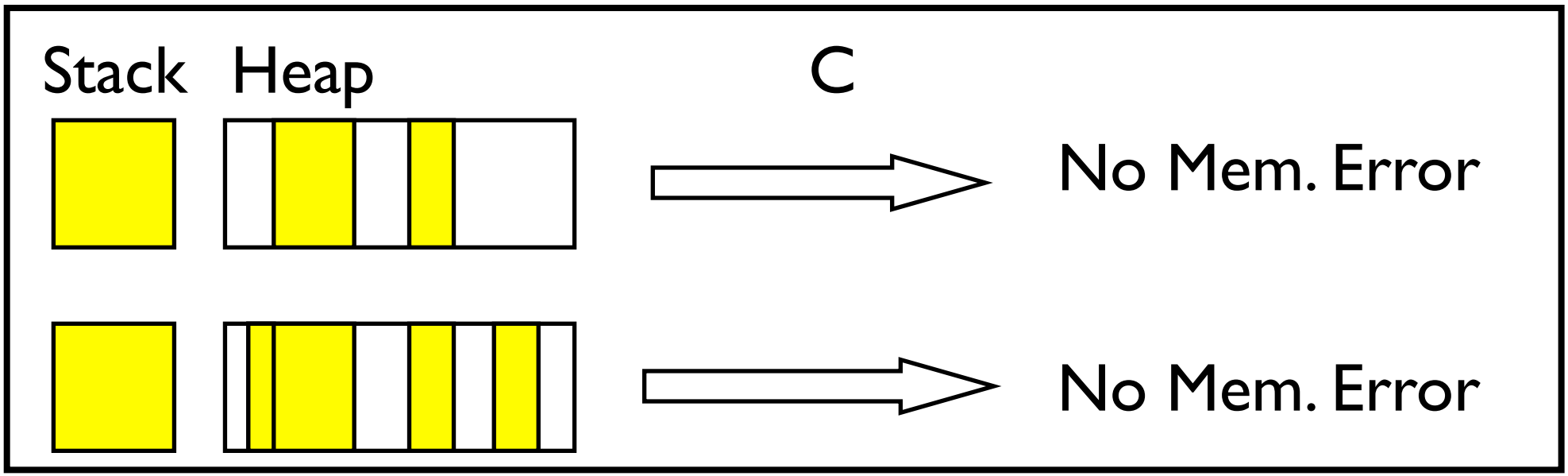
$$[\![C]\!] \;:\; \text{States} \to \mathcal{P}(\text{States} \cup \{\text{err}\})$$

- Safety monotonicity:

$$h \# h_0 \wedge \text{err} \notin [\![C]\!](s,h) \implies \text{err} \notin [\![C]\!](s, h * h_0)$$

- Frame property:

$$h \# h_0 \wedge \text{err} \notin [\![C]\!](s,h) \wedge h_1' \in [\![C]\!](s, h * h_0)$$
$$\implies \exists h'.\, h_1' = h' * h_0 \wedge h' \in [\![C]\!](s,h).$$



Locality Properties

$$[\![C]\!] \quad : \quad \mathsf{States} \to \mathcal{P}(\mathsf{States} \cup \{\mathsf{err}\})$$

- Safety monotonicity:

$$h \# h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \implies \mathsf{err} \notin [\![C]\!](s, h * h_0)$$

- Frame property:

$$h \# h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \wedge h_1' \in [\![C]\!](s, h * h_0)$$
$$\implies \exists h'.\, h_1' = h' * h_0 \wedge h' \in [\![C]\!](s,h).$$

