Programming Logics and Software Verification

Automating Verification:
Program Analysis with Separation Logic

Josh Berdine

thanks to Hongseok Yang for slides

What's missing to build an automatic verifier?

```
$\{\mathsf{ls}\ (x,0)\}$
 $y = 0;$
 INV : $\mathsf{ls}\ (y,0) * \mathsf{ls}\ (x,0)$
 while $(x \neq 0)$ {
   $\{x{\neq}0 \wedge \mathsf{ls}\ (y,0) * \mathsf{ls}\ (x,0)\}$
   $\{\exists x'.\, \mathsf{ls}\ (y,0) * (x \mapsto x') * \mathsf{ls}\ (x',0)\}$
    $t = x;$
   $\{\exists x'.\, t{=}x \wedge \mathsf{ls}\ (y,0) * (x \mapsto x') * \mathsf{ls}\ (x',0)\}$
   $\{\exists x'.\, \mathsf{ls}\ (y,0) * (t \mapsto x') * \mathsf{ls}\ (x',0)\}$
    $x = *t;$
   $\{\exists x'.\, x{=}x' \wedge \mathsf{ls}\ (y,0) * (t \mapsto x') * \mathsf{ls}\ (x',0)\}$
   $\{\mathsf{ls}\ (y,0) * (t \mapsto x) * \mathsf{ls}\ (x,0)\}$
    $*t = y;$
   $\{\mathsf{ls}\ (y,0) * (t \mapsto y) * \mathsf{ls}\ (x,0)\}$
    $y = t$
   $\{\exists y'.\, y{=}t \wedge \mathsf{ls}\ (y',0) * (t \mapsto y') * \mathsf{ls}\ (x,0)\}$
   $\{\mathsf{ls}\ (y,0) * \mathsf{ls}\ (x,0)\}$
 }
$\{x{=}0 \wedge \mathsf{ls}\ (y,0) * \mathsf{ls}\ (x,0)\}$
$\{\mathsf{ls}\ (y,0)\}$
```

What’s missing to build an automatic verifier?

```
$\{\mathsf{ls}\ (x, 0)\}$
 $y = 0;$
 $\mathsf{INV}: \ \mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)$
 $\mathtt{while}\ (x \neq 0)\ \{$
   $\{x{\neq}0 \wedge \mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$
   $\{\exists x'.\, \mathsf{ls}\ (y, 0) * (x \mapsto x') * \mathsf{ls}\ (x', 0)\}$
    $t = x;$
   $\{\exists x'.\, t{=}x \wedge \mathsf{ls}\ (y, 0) * (x \mapsto x') * \mathsf{ls}\ (x', 0)\}$
   $\{\exists x'.\, \mathsf{ls}\ (y, 0) * (t \mapsto x') * \mathsf{ls}\ (x', 0)\}$
    $x = *t;$
   $\{\exists x'.\, x{=}x' \wedge \mathsf{ls}\ (y, 0) * (t \mapsto x') * \mathsf{ls}\ (x', 0)\}$
   $\{\mathsf{ls}\ (y, 0) * (t \mapsto x) * \mathsf{ls}\ (x, 0)\}$
    $*t = y;$
   $\{\mathsf{ls}\ (y, 0) * (t \mapsto y) * \mathsf{ls}\ (x, 0)\}$
    $y = t$
   $\{\exists y'.\, y{=}t \wedge \mathsf{ls}\ (y', 0) * (t \mapsto y') * \mathsf{ls}\ (x, 0)\}$
   $\{\mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$
 $\}$
$\{x{=}0 \wedge \mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$
$\{\mathsf{ls}\ (y, 0)\}$
```

1. Infer a loop invariant.

$\{\mathsf{ls}\ (x, 0)\}$

$y = 0;$

$\mathsf{INV}: \ \mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)$

$\texttt{while}\ (x \neq 0)\ \{$

$\{x{\neq}0 \wedge \mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$

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

$t = x;$

$\{\exists x'.\, t{=}x \wedge \mathsf{ls}\ (y, 0) * (x \mapsto x') * \mathsf{ls}\ (x', 0)\}$

$\{\exists x'.\, \mathsf{ls}\ (y, 0) * (t \mapsto x') * \mathsf{ls}\ (x', 0)\}$

$x = *t;$

$\{\exists x'.\, x{=}x' \wedge \mathsf{ls}\ (y, 0) * (t \mapsto x') * \mathsf{ls}\ (x', 0)\}$

$\{\mathsf{ls}\ (y, 0) * (t \mapsto x) * \mathsf{ls}\ (x, 0)\}$

$*t = y;$

$\{\mathsf{ls}\ (y, 0) * (t \mapsto y) * \mathsf{ls}\ (x, 0)\}$

$y = t$

$\{\exists y'.\, y{=}t \wedge \mathsf{ls}\ (y', 0) * (t \mapsto y') * \mathsf{ls}\ (x, 0)\}$

$\{\mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$

$\}$

$\{x{=}0 \wedge \mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$

$\{\mathsf{ls}\ (y, 0)\}$

What's missing to build an automatic verifier?

1. Infer a loop invariant.
2. Massage assertions using Consequence.

a) exposing pointsto.

```
$\{\mathsf{ls}\ (x, 0)\}$
 $y = 0;$
 INV : $\mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)$
 while $(x \neq 0)$ {
   $\{x{\neq}0 \land \mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$
   $\{\exists x'.\, \mathsf{ls}\ (y, 0) * (x \mapsto x') * \mathsf{ls}\ (x', 0)\}$
   $t = x;$
   $\{\exists x'.\, t{=}x \land \mathsf{ls}\ (y, 0) * (x \mapsto x') * \mathsf{ls}\ (x', 0)\}$
   $\{\exists x'.\, \mathsf{ls}\ (y, 0) * (t \mapsto x') * \mathsf{ls}\ (x', 0)\}$
   $x = *t;$
   $\{\exists x'.\, x{=}x' \land \mathsf{ls}\ (y, 0) * (t \mapsto x') * \mathsf{ls}\ (x', 0)\}$
   $\{\mathsf{ls}\ (y, 0) * (t \mapsto x) * \mathsf{ls}\ (x, 0)\}$
   $*t = y;$
   $\{\mathsf{ls}\ (y, 0) * (t \mapsto y) * \mathsf{ls}\ (x, 0)\}$
   $y = t$
   $\{\exists y'.\, y{=}t \land \mathsf{ls}\ (y', 0) * (t \mapsto y') * \mathsf{ls}\ (x, 0)\}$
   $\{\mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$
 }
$\{x{=}0 \land \mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$
$\{\mathsf{ls}\ (y, 0)\}$
```

What's missing to build an automatic verifier?

1. Infer a loop invariant.
2. Massage assertions using Consequence.

a) exposing pointsto.

b) removing equality.

```
$\{\mathsf{ls}\ (x, 0)\}$
 $y = 0;$
 INV :  $\mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)$
 while $(x \neq 0)$ {
   $\{x{\neq}0 \wedge \mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$
   $\{\exists x'.\, \mathsf{ls}\ (y, 0) * (x \mapsto x') * \mathsf{ls}\ (x', 0)\}$
    $t = x;$
   $\{\exists x'.\, t{=}x \wedge \mathsf{ls}\ (y, 0) * (x \mapsto x') * \mathsf{ls}\ (x', 0)\}$
   $\{\exists x'.\, \mathsf{ls}\ (y, 0) * (t \mapsto x') * \mathsf{ls}\ (x', 0)\}$
    $x = *t;$
   $\{\exists x'.\, x{=}x' \wedge \mathsf{ls}\ (y, 0) * (t \mapsto x') * \mathsf{ls}\ (x', 0)\}$
   $\{\mathsf{ls}\ (y, 0) * (t \mapsto x) * \mathsf{ls}\ (x, 0)\}$
    $*t = y;$
   $\{\mathsf{ls}\ (y, 0) * (t \mapsto y) * \mathsf{ls}\ (x, 0)\}$
    $y = t$
   $\{\exists y'.\, y{=}t \wedge \mathsf{ls}\ (y', 0) * (t \mapsto y') * \mathsf{ls}\ (x, 0)\}$
   $\{\mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$
 }
$\{x{=}0 \wedge \mathsf{ls}\ (y, 0) * \mathsf{ls}\ (x, 0)\}$
$\{\mathsf{ls}\ (y, 0)\}$
```

What's missing to build an automatic verifier?

1. Infer a loop invariant.
2. Massage assertions using Consequence.

a) exposing pointsto.

b) removing equality.

c) removing emp.

$\{\mathsf{ls}\ (x,0)\}$

$y = 0;$

$\mathsf{INV} : \ \mathsf{ls}\ (y,0) * \mathsf{ls}\ (x,0)$

$\mathtt{while}\ (x \neq 0)\ \{$

$\quad \{x{\neq}0 \wedge \mathsf{ls}\ (y,0) * \mathsf{ls}\ (x,0)\}$

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

$\quad t = x;$

$\quad \{\exists x'.\, t{=}x$ $))\}$

$\quad \{\exists x'.\, \mathsf{ls}\ (y$

$\quad x = *t;$

$\quad \{\exists x'.\, x{=}x$ $0)\}$

$\quad \{\mathsf{ls}\ (y,0) *$

$\quad *t = y;$

$\quad \{\mathsf{ls}\ (y,0) * (t \mapsto y) * \mathsf{ls}\ (x,0)\}$

$\quad y = t$

$\quad \{\exists y'.\, y{=}t \wedge \mathsf{ls}\ (y',0) * (t \mapsto y') * \mathsf{ls}\ (x,0)\}$

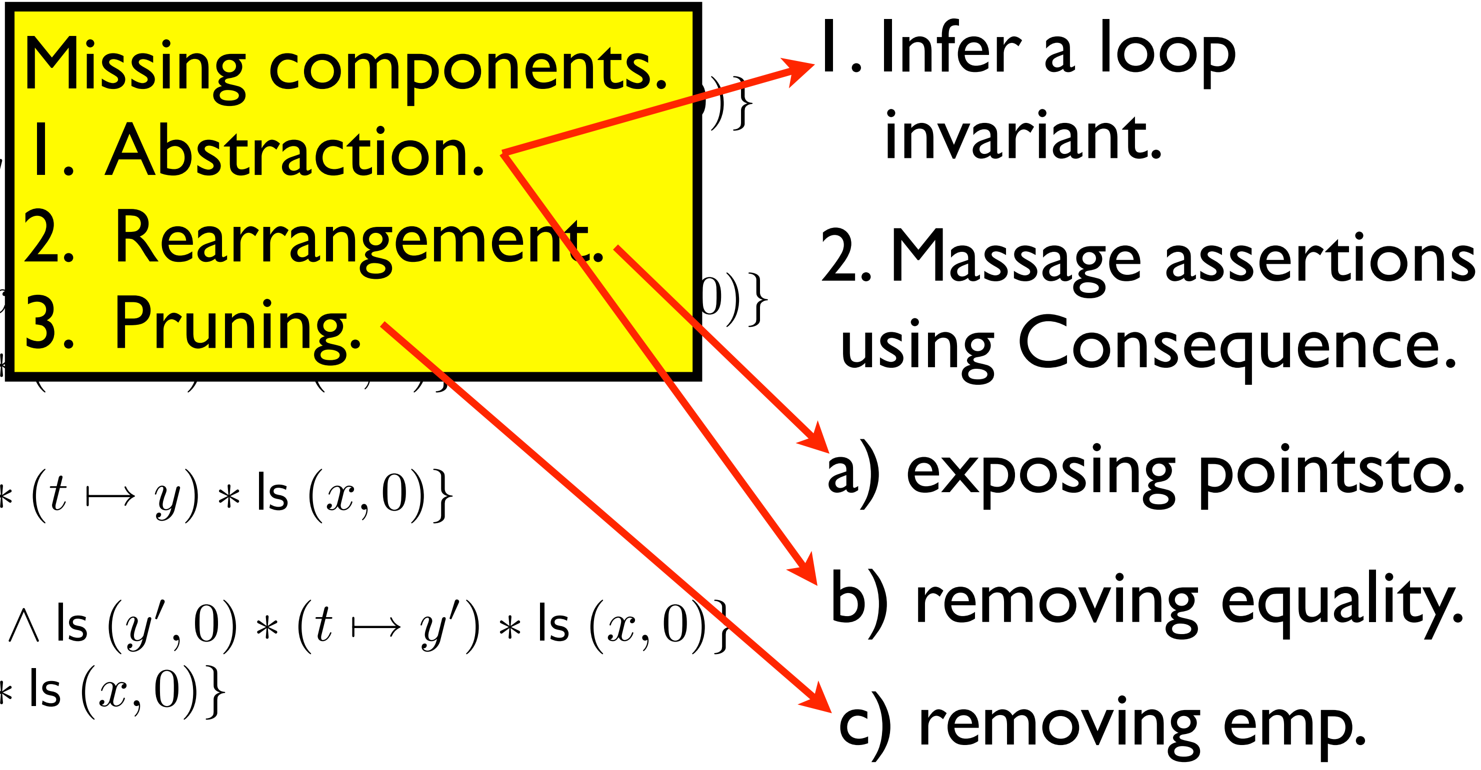
$\quad \{\mathsf{ls}\ (y,0) * \mathsf{ls}\ (x,0)\}$

$\}$

$\{x{=}0 \wedge \mathsf{ls}\ (y,0) * \mathsf{ls}\ (x,0)\}$

$\{\mathsf{ls}\ (y,0)\}$

What's missing to build an automatic verifier?



Missing components.

1. Abstraction.
2. Rearrangement.
3. Pruning.

1. Infer a loop invariant.

2. Massage assertions using Consequence.

a) exposing pointsto.

b) removing equality.

c) removing emp.

Symbolic heaps

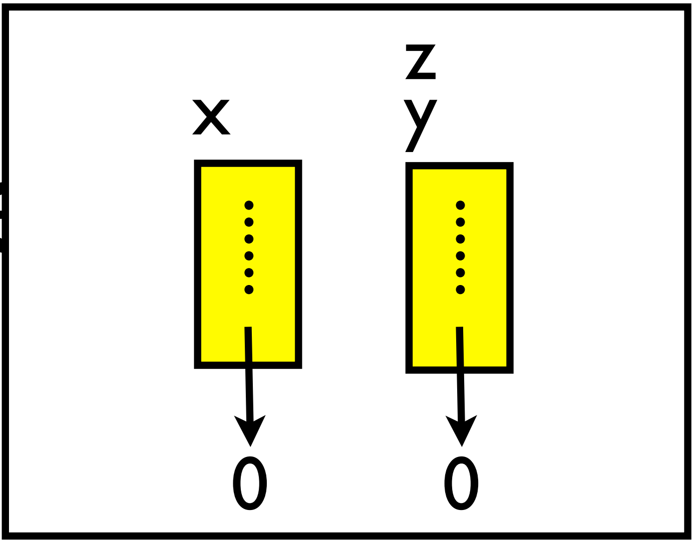
- Assertions of the particular form:

$$\begin{array}{rcl} E, F & ::= & x \mid 0 \\ \Pi & ::= & E = F \mid E \neq F \mid \mathsf{true} \mid \Pi \wedge \Pi' \\ \Sigma & ::= & \mathsf{emp} \mid (E \mapsto F) \mid \mathsf{ls}\,(E, F) \mid \mathsf{true} \mid \Sigma * \Sigma' \\ P, Q & ::= & \exists \vec{x}.\ \Pi \wedge \Sigma \end{array}$$

- Restricted. No negation and no univ. quan.
- E.g.

$$y = z \wedge \mathsf{ls}\,(x, 0) * \mathsf{ls}\,(y, 0)$$
$$\exists v' w'.\, \mathsf{ls}\,(x, v') * y \mapsto v' * v' \mapsto w'$$

Why Symbolic Heaps?
1. Easy to understand visually.
2. Easy to design heuristics for abstraction.



- Assertions of the particular form

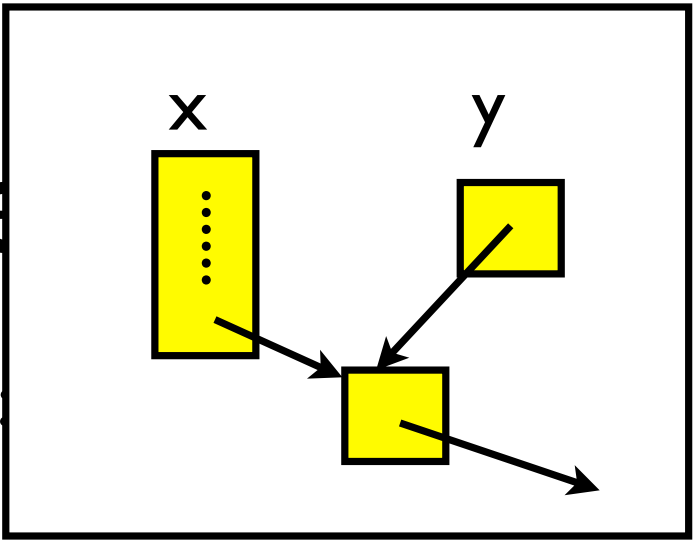
$$\begin{array}{rcl} E, F & ::= & x \mid 0 \\ \Pi & ::= & E = F \mid E \neq F \mid \mathsf{true} \mid \Pi \wedge \Pi' \\ \Sigma & ::= & \mathsf{emp} \mid (E \mapsto F) \mid \mathsf{ls}\,(E, F) \mid \mathsf{true} \mid \Sigma * \Sigma' \\ P, Q & ::= & \exists \vec{x}.\ \Pi \wedge \Sigma \end{array}$$

- Restricted. No negation and no univ. quan.
- E.g.

$$\boxed{y = z \wedge \mathsf{ls}\,(x, 0) * \mathsf{ls}\,(y, 0)}$$

$$\exists v' w'. \mathsf{ls}\,(x, v') * y \mapsto v' * v' \mapsto w'$$

Why Symbolic Heaps?
1. Easy to understand visually.
2. Easy to design heuristics for abstraction.



- Assertions of the particular form:

$$\begin{array}{rcl} E, F & ::= & x \mid 0 \\ \Pi & ::= & E = F \mid E \neq F \mid \mathsf{true} \mid \Pi \wedge \Pi' \\ \Sigma & ::= & \mathsf{emp} \mid (E \mapsto F) \mid \mathsf{ls}\,(E, F) \mid \mathsf{true} \mid \Sigma * \Sigma' \\ P, Q & ::= & \exists \vec{x}.\ \Pi \wedge \Sigma \end{array}$$

- Restricted. No negation and no univ. quan.
- E.g.

$$y = z \wedge \mathsf{ls}\,(x, 0) * \mathsf{ls}\,(y, 0)$$

$$\boxed{\exists v' w'.\, \mathsf{ls}\,(x, v') * y \mapsto v' * v' \mapsto w'}$$

Analysis algorithm

- Input: a set (i.e., disjunction) of sym. heaps, and a program.
- Output: a set of sym. heaps at each program point.
- Finds a proof sketch in separation logic.
- Algorithm:
  - Abstractly run a program with sym. heaps.
  - Accumulate all the obtained sym. heaps.
  - Repeat until no changes.