Locality Properties

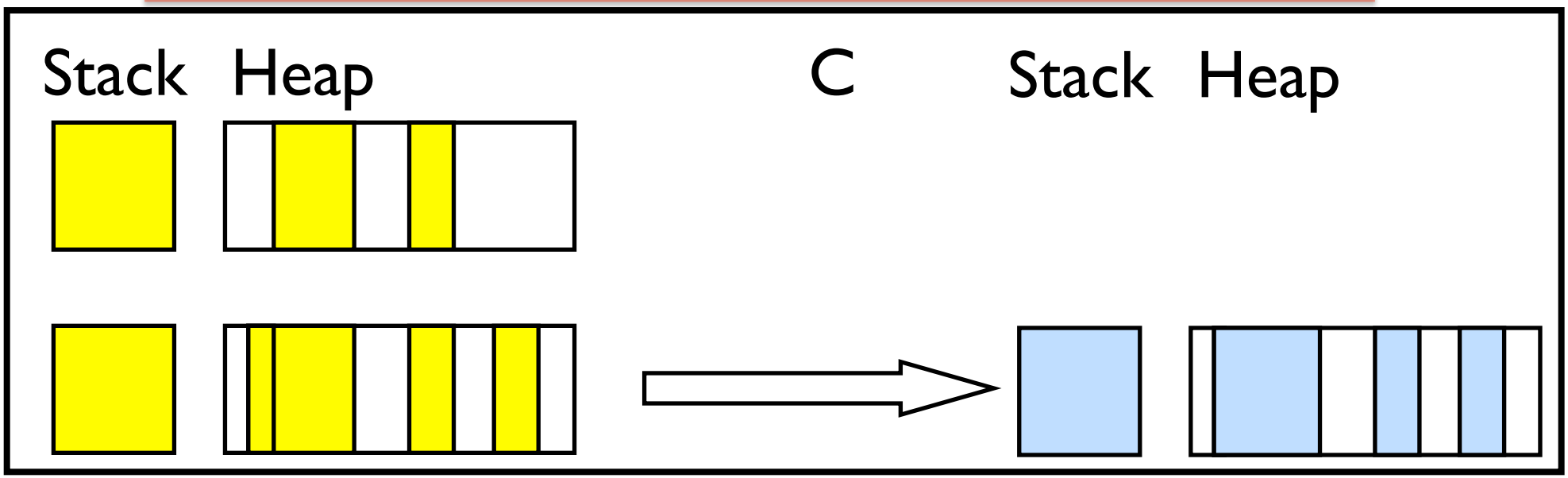
$$[\![C]\!] \;:\; \mathsf{States} \to \mathcal{P}(\mathsf{States} \cup \{\mathsf{err}\})$$

- Safety monotonicity:

$$h \# h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \implies \mathsf{err} \notin [\![C]\!](s, h * h_0)$$

- Frame property:

$$h \# h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \wedge h_1' \in [\![C]\!](s, h * h_0)$$
$$\implies \exists h'.\, h_1' = h' * h_0 \wedge h' \in [\![C]\!](s,h).$$



Locality Properties

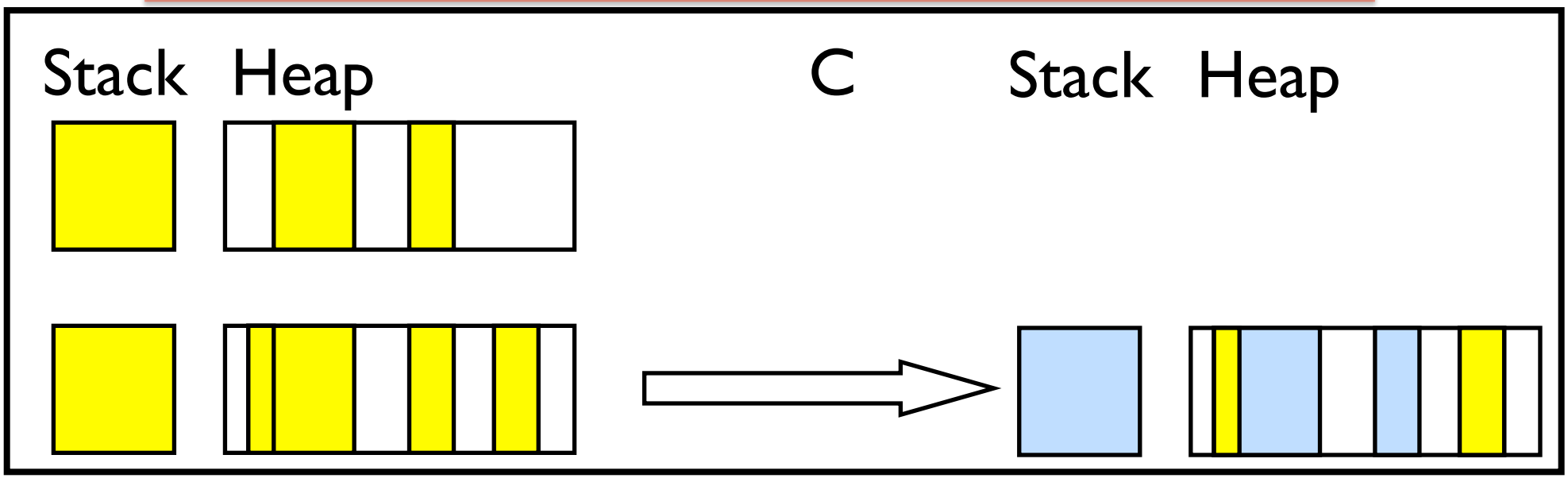
$$[\![C]\!] \;:\; \mathsf{States} \to \mathcal{P}(\mathsf{States} \cup \{\mathsf{err}\})$$