Analysis algorithm

- Input: a set (i.e., disjunction) of sym. heaps, and a program.
- Output: a set of sym. heaps at each program point
- Finds a proof sketch in s
- Algorithm:
  - Abstractly run a program with sym. heaps.
  - Accumulate all the obtained sym. heaps.
  - Repeat until no changes.

Rearrangement +
Proof rule in sep. logic +
Abstraction

create

emp

```
h = 0;


while (nondet)
{

   t = new(1);


   *t=h;


   h=t;


   t=0;

}
```

create

emp

h = 0;

h=0 ∧ emp

```
while (nondet)
{

   t = new(1);


   *t=h;


   h=t;


   t=0;

}
```

create

```
emp
h = 0;
        h=0 ∧ emp
while (nondet)
{
        h=0 ∧ emp
   t = new(1);


   *t=h;


   h=t;


   t=0;

}
```

create

```
emp
h = 0;
        h=0 ∧ emp
while (nondet)
{
        h=0 ∧ emp
   t = new(1);
      ∃t'. h=0 ∧ t↦t'
   *t=h;


   h=t;


   t=0;

}
```

create

```
            emp
h = 0;
            h=0 ∧ emp
while (nondet)
{
            h=0 ∧ emp
   t = new(1);
         ∃t’. h=0 ∧ t↦t’
   *t=h;
         ∃t’. h=0 ∧ t↦h
   h=t;


   t=0;

}
```

create

```
                emp
h = 0;
                h=0 ∧ emp
while (nondet)
{
                h=0 ∧ emp
   t = new(1);
            ∃t’. h=0 ∧ t↦t’
   *t=h;
            h=0 ∧ t↦0
   h=t;


   t=0;

}
```

create

```
                emp
h = 0;
                h=0 ∧ emp
while (nondet)
{
                h=0 ∧ emp
    t = new(1);
            ∃t’. h=0 ∧ t↦t’
    *t=h;
            h=0 ∧ t↦0
    h=t;
            ∃h’. h=t ∧ h’=0 ∧ t↦0
    t=0;

}
```

create

```
emp
h = 0;
        h=0 ∧ emp
while (nondet)
{
        h=0 ∧ emp
   t = new(1);
     ∃t'. h=0 ∧ t↦t'
   *t=h;
     h=0 ∧ t↦0
   h=t;
     h=t ∧ t↦0
   t=0;

}
```

create

```
            emp
h = 0;
            $h=0 \wedge emp$
while (nondet)
{
            $h=0 \wedge emp$
   t = new(1);
            $\exists t'. h=0 \wedge t \mapsto t'$
   *t=h;
            $h=0 \wedge t \mapsto 0$
   h=t;
            $h=t \wedge t \mapsto 0$
   t=0;
            $\exists t'. t=0 \wedge h=t' \wedge t' \mapsto 0$
}
```

create

```
        $\mathrm{emp}$
h = 0;
        $h=0 \wedge \mathrm{emp}$
while (nondet)
{
        $h=0 \wedge \mathrm{emp}$
    t = new(1);
        $\exists t'. h=0 \wedge t \mapsto t'$
    *t=h;
        $h=0 \wedge t \mapsto 0$
    h=t;
        $h=t \wedge t \mapsto 0$
    t=0;
        $t=0 \wedge h \mapsto 0$
}
```

create

```
emp
h = 0;
        h=0 ∧ emp
while (nondet)
{
        h=0 ∧ emp
    t = new(1);
      ∃t’. h=0 ∧ t↦t’
    *t=h;
      h=0 ∧ t↦0
    h=t;
      h=t ∧ t↦0
    t=0;
      t=0 ∧ h↦0
}
t=0 ∧ h↦0
```

create

```
emp
h = 0;
        h=0 ∧ emp
while (nondet)
{
        h=0 ∧ emp
   t = new(1);
      ∃t'. h=0 ∧ t↦t'
   *t=h;
      h=0 ∧ t↦0
   h=t;
      h=t ∧ t↦0
   t=0;
      t=0 ∧ h↦0
}
```

t=0 ∧ h↦0

t=0 ∧ h↦0

create

emp

h = 0;

$h=0 \wedge \mathrm{emp}$ | $t=0 \wedge h \mapsto 0$

while (nondet)

{

$h=0 \wedge \mathrm{emp}$ | $t=0 \wedge h \mapsto 0$

t = new(1);

$\exists t'.\, h=0 \wedge t \mapsto t'$ | $\exists t' t''.\, t'=0 \wedge t \mapsto t'' * h \mapsto 0$

*t=h;

$h=0 \wedge t \mapsto 0$

h=t;

$h=t \wedge t \mapsto 0$

t=0;

$t=0 \wedge h \mapsto 0$

}

create

```
            emp
h = 0;
            h=0 ∧ emp                    t=0 ∧ h↦0
while (nondet)
{
            h=0 ∧ emp                    t=0 ∧ h↦0
    t = new(1);
        ∃t’. h=0 ∧ t↦t’             ∃t’’. t↦t’’ * h↦0
    *t=h;
        h=0 ∧ t↦0
    h=t;
        h=t ∧ t↦0
    t=0;
        t=0 ∧ h↦0
}
```

create

```
          emp
h = 0;
              h=0 ∧ emp              t=0 ∧ h↦0
while (nondet)
{
              h=0 ∧ emp              t=0 ∧ h↦0
   t = new(1);
          ∃t'. h=0 ∧ t↦t'        ∃t''. t↦t'' * h↦0
   *t=h;
          h=0 ∧ t↦0                 ∃t''. t↦h * h↦0
   h=t;
          h=t ∧ t↦0
   t=0;
          t=0 ∧ h↦0
}
```

create

```
        emp
h = 0;
            h=0 ∧ emp              t=0 ∧ h↦0
while (nondet)
{
            h=0 ∧ emp              t=0 ∧ h↦0
   t = new(1);
        ∃t’. h=0 ∧ t↦t’            ∃t”. t↦t” * h↦0
   *t=h;
        h=0 ∧ t↦0                  t↦h * h↦0
   h=t;
        h=t ∧ t↦0
   t=0;
        t=0 ∧ h↦0
}
```

create

```
            emp
h = 0;
            h=0 ∧ emp                 t=0 ∧ h↦0
while (nondet)
{
            h=0 ∧ emp                 t=0 ∧ h↦0
   t = new(1);
        ∃t'. h=0 ∧ t↦t'          ∃t''. t↦t'' * h↦0
   *t=h;
        h=0 ∧ t↦0                    t↦h * h↦0
   h=t;
        h=t ∧ t↦0                ∃h'. h=t ∧ t↦h' * h'↦0
   t=0;
        t=0 ∧ h↦0
}
```

create

```
                  emp
h = 0;
          h=0 ∧ emp                  t=0 ∧ h↦0
while (nondet)
{
          h=0 ∧ emp                  t=0 ∧ h↦0
   t = new(1);
       ∃t’. h=0 ∧ t↦t’           ∃t’’. t↦t’’ * h↦0
   *t=h;
       h=0 ∧ t↦0                   t↦h * h↦0
   h=t;
       h=t ∧ t↦0                   ∃h’. h=t ∧ ls (t,0)
   t=0;
       t=0 ∧ h↦0
}
```

create

emp

h = 0;

$h=0 \wedge \mathrm{emp}$ $t=0 \wedge h \mapsto 0$

while (nondet)

{

$h=0 \wedge \mathrm{emp}$ $t=0 \wedge h \mapsto 0$

t = new(1);

$\exists t'.\, h=0 \wedge t \mapsto t'$ $\exists t''.\, t \mapsto t'' * h \mapsto 0$

*t=h;

$h=0 \wedge t \mapsto 0$ $t \mapsto h * h \mapsto 0$

h=t;

$h=t \wedge t \mapsto 0$ $h=t \wedge \mathrm{ls}(t,0)$

t=0;

$t=0 \wedge h \mapsto 0$

}

create

emp

h = 0;

$h=0 \land \mathrm{emp}$

$t=0 \land h \mapsto 0$

while (nondet)

{

$h=0 \land \mathrm{emp}$

$t=0 \land h \mapsto 0$

t = new(1);

$\exists t'.\, h=0 \land t \mapsto t'$

$\exists t''.\, t \mapsto t'' * h \mapsto 0$

*t=h;

$h=0 \land t \mapsto 0$

$t \mapsto h * h \mapsto 0$

h=t;

$h=t \land t \mapsto 0$

$h=t \land \mathrm{ls}(t,0)$

t=0;

$t=0 \land h \mapsto 0$

$\exists t'.\, t=0 \land h=t' \land \mathrm{ls}(t',0)$

}

create

```
           emp
h = 0;
           h=0 ∧ emp                t=0 ∧ h↦0
while (nondet)
{
           h=0 ∧ emp                t=0 ∧ h↦0
   t = new(1);
       ∃t’. h=0 ∧ t↦t’          ∃t”. t↦t” * h↦0
   *t=h;
       h=0 ∧ t↦0                   t↦h * h↦0
   h=t;
       h=t ∧ t↦0                   h=t ∧ ls(t,0)
   t=0;
       t=0 ∧ h↦0                   t=0 ∧ ls(h,0)
}
```

create

```
              emp
h = 0;
              h=0 ∧ emp          t=0 ∧ h↦0                    t=0 ∧ ls(h,0)
while (nondet)
{
              h=0 ∧ emp          t=0 ∧ h↦0
   t = new(1);
         ∃t'. h=0 ∧ t↦t'      ∃t''. t↦t'' * h↦0
   *t=h;
         h=0 ∧ t↦0              t↦h * h↦0
   h=t;
         h=t ∧ t↦0              h=t ∧ ls(t,0)
   t=0;
         t=0 ∧ h↦0              t=0 ∧ ls(h,0)
}
```

create

emp

h = 0;

$h=0 \wedge \mathrm{emp}$ $\quad$ $t=0 \wedge h \mapsto 0$ $\quad$ $t=0 \wedge \mathrm{ls}(h,0)$

while (nondet)

{

$h=0 \wedge \mathrm{emp}$ $\quad$ $t=0 \wedge h \mapsto 0$

t = new(1);

$\exists t'.\, h=0 \wedge t \mapsto t'$ $\quad$ $\exists t''.\, t \mapsto t'' * h \mapsto 0$

*t=h;

$h=0 \wedge t \mapsto 0$ $\quad$ $t \mapsto h * h \mapsto 0$

h=t;

$h=t \wedge t \mapsto 0$ $\quad$ $h=t \wedge \mathrm{ls}(t,0)$

t=0;

$t=0 \wedge h \mapsto 0$ $\quad$ $t=0 \wedge \mathrm{ls}(h,0)$ $\quad$ $t=0 \wedge \mathrm{ls}(h,0)$

}

create

	emp		
`h = 0;`			
	$h=0 \wedge \mathrm{emp}$	$t=0 \wedge h \mapsto 0$	$t=0 \wedge \mathrm{ls}(h,0)$
`while (nondet)`			
`{`	$h=0 \wedge \mathrm{emp}$	$t=0 \wedge h \mapsto 0$	
`t = new(1);`			
	$\exists t'.\ h=0 \wedge t \mapsto t'$	$\exists t''.\ t \mapsto t'' * h \mapsto 0$	
`*t=h;`			
	$h=0 \wedge t \mapsto 0$	$t \mapsto h * h \mapsto 0$	
`h=t;`			
	$h=t \wedge t \mapsto 0$	$h=t \wedge \mathrm{ls}(t,0)$	
`t=0;`			
	$t=0 \wedge h \mapsto 0$	$t=0 \wedge \mathrm{ls}(h,0)$	$t=0 \wedge \mathrm{ls}(h,0)$
`}`			

create

emp

h = 0;

$h=0 \wedge \mathrm{emp}$ $t=0 \wedge h \mapsto 0$ $t=0 \wedge \mathrm{ls}(h,0)$

while (nondet)

{

$h=0 \wedge \mathrm{emp}$ $t=0 \wedge h \mapsto 0$

t = new(1);

$\exists t'.\, h=0 \wedge t \mapsto t'$ $\exists t''.\, t \mapsto t'' * h \mapsto 0$

*t=h;

$h=0 \wedge t \mapsto 0$ $t \mapsto h * h \mapsto 0$

h=t;

$h=t \wedge t \mapsto 0$ $h=t \wedge \mathrm{ls}(t,0)$

t=0;

$t=0 \wedge h \mapsto 0$ $t=0 \wedge \mathrm{ls}(h,0)$ $t=0 \wedge \mathrm{ls}(h,0)$

}

$h=0 \wedge \mathrm{emp}$ $t=0 \wedge h \mapsto 0$ $t=0 \wedge \mathrm{ls}(h,0)$

create

emp

h = 0;

$h=0 \wedge \mathrm{emp}$ $t=0 \wedge h\mapsto 0$ $t=0 \wedge \mathrm{ls}(h,0)$

while (nondet)

{

$h=0 \wedge \mathrm{emp}$ $t=0 \wedge h\mapsto 0$

t = new(1);

The output is a proof sketch of

$\{\ \mathrm{emp}\ \}\mathrm{create}\{\ (h=0\wedge \mathrm{emp}) \vee (t=0\wedge h\mapsto 0) \vee (t=0\wedge \mathrm{ls}(h,0))\ \}$

h=t;

$\exists h'.\ h=t \wedge h'=0 \wedge t\mapsto h'$ $\exists h'.\ h=t \wedge t\mapsto h' * h'\mapsto 0$

t=0;

$\exists t'.\ t=0 \wedge h=t' \wedge t'\mapsto 0$ $\exists t'.\ t=0 \wedge h=t' \wedge \mathrm{ls}(t',0)$ $t=0 \wedge \mathrm{ls}(h,0)$

}

$h=0 \wedge \mathrm{emp}$ $t=0 \wedge h\mapsto 0$ $t=0 \wedge \mathrm{ls}(h,0)$

Abstract semantics of atomic command A

$$\begin{aligned} (\!|A|\!) &: \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap})^{\top} \\ (\!|A|\!) &= \mathsf{Abs} \circ \mathsf{RuleApply}_{A} \circ \mathsf{Rearr}_{A} \end{aligned}$$

Abstract semantics of atomic command A

$$\begin{array}{lcl} (\!|A|\!) & : & \mathsf{SymHeap} \longrightarrow \mathcal{P}(\mathsf{SymHeap})^{\top} \\ (\!|A|\!) & = & \mathsf{Abs} \circ \mathsf{RuleApply}_{A} \circ \boxed{\mathsf{Rearr}_{A}} \end{array}$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

$$\frac{}{\{\exists \vec{x'}.\ \Pi \wedge x {\mapsto} x' * \Sigma\} {*}x{=}t \{\exists \vec{x'}.\ \Pi \wedge x {\mapsto} t * \Sigma\}}$$

$$\dfrac{\dfrac{\overline{\{x{\mapsto}x' * (\Pi \wedge \Sigma)\}{*}x{=}t\{x{\mapsto}t * (\Pi \wedge \Sigma)\}}}{\{\Pi \wedge x{\mapsto}x' * \Sigma\}{*}x{=}t\{\Pi \wedge x{\mapsto}t * \Sigma\}}\ \text{Conseq.}}{\{\exists\vec{x'}.\ \Pi \wedge x{\mapsto}x' * \Sigma\}{*}x{=}t\{\exists\vec{x'}.\ \Pi \wedge x{\mapsto}t * \Sigma\}}\ \text{Exist.}$$

omic command A

SymHeap)$^\top$

$$\llparenthesis A \rrparenthesis \quad = \quad \mathsf{Abs} \circ \mathsf{RuleApply}_A \circ \boxed{\mathsf{Rearr}_A}$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

$$\dfrac{}{\{\exists\vec{x'}.\ \Pi \wedge x{\mapsto}x' * \Sigma\}{*}x{=}t\{\exists\vec{x'}.\ \Pi \wedge x{\mapsto}t * \Sigma\}}$$

omic command A

$$\dfrac{\dfrac{\overline{\{x\mapsto x' * (\Pi \wedge \Sigma)\}*x{=}t\{x\mapsto t * (\Pi \wedge \Sigma)\}}}{\{\Pi \wedge x\mapsto x' * \Sigma\}*x{=}t\{\Pi \wedge x\mapsto t * \Sigma\}}\ \text{Conseq.}}{\{\exists \vec{x'}.\ \Pi \wedge x\mapsto x' * \Sigma\}*x{=}t\{\exists \vec{x'}.\ \Pi \wedge x\mapsto t * \Sigma\}}\ \text{Exist.}$$

$\mathsf{SymHeap})^{\top}$

$$(\!|A|\!) \quad = \quad \mathsf{Abs} \circ \mathsf{RuleApply}_A \circ \boxed{\mathsf{Rearr}_A}$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

$$\overline{\{\exists \vec{x'}.\ \Pi \wedge x\mapsto x' * \Sigma\}*x{=}t\{\exists \vec{x'}.\ \Pi \wedge x\mapsto t * \Sigma\}}$$

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

$\Downarrow$

$$\cfrac{\cfrac{\overline{\{x\mapsto x' * (\Pi \wedge \Sigma)\}{*x{=}t}\{x\mapsto t * (\Pi \wedge \Sigma)\}}}{\{\Pi \wedge x\mapsto x' * \Sigma\}{*x{=}t}\{\Pi \wedge x\mapsto t * \Sigma\}}\ \text{Conseq.}}{\{\exists \vec{x'}.\ \Pi \wedge x\mapsto x' * \Sigma\}{*x{=}t}\{\exists \vec{x'}.\ \Pi \wedge x\mapsto t * \Sigma\}}\ \text{Exist.}$$

omic command A

SymHeap)$^{\top}$

$$(\!|A|\!) \quad = \quad \mathsf{Abs} \circ \mathsf{RuleApply}_A \circ \boxed{\mathsf{Rearr}_A}$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

$$\overline{\{\exists \vec{x'}.\ \Pi \wedge x\mapsto x' * \Sigma\}{*x{=}t}\{\exists \vec{x'}.\ \Pi \wedge x\mapsto t * \Sigma\}}$$

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

$$\Downarrow$$

$$\{\ \ \exists z'.\, \underline{(x{=}z')} \wedge (z\mapsto x) * (x\mapsto 0),\ \ \exists x' z'.\underline{(x\mapsto x') * \mathsf{ls}\,(x', z')} * (z\mapsto z') * (z'\mapsto 0)\ \ \}$$

Abstract semantics of atomic command A

$$
\begin{aligned}
(\!|A|\!) &: \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap})^{\top} \\
(\!|A|\!) &= \mathsf{Abs} \circ \boxed{\mathsf{RuleApply}_A} \circ \mathsf{Rearr}_A
\end{aligned}
$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

[Step2] Apply the rule.

$$\frac{}{\{\exists \vec{x'}.\ \Pi \wedge x{\mapsto}x' * \Sigma\} {*x{=}t} \{\exists \vec{x'}.\ \Pi \wedge x{\mapsto}t * \Sigma\}}$$

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

$$\Downarrow$$

$$\{\ \ \exists z'.\, (x{=}z') \wedge (z{\mapsto}x) * (x{\mapsto}0),\ \ \exists x'z'.(x{\mapsto}x') * \mathsf{ls}\,(x', z') * (z{\mapsto}z') * (z'{\mapsto}0)\ \ \}$$

Abstract semantics of atomic command A

$$(\!|A|\!) \quad : \quad \mathsf{SymHeap} \longrightarrow \mathcal{P}(\mathsf{SymHeap})^{\top}$$

$$(\!|A|\!) \quad = \quad \mathsf{Abs} \circ \boxed{\mathsf{RuleApply}_A} \circ \mathsf{Rearr}_A$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

[Step2] Apply the rule.

$$\frac{}{\{\exists \vec{x'}.\ \Pi \wedge x{\mapsto}x' * \Sigma\}{*}x{=}t\{\exists \vec{x'}.\ \Pi \wedge x{\mapsto}t * \Sigma\}}$$

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

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z{\mapsto}x) * \underline{(x{\mapsto}0)},\ \ \exists x' z'.\underline{(x{\mapsto}x')} * \mathsf{ls}\,(x', z') * (z{\mapsto}z') * (z'{\mapsto}0)\ \}$$

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z{\mapsto}x) * \underline{(x{\mapsto}t)},\ \ \exists x' z'.\underline{(x{\mapsto}t)} * \mathsf{ls}\,(x', z') * (z{\mapsto}z') * (z'{\mapsto}0)\ \}$$

Abstract semantics of atomic command A

$$\begin{aligned} (\!|A|\!) &\;:\; \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap})^{\top} \\ (\!|A|\!) &\;=\; \boxed{\mathsf{Abs}} \circ \mathsf{RuleApply}_A \circ \mathsf{Rearr}_A \end{aligned}$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

$$\frac{}{\{\exists \vec{x'}.\ \Pi \wedge x{\mapsto}x' * \Sigma\} {*}x{=}t \{\exists \vec{x'}.\ \Pi \wedge x{\mapsto}t * \Sigma\}}$$

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

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z{\mapsto}x) * (x{\mapsto}0),\ \ \exists x'z'.(x{\mapsto}x') * \mathsf{ls}\,(x', z') * (z{\mapsto}z') * (z'{\mapsto}0)\ \}$$

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z{\mapsto}x) * (x{\mapsto}t),\ \ \exists x'z'.(x{\mapsto}t) * \mathsf{ls}\,(x', z') * (z{\mapsto}z') * (z'{\mapsto}0)\ \}$$

Abstract semantics of atomic command A

$$(\!|A|\!) \quad : \quad \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^{\top}$$

$$(\!|A|\!) \quad = \quad \boxed{\mathsf{Abs}} \circ \mathsf{RuleApply}_A \circ \mathsf{Rearr}_A$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

$$\overline{\{\exists \vec{x}'.\, \Pi \wedge x \mapsto x' * \Sigma\} * x{=}t \{\exists \vec{x}'.\, \Pi \wedge x \mapsto t * \Sigma\}}$$

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

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z \mapsto x) * (x \mapsto 0),\ \ \exists x' z'.(x \mapsto x') * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \}$$

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z \mapsto x) * (x \mapsto t),\ \ \exists x' z'.(x \mapsto t) * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \}$$

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z \mapsto x) * (x \mapsto t),\ \ \exists x' z'.(x \mapsto t) * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \}$$

Abstract semantics of atomic command A

$$(\!|A|\!) \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap})^{\top}$$

$$(\!|A|\!) \quad = \quad \boxed{\mathsf{Abs}} \circ \mathsf{RuleApply}_A \circ \mathsf{Rearr}_A$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

$$\overline{\{\exists \vec{x'}.\ \Pi \wedge x{\mapsto}x' * \Sigma\}{*}x{=}t\{\exists \vec{x'}.\ \Pi \wedge x{\mapsto}t * \Sigma\}}$$

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

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z{\mapsto}x) * (x{\mapsto}0),\ \ \exists x'z'.(x{\mapsto}x') * \mathsf{ls}\,(x', z') * (z{\mapsto}z') * (z'{\mapsto}0)\ \}$$

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z{\mapsto}x) * (x{\mapsto}t),\ \ \exists x'z'.(x{\mapsto}t) * \mathsf{ls}\,(x', z') * (z{\mapsto}z') * (z'{\mapsto}0)\ \}$$

$$\Downarrow$$

$$\{\ (z{\mapsto}x) * (x{\mapsto}t),\ \ \exists x'z'.(x{\mapsto}t) * \mathsf{ls}\,(x', z') * (z{\mapsto}z') * (z'{\mapsto}0)\ \}$$

Abstract semantics of atomic command A

$$
\begin{aligned}
(\!| A |\!) &: \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^{\top} \\
(\!| A |\!) &= \boxed{\mathsf{Abs}} \circ \mathsf{RuleApply}_A \circ \mathsf{Rearr}_A
\end{aligned}
$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

$$\frac{}{\{\exists \vec{x'}.\ \Pi \wedge x \mapsto x' * \Sigma\} {*x}{=}t \{\exists \vec{x'}.\ \Pi \wedge x \mapsto t * \Sigma\}}$$

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

⇩

$$\{\ \exists z'.\ (x{=}z') \wedge (z \mapsto x) * (x \mapsto 0),\ \ \exists x' z'.(x \mapsto x') * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \}$$

⇩

$$\{\ \exists z'.\ (x{=}z') \wedge (z \mapsto x) * (x \mapsto t),\ \ \exists x' z'.(x \mapsto t) * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \}$$

⇩

$$\{\ (z \mapsto x) * (x \mapsto t),\ \ \exists x' z'.(x \mapsto t) * \mathsf{true} * (z \mapsto z') * (z' \mapsto 0)\ \}$$

Abstract semantics of atomic command A

$$(\!|A|\!) \quad : \quad \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^{\top}$$
$$(\!|A|\!) \quad = \quad \boxed{\mathsf{Abs}} \circ \mathsf{RuleApply}_A \circ \mathsf{Rearr}_A$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

$$\frac{}{\{\exists \vec{x}'.\, \Pi \wedge x \mapsto x' * \Sigma\} *x{=}t \{\exists \vec{x}'.\, \Pi \wedge x \mapsto t * \Sigma\}}$$

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

⇩

$$\{\ \exists z'.\, (x{=}z') \wedge (z \mapsto x) * (x \mapsto 0),\ \ \exists x' z'.(x \mapsto x') * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \}$$

⇩

$$\{\ \exists z'.\, (x{=}z') \wedge (z \mapsto x) * (x \mapsto t),\ \ \exists x' z'.(x \mapsto t) * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \}$$

⇩

$$\{\ (z \mapsto x) * (x \mapsto t),\ \ \exists x' z'.(x \mapsto t) * \mathsf{true} * (\mathsf{ls}\,(z, 0))\ \}$$

Abstract semantics of atomic command A

$$(\!|A|\!) \quad : \quad \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^{\top}$$

$$(\!|A|\!) \quad = \quad \boxed{\mathsf{Abs}} \circ \mathsf{RuleApply}_A \circ \mathsf{Rearr}_A$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

$$\frac{}{\{\exists \vec{x'}.\ \mathit{\Pi} \wedge x{\mapsto}x' * \mathit{\Sigma}\}{*}x{=}t\{\exists \vec{x'}.\ \mathit{\Pi} \wedge x{\mapsto}t * \mathit{\Sigma}\}}$$

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

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z{\mapsto}x) * (x{\mapsto}0),\quad \exists x'z'.(x{\mapsto}x') * \mathsf{ls}\,(x', z') * (z{\mapsto}z') * (z'{\mapsto}0)\ \}$$

$$\Downarrow$$

$$\{\ \exists z'.\,(x{=}z') \wedge (z{\mapsto}x) * (x{\mapsto}t),\quad \exists x'z'.(x{\mapsto}t) * \mathsf{ls}\,(x', z') * (z{\mapsto}z') * (z'{\mapsto}0)\ \}$$

$$\Downarrow$$

$$\{\ (z{\mapsto}x) * (x{\mapsto}t),\quad (x{\mapsto}t) * \mathsf{true} * (\mathsf{ls}\,(z, 0))\ \}$$

Abstract semantics of atomic command A

$$
\begin{aligned}
(\!|A|\!) &: \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^{\top} \\
(\!|A|\!) &= \mathsf{Abs} \circ \mathsf{RuleApply}_A \circ \boxed{\mathsf{Rearr}_A}
\end{aligned}
$$

[Step1] Rearrange a symbolic heap, so that it pattern-matches the precond. of an adjusted proof rule.

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

$$\frac{}{\{\exists \vec{x'}.\, \Pi \land x{\mapsto}x' * \Sigma\} {*}x{=}t \{\exists \vec{x'}.\, \Pi \land x{\mapsto}t * \Sigma\}}$$

$\exists z'.$

$\{ \quad \exists z'.$ $\}$

$\{\ \exists z'.(x$

$\{$

$$
\begin{gathered}
\mathsf{ls}\ (x, 0) \\
\Downarrow \\
\top \\
\Downarrow \\
\top \\
\Downarrow \\
\top
\end{gathered}
$$

Adjusting a proof rule

$$\dfrac{\dfrac{\dfrac{}{\{E\mapsto E_0 * (\mathit{\Pi} \wedge \mathit{\Sigma})\}{*E{=}F}\{E\mapsto F * (\mathit{\Pi} \wedge \mathit{\Sigma})\}}\ \text{Update}}{\{\mathit{\Pi} \wedge (E\mapsto E_0 * \mathit{\Sigma})\}{*E{=}F}\{\mathit{\Pi} \wedge (E\mapsto F * \mathit{\Sigma})\}}\ \text{Conseq.}}{\{\exists \vec{x'}.\ \mathit{\Pi} \wedge (E\mapsto E_0 * \mathit{\Sigma})\}{*E{=}F}\{\exists \vec{x'}.\ \mathit{\Pi} \wedge (E\mapsto F * \mathit{\Sigma})\}}\ \text{Exist.}$$

- Make a rule work for symbolic heaps.
  - The precondition becomes a symbolic heap with the accessed cell exposed.
- Use Consequence and the below equivalence:

$$(E{=}F \wedge (\mathit{\Sigma}_0 * \mathit{\Sigma}_1)) \iff \mathit{\Sigma}_0 * (\mathit{\Sigma}_1 \wedge E{=}F)$$

$$(E{\neq}F \wedge (\mathit{\Sigma}_0 * \mathit{\Sigma}_1)) \iff \mathit{\Sigma}_0 * (\mathit{\Sigma}_1 \wedge E{\neq}F)$$

- Use the existential elimination rule.

Adjusting a proof rule

$$\dfrac{\dfrac{}{\{\Pi \wedge E{\mapsto}F * \Sigma\}x{=}{*}E\{\exists x'.\,x{=}E[x'/x] \wedge (\Pi \wedge E{\mapsto}F * \Sigma))[x'/x]\}}\ \text{Deref.}}{\{\exists\vec{y'}.\Pi \wedge E{\mapsto}F * \Sigma\}x{=}{*}E\{\exists\vec{y'}x'.\,x{=}E[x'/x] \wedge (\Pi \wedge E{\mapsto}F * \Sigma))[x'/x]\}}\ \text{Exist.}$$

- Make a rule work for symbolic heaps.
  - The precondition becomes a symbolic heap with the accessed cell exposed.
- Use Consequence and the below equivalence:

$$(E{=}F \wedge (\Sigma_0 * \Sigma_1)) \iff \Sigma_0 * (\Sigma_1 \wedge E{=}F)$$

$$(E{\neq}F \wedge (\Sigma_0 * \Sigma_1)) \iff \Sigma_0 * (\Sigma_1 \wedge E{\neq}F)$$

- Use the existential elimination rule.

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap})^\top$$

$$\frac{}{\{\exists \vec{x'}.\ \Pi \wedge (E{\mapsto}E_0 * \Sigma)\}{*}E{=}F\{\exists \vec{x'}.\ \Pi \wedge (E{\mapsto}F * \Sigma)\}}$$

- Transform a sym. heap so that it pattern-matches the precondition of the rule.
- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists \vec{x'}.\ \Pi \wedge \mathsf{ls}\,(E',F') * \Sigma \quad \rightsquigarrow_E \quad \{\ \exists \vec{x'}.\ \Pi \wedge E'{=}F' \wedge \Sigma, \quad \exists \vec{x'}y'.\ \Pi \wedge E{\mapsto}y' * \mathsf{ls}\,(y',F') * \Sigma\ \}$$

$$(\text{when } \Pi \Rightarrow E{=}E')$$

$$\exists \vec{x'}.\ \Pi \wedge E'{\mapsto}F' * \Sigma \quad \rightsquigarrow_E \quad \{\ \exists \vec{x'}.\ \Pi \wedge E{\mapsto}F' * \Sigma\ \}$$

$$(\text{when } \Pi \Rightarrow E{=}E')$$

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap})^{\top}$$

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

- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists \vec{x'}.\, \mathit{\Pi} \wedge \mathsf{ls}\,(E', F') * \mathit{\Sigma} \quad \rightsquigarrow_E \quad \{\, \exists \vec{x'}.\, \mathit{\Pi} \wedge E'{=}F' \wedge \mathit{\Sigma}, \quad \exists \vec{x'} y'.\, \mathit{\Pi} \wedge E{\mapsto}y' * \mathsf{ls}\,(y', F') * \mathit{\Sigma} \,\}$$
$$(\text{when } \mathit{\Pi} \Rightarrow E{=}E')$$

$$\exists \vec{x'}.\, \mathit{\Pi} \wedge E'{\mapsto}F' * \mathit{\Sigma} \quad \rightsquigarrow_E \quad \{\, \exists \vec{x'}.\, \mathit{\Pi} \wedge E{\mapsto}F' * \mathit{\Sigma} \,\}$$
$$(\text{when } \mathit{\Pi} \Rightarrow E{=}E')$$

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^{\top}$$

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

$$\Downarrow \mathsf{x}$$

$$\{\ \exists z'.\,\underline{x{=}z'} \wedge (z \mapsto z') * (z' \mapsto 0),\ \ \exists z'x'.\underline{(x \mapsto x') * \mathsf{ls}\,(x', z')} * (z \mapsto z') * (z' \mapsto 0)\ \}$$

- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists \vec{x'}.\ \Pi \wedge \mathsf{ls}\,(E', F') * \Sigma \ \ \rightsquigarrow_E\ \ \{\ \exists \vec{x'}.\ \Pi \wedge E'{=}F' \wedge \Sigma,\ \ \exists \vec{x'}y'.\ \Pi \wedge E \mapsto y' * \mathsf{ls}\,(y', F') * \Sigma\ \}$$
$$(\text{when } \Pi \Rightarrow E{=}E')$$

$$\exists \vec{x'}.\ \Pi \wedge E' \mapsto F' * \Sigma \ \ \rightsquigarrow_E\ \ \{\ \exists \vec{x'}.\ \Pi \wedge E \mapsto F' * \Sigma\ \}$$
$$(\text{when } \Pi \Rightarrow E{=}E')$$

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap})^{\top}$$

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

$$\Downarrow_{x}$$

$$\{\ \exists z'.\, x{=}z' \wedge (z \mapsto z') * \underline{(z' \mapsto 0)},\ \ \exists z' x'.(x \mapsto x') * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \ \}$$

$$\Downarrow_{x}$$

$$\{\ \exists z'.\, x{=}z' \wedge (z \mapsto z') * \underline{(x \mapsto 0)},\ \ \exists z' x'.(x \mapsto x') * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \ \}$$

- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists \vec{x'}.\, \Pi \wedge \mathsf{ls}\,(E', F') * \Sigma \ \ \rightsquigarrow_E\ \ \{\, \exists \vec{x'}.\, \Pi \wedge E'{=}F' \wedge \Sigma,\ \ \exists \vec{x'} y'.\, \Pi \wedge E \mapsto y' * \mathsf{ls}\,(y', F') * \Sigma \,\}$$
$$(\text{when } \Pi \Rightarrow E{=}E')$$

$$\exists \vec{x'}.\, \Pi \wedge E' \mapsto F' * \Sigma \ \ \rightsquigarrow_E\ \ \{\, \exists \vec{x'}.\, \Pi \wedge E \mapsto F' * \Sigma \,\}$$
$$(\text{when } \Pi \Rightarrow E{=}E')$$

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^{\top}$$

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

$$\Downarrow \mathsf{x}$$

$$\{\;\exists z'.\, x{=}z' \wedge (z\mapsto z') * (z'\mapsto 0),\;\; \exists z' x'.(x\mapsto x') * \mathsf{ls}\,(x', z') * (z\mapsto z') * (z'\mapsto 0)\;\}$$

$$\Downarrow \mathsf{x}$$

$$\{\;\exists z'.\, x{=}z' \wedge (z\mapsto z') * \underline{(x\mapsto 0),}\;\; \exists z' x'\underline{.(x\mapsto x')} * \mathsf{ls}\,(x', z') * (z\mapsto z') * (z'\mapsto 0)\;\}$$

- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists \vec{x}'.\, \Pi \wedge \mathsf{ls}\,(E', F') * \Sigma \;\rightsquigarrow_E\; \{\, \exists \vec{x}'.\, \Pi \wedge E'{=}F' \wedge \Sigma,\;\; \exists \vec{x}' y'.\, \Pi \wedge E\mapsto y' * \mathsf{ls}\,(y', F') * \Sigma \,\}$$

(when $\Pi \Rightarrow E{=}E'$)

$$\exists \vec{x}'.\, \Pi \wedge E'\mapsto F' * \Sigma \;\rightsquigarrow_E\; \{\, \exists \vec{x}'.\, \Pi \wedge E\mapsto F' * \Sigma \,\}$$

(when $\Pi \Rightarrow E{=}E'$)

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^\top$$

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

$$\Downarrow_{x}$$

$$\{\ \exists z'.\, x{=}z' \wedge (z \mapsto z') * (z' \mapsto 0),\ \ \exists z'x'.(x \mapsto x') * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \}$$

return

$$\Downarrow_{x}$$

$$\boxed{\{\ \exists z'.\, x{=}z' \wedge (z \mapsto z') * (x \mapsto 0),\ \ \exists z'x'.(x \mapsto x') * \mathsf{ls}\,(x', z') * (z \mapsto z') * (z' \mapsto 0)\ \}}$$

- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists \vec{x}'.\, \Pi \wedge \mathsf{ls}\,(E', F') * \Sigma \ \ \leadsto_E\ \ \{\ \exists \vec{x}'.\, \Pi \wedge E'{=}F' \wedge \Sigma,\ \ \exists \vec{x}'y'.\, \Pi \wedge E \mapsto y' * \mathsf{ls}\,(y', F') * \Sigma\ \}$$

$$(\text{when } \Pi \Rightarrow E{=}E')$$

$$\exists \vec{x}'.\, \Pi \wedge E' \mapsto F' * \Sigma \ \ \leadsto_E\ \ \{\ \exists \vec{x}'.\, \Pi \wedge E \mapsto F' * \Sigma\ \}$$

$$(\text{when } \Pi \Rightarrow E{=}E')$$

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^{\top}$$

$\mathsf{ls}(x, 0)$

⇩x

- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists \vec{x'}.\ \Pi \wedge \mathsf{ls}\,(E', F') * \Sigma \quad \rightsquigarrow_E \quad \{\ \exists \vec{x'}.\ \Pi \wedge E'{=}F' \wedge \Sigma, \quad \exists \vec{x'} y'.\ \Pi \wedge E{\mapsto}y' * \mathsf{ls}\,(y', F') * \Sigma\ \}$$

$$(\text{when } \Pi \Rightarrow E{=}E')$$

$$\exists \vec{x'}.\ \Pi \wedge E'{\mapsto}F' * \Sigma \quad \rightsquigarrow_E \quad \{\ \exists \vec{x'}.\ \Pi \wedge E{\mapsto}F' * \Sigma\ \}$$

$$(\text{when } \Pi \Rightarrow E{=}E')$$

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap})^{\top}$$

$$\mathsf{ls}(x,0)$$

$$\Downarrow x$$

$$\{\ \ x{=}0 \wedge \mathsf{emp},\ \ \ \exists x'.\,(x{\mapsto}x') * \mathsf{ls}\,(x',0)\ \ \}$$

- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists\vec{x'}.\ \Pi \wedge \mathsf{ls}\,(E',F') * \Sigma \quad \leadsto_E \quad \{\ \exists\vec{x'}.\ \Pi \wedge E'{=}F' \wedge \Sigma,\quad \exists\vec{x'}y'.\ \Pi \wedge E{\mapsto}y' * \mathsf{ls}\,(y',F') * \Sigma\ \}$$

$$(\text{when } \Pi \Rightarrow E{=}E')$$

$$\exists\vec{x'}.\ \Pi \wedge E'{\mapsto}F' * \Sigma \quad \leadsto_E \quad \{\ \exists\vec{x'}.\ \Pi \wedge E{\mapsto}F' * \Sigma\ \}$$

$$(\text{when } \Pi \Rightarrow E{=}E')$$

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^{\top}$$

$$\mathsf{ls}(x, 0)$$

$$\Downarrow_{\mathsf{x}}$$

$$\boxed{\{\ \ x{=}0 \wedge \mathsf{emp},}\ \exists x'.\,(x{\mapsto}x') * \mathsf{ls}\,(x', 0)\ \ \}$$

- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists \vec{x}'.\ \Pi \wedge \mathsf{ls}\,(E', F') * \Sigma \quad \rightsquigarrow_E \quad \{\ \exists \vec{x}'.\ \Pi \wedge E'{=}F' \wedge \Sigma, \quad \exists \vec{x}' y'.\ \Pi \wedge E{\mapsto}y' * \mathsf{ls}\,(y', F') * \Sigma\ \}$$
$$(\text{when } \Pi \Rightarrow E{=}E')$$

$$\exists \vec{x}'.\ \Pi \wedge E'{\mapsto}F' * \Sigma \quad \rightsquigarrow_E \quad \{\ \exists \vec{x}'.\ \Pi \wedge E{\mapsto}F' * \Sigma\ \}$$
$$(\text{when } \Pi \Rightarrow E{=}E')$$

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap})^{\top}$$

$$\mathsf{ls}(x,0)$$

$$\Downarrow \mathsf{x}$$

$$\{\ \ x{=}0 \wedge \mathsf{emp},\ \ \exists x'.\,(x{\mapsto}x') * \mathsf{ls}\,(x',0)\ \ \}$$

return $\top$

- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists\vec{x'}.\ \Pi \wedge \mathsf{ls}\,(E',F') * \Sigma \ \ \leadsto_E\ \ \{\ \exists\vec{x'}.\ \Pi \wedge E'{=}F' \wedge \Sigma,\ \ \ \exists\vec{x'}y'.\ \Pi \wedge E{\mapsto}y' * \mathsf{ls}\,(y',F') * \Sigma\ \}$$
$$(\text{when } \Pi \Rightarrow E{=}E')$$

$$\exists\vec{x'}.\ \Pi \wedge E'{\mapsto}F' * \Sigma \ \ \leadsto_E\ \ \{\ \exists\vec{x'}.\ \Pi \wedge E{\mapsto}F' * \Sigma\ \}$$
$$(\text{when } \Pi \Rightarrow E{=}E')$$

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \rightarrow \mathcal{P}(\mathsf{SymHeap})^{\top}$$

$$\frac{}{\{\exists \vec{x'}.\, \Pi \wedge (E{\mapsto}E_0 * \Sigma)\}{*}E{=}F\{\exists \vec{x'}.\, \Pi \wedge (E{\mapsto}F * \Sigma)\}}$$

- Transform a sym. heap so that it pattern-matches the preco
- Unro s) to expos

Allocatedness check:

1. Filter out inconsistent sym. heaps.
2. Check the existence of E↦F.

- Defined by rewriting rules and an allocatedness check.

$$\exists \vec{x'}.\, \Pi \wedge \mathsf{ls}\,(E',F') * \Sigma \quad \rightsquigarrow_E \quad \{\, \exists \vec{x'}.\, \Pi \wedge E'{=}F' \wedge \Sigma, \quad \exists \vec{x'}y'.\, \Pi \wedge E{\mapsto}y' * \mathsf{ls}\,(y',F') * \Sigma \,\}$$

$$(\text{when } \Pi \Rightarrow E{=}E')$$

$$\exists \vec{x'}.\, \Pi \wedge E'{\mapsto}F' * \Sigma \quad \rightsquigarrow_E \quad \{\, \exists \vec{x'}.\, \Pi \wedge E{\mapsto}F' * \Sigma \,\}$$

$$(\text{when } \Pi \Rightarrow E{=}E')$$

Rearr

$$\mathsf{Rearr}_A \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap})^{\top}$$

[Exercise 1] Compute the following.

(1) $\mathsf{Rearr}_{*x=1}(\mathsf{ls}(x,y) * \mathsf{ls}(y,z) * z \mapsto 0)$

(2) $\mathsf{Rearr}_{*x=1}(\mathsf{ls}(x,y) * \mathsf{ls}(y,z) * \mathsf{ls}(z,0))$

(3) $\mathsf{Rearr}_{*x=1}(z \neq 0 \wedge (\mathsf{ls}(x,y) * \mathsf{ls}(y,z) * \mathsf{ls}(z,0)))$

- Unroll inductively defined predicates (e.g., ls) to expose (E↦F) about accessed cell E.
- Defined by rewriting rules and an allocatedness check.

$$\exists \vec{x}'.\, \Pi \wedge \mathsf{ls}\,(E', F') * \Sigma \quad \rightsquigarrow_E \quad \{\, \exists \vec{x}'.\, \Pi \wedge E'{=}F' \wedge \Sigma, \quad \exists \vec{x}' y'.\, \Pi \wedge E{\mapsto}y' * \mathsf{ls}\,(y', F') * \Sigma \,\}$$
$$(\text{when } \Pi \Rightarrow E{=}E')$$

$$\exists \vec{x}'.\, \Pi \wedge E'{\mapsto}F' * \Sigma \quad \rightsquigarrow_E \quad \{\, \exists \vec{x}'.\, \Pi \wedge E{\mapsto}F' * \Sigma \,\}$$
$$(\text{when } \Pi \Rightarrow E{=}E')$$

Correctness of Rearr

Suppose $\mathrm{Rearr}_A(P) = \{Q_1, ..., Q_k\}$ and $A \equiv (*E=F)$.
[Question] Does the below statement hold?

1. $P \Longrightarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

Correctness of Rearr

Suppose $\mathrm{Rearr}_A(P) = \{Q_1, ..., Q_k\}$ and $A \equiv (*E=F)$.
[Question] Does the below statement hold?

$\boxed{\text{Yes}}$ 1. $P \Rightarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

Correctness of Rearr

Suppose $\mathrm{Rearr}_A(P) = \{Q_1, ..., Q_k\}$ and $A \equiv (*E=F)$.
[Question] Does the below statement hold?

Yes 1. $P \Longrightarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

2. $P \Longrightarrow Q_i$ for some $i \in \{1, ..., k\}$.

Correctness of Rearr

Suppose $\mathrm{Rearr}_A(P) = \{Q_1, ..., Q_k\}$ and $A \equiv (*E=F)$.

[Question] Does the below statement hold?

Yes 1. $P \Longrightarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

No 2. $P \Longrightarrow Q_i$ for some $i \in \{1, ..., k\}$.

Correctness of Rearr

Suppose $\mathrm{Rearr}_A(P) = \{Q_1, ..., Q_k\}$ and $A \equiv (*E=F)$.
[Question] Does the below statement hold?

$\boxed{\text{Yes}}$ 1. $P \Longrightarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

$\boxed{\text{No}}$ 2. $P \Longrightarrow Q_i$ for some $i \in \{1, ..., k\}$.

3. $P \Longleftarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

Correctness of Rearr

Suppose $\mathrm{Rearr}_A(P) = \{Q_1, ..., Q_k\}$ and $A \equiv (*E=F)$.

[Question] Does the below statement hold?

$\boxed{\text{Yes}}$ 1. $P \Rightarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

$\boxed{\text{No}}$ 2. $P \Rightarrow Q_i$ for some $i \in \{1, ..., k\}$.

$\boxed{\text{Yes}}$ 3. $P \Leftarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

Correctness of Rearr

Suppose $\mathrm{Rearr}_A(P) = \{Q_1, ..., Q_k\}$ and $A \equiv (*E=F)$.
[Question] Does the below statement hold?

Yes 1. $P \Rightarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

No 2. $P \Rightarrow Q_i$ for some $i \in \{1, ..., k\}$.

Yes 3. $P \Leftarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

4. $Q_i \Rightarrow E \mapsto \_ * \mathrm{true}$ for all $i \in \{1, ..., k\}$.

Correctness of Rearr

Suppose $\mathrm{Rearr}_A(P) = \{Q_1, \ldots, Q_k\}$ and $A \equiv (*E=F)$.

[Question] Does the below statement hold?

Yes 1. $P \Rightarrow Q_1 \vee Q_2 \vee \ldots \vee Q_k$.

No 2. $P \Rightarrow Q_i$ for some $i \in \{1, \ldots, k\}$.

Yes 3. $P \Leftarrow Q_1 \vee Q_2 \vee \ldots \vee Q_k$.

Yes 4. $Q_i \Rightarrow E \mapsto \_ * \mathrm{true}$ for all $i \in \{1, \ldots, k\}$.

Correctness of Rearr

Suppose $\mathrm{Rearr}_A(P) = \{Q_1, ..., Q_k\}$ and $A \equiv (*E{=}F)$.

[Question] Does the below statement hold?

Yes 1. $P \Rightarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

~~No 2. $P \Rightarrow Q_i$ for some $i \in \{1, ..., k\}$~~

Yes 3. $P \Leftarrow Q_1 \vee Q_2 \vee ... \vee Q_k$.

Yes 4. $Q_i \Rightarrow E \mapsto \_ * \mathrm{true}$ for all $i \in \{1, ..., k\}$.

Abs

$$\mathsf{Abs} \quad : \quad \mathcal{P}(\mathsf{SymHeap})^{\top} \to \mathcal{P}(\mathsf{SymHeap})^{\top}$$

- Forget the length of linked lists, and simplify quantifiers.
- Map $\top$ to $\top$.
- Defined by rewriting rules (true implications in sep. logic)

$$(\exists \vec{y'} x'.\, \Pi \wedge E \mapsto x' * x' \mapsto F * \Sigma) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{ls}\,(E, F) * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F))$$

$$(\exists \vec{y'} x'.\, \Pi \wedge \mathsf{ls}\,(E, x') * \mathsf{ls}\,(x', F) * \Sigma) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{ls}\,(E, F) * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F))$$

$$(\exists \vec{y'} x'.\, \Pi \wedge x' \mapsto F * \Sigma) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{true} * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma))$$

$$(\exists \vec{y'} x'.\, x'{=}E \wedge \Pi \wedge \Sigma) \;\rightsquigarrow\; (\exists \vec{y'}.\, (\Pi \wedge \Sigma)[E/x'])$$

Abs

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

$$\leadsto$$

$$(\exists \vec{y'} x'.\, \Pi \wedge E \mapsto x' * x' \mapsto F * \Sigma) \quad \leadsto \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{ls}\,(E, F) * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F))$$

$$(\exists \vec{y'} x'.\, \Pi \wedge \mathsf{ls}\,(E, x') * \mathsf{ls}\,(x', F) * \Sigma) \quad \leadsto \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{ls}\,(E, F) * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F))$$

$$(\exists \vec{y'} x'.\, \Pi \wedge x' \mapsto F * \Sigma) \quad \leadsto \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{true} * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma))$$

$$(\exists \vec{y'} x'.\, x' = E \wedge \Pi \wedge \Sigma) \quad \leadsto \quad (\exists \vec{y'}.\, (\Pi \wedge \Sigma)[E/x'])$$

Abs

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

$$\rightsquigarrow$$

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

$$\boxed{\begin{array}{rcl} (\exists \vec{y'} x'.\, \mathit{\Pi} \wedge E \mapsto x' * x' \mapsto F * \mathit{\Sigma}) & \rightsquigarrow & (\exists \vec{y'}.\, \mathit{\Pi} \wedge \mathsf{ls}\,(E, F) * \mathit{\Sigma}) \\ & & (\text{when } x' \notin \mathsf{FV}(\mathit{\Pi}, \mathit{\Sigma}, E, F)) \end{array}}$$

$$(\exists \vec{y'} x'.\, \mathit{\Pi} \wedge \mathsf{ls}\,(E, x') * \mathsf{ls}\,(x', F) * \mathit{\Sigma}) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \mathit{\Pi} \wedge \mathsf{ls}\,(E, F) * \mathit{\Sigma})$$
$$(\text{when } x' \notin \mathsf{FV}(\mathit{\Pi}, \mathit{\Sigma}, E, F))$$

$$(\exists \vec{y'} x'.\, \mathit{\Pi} \wedge x' \mapsto F * \mathit{\Sigma}) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \mathit{\Pi} \wedge \mathsf{true} * \mathit{\Sigma})$$
$$(\text{when } x' \notin \mathsf{FV}(\mathit{\Pi}, \mathit{\Sigma}))$$

$$(\exists \vec{y'} x'.\, x' = E \wedge \mathit{\Pi} \wedge \mathit{\Sigma}) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, (\mathit{\Pi} \wedge \mathit{\Sigma})[E/x'])$$

Abs

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

$$\rightsquigarrow$$

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

$$\rightsquigarrow$$

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

$$\begin{array}{rcl} (\exists \vec{y}' x'.\ \Pi \wedge E \mapsto x' * x' \mapsto F * \Sigma) & \rightsquigarrow & (\exists \vec{y}'.\ \Pi \wedge \mathsf{ls}\,(E, F) * \Sigma) \\ & & (\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F)) \end{array}$$

$$\boxed{\begin{array}{rcl} (\exists \vec{y}' x'.\ \Pi \wedge \mathsf{ls}\,(E, x') * \mathsf{ls}\,(x', F) * \Sigma) & \rightsquigarrow & (\exists \vec{y}'.\ \Pi \wedge \mathsf{ls}\,(E, F) * \Sigma) \\ & & (\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F)) \end{array}}$$

$$\begin{array}{rcl} (\exists \vec{y}' x'.\ \Pi \wedge x' \mapsto F * \Sigma) & \rightsquigarrow & (\exists \vec{y}'.\ \Pi \wedge \mathsf{true} * \Sigma) \\ & & (\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma)) \end{array}$$

$$(\exists \vec{y}' x'.\ x' = E \wedge \Pi \wedge \Sigma) \ \rightsquigarrow\ (\exists \vec{y}'.\ (\Pi \wedge \Sigma)[E/x'])$$

Abs

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

$$\rightsquigarrow$$

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

$$\rightsquigarrow$$

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

$$\rightsquigarrow$$

$$\mathsf{ls}\,(x, 0) * \underline{\mathsf{true}}$$

$$(\exists \vec{y'} x'.\, \Pi \wedge E \mapsto x' * x' \mapsto F * \Sigma) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{ls}\,(E, F) * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F))$$

$$(\exists \vec{y'} x'.\, \Pi \wedge \mathsf{ls}\,(E, x') * \mathsf{ls}\,(x', F) * \Sigma) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{ls}\,(E, F) * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F))$$

$$(\exists \vec{y'} x'.\, \Pi \wedge x' \mapsto F * \Sigma) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{true} * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma))$$

$$(\exists \vec{y'} x'.\, x' = E \wedge \Pi \wedge \Sigma) \ \rightsquigarrow \ (\exists \vec{y'}.\, (\Pi \wedge \Sigma)[E/x'])$$

Abs

$$\mathsf{Abs} \quad : \quad \mathcal{P}(\mathsf{SymHeap})^{\top} \to \mathcal{P}(\mathsf{SymHeap})^{\top}$$

- Forget the length of linked lists, and simplify quantifiers.
- Map $\top$ to $\top$.
- Defined by rewriting rules (true implications in sep. logic)

$$(\exists \vec{y'} x'.\, \Pi \wedge E \mapsto x' * x' \mapsto F * \Sigma) \quad \leadsto \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{ls}\,(E, F) * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F))$$

$$(\exists \vec{y'} x'.\, \Pi \wedge \mathsf{ls}\,(E, x') * \mathsf{ls}\,(x', F) * \Sigma) \quad \leadsto \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{ls}\,(E, F) * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F))$$

$$(\exists \vec{y'} x'.\, \Pi \wedge x' \mapsto F * \Sigma) \quad \leadsto \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{true} * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma))$$

$$(\exists \vec{y'} x'.\, x' {=} E \wedge \Pi \wedge \Sigma) \;\leadsto\; (\exists \vec{y'}.\, (\Pi \wedge \Sigma)[E/x'])$$

Abs

[Exercise 2] Compute the following:

$\mathsf{Abs}(\{\ \exists abcd.\ c{=}d \ \wedge\ x{\mapsto}a * a{\mapsto}b * y{\mapsto}b * \mathsf{ls}(b,c) * \mathsf{ls}(d,0)\ \})$

$\mathsf{Abs}(\{\ \exists abcde.\ \mathsf{ls}(x,a) * \mathsf{ls}(a,0) * b{\mapsto}d * c{\mapsto}e\ \})$

$\mathsf{Abs}(\{\ \exists abcde.\ d{=}c \ \wedge\ x{\mapsto}a * \mathsf{ls}(a,0) * b{\mapsto}d * c{\mapsto}e\ \})$

[Exercise 3] Add other sensible rewriting rules.

c)

$$(\exists \vec{y'} x'.\, \Pi \wedge E{\mapsto}x' * x'{\mapsto}F * \Sigma) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{ls}\,(E,F) * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F))$$

$$(\exists \vec{y'} x'.\, \Pi \wedge \mathsf{ls}\,(E,x') * \mathsf{ls}\,(x',F) * \Sigma) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{ls}\,(E,F) * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma, E, F))$$

$$(\exists \vec{y'} x'.\, \Pi \wedge x'{\mapsto}F * \