- Safety monotonicity:

$$h\#h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \implies \mathsf{err} \notin [\![C]\!](s, h * h_0)$$

- Frame property:

$$h\#h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \wedge h_1' \in [\![C]\!](s, h * h_0)$$
$$\implies \exists h'.\, h_1' = h' * h_0 \wedge h' \in [\![C]\!](s,h).$$



Locality Properties

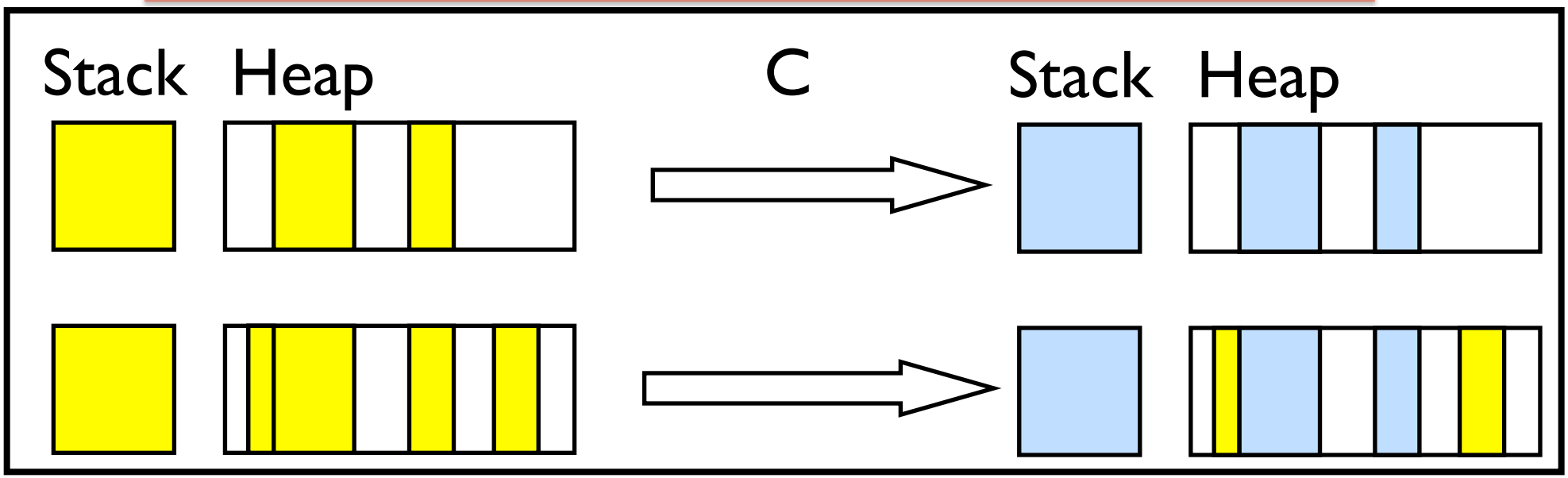
$$[\![C]\!] \;:\; \mathsf{States} \to \mathcal{P}(\mathsf{States} \cup \{\mathsf{err}\})$$

- Safety monotonicity:

$$h\#h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \implies \mathsf{err} \notin [\![C]\!](s, h * h_0)$$

- Frame property:

$$h\#h_0 \wedge \mathsf{err} \notin [\![C]\!](s,h) \wedge h_1' \in [\![C]\!](s, h * h_0)$$
$$\implies \exists h'.\, h_1' = h' * h_0 \wedge h' \in [\![C]\!](s,h).$$



Locality Properties

$$[\![C]\!] \quad : \quad \text{States} \to \mathcal{P}(\text{States} \cup \{\text{err}\})$$

- Safety monotonicity:

$$h \# h_0 \wedge \text{err} \notin [\![C]\!](s,h) \implies \text{err} \notin [\![C]\!](s, h * h_0)$$

- Frame property:

$$h \# h_0 \wedge \text{err} \notin [\![C]\!](s,h) \wedge h_1' \in [\![C]\!](s, h * h_0)$$
$$\implies \exists h'.\, h_1' = h' * h_0 \wedge h' \in [\![C]\!](s,h).$$

- Bad example again:
  - "InitializeAllAllocatedCellsToZero".
- Good example:
  - *7 = 200, ([x:0,...], [7:0]) ([x:0,...], [7:0, 9:0])

HW

- Prove the soundness of the frame rule using the locality properties.

Reference

- Frame Rule:
  - O'Hearn et al's CSL 2001 paper.
  - Yang and O'Hearn's FOSSACS 2002 paper.
- Interproc. Analysis and Frame Rule:
  - Gotsman et al's SAS 2006 paper.
- Heuristic based on reachability:
  - Rinetzky et al's POPL 2005 paper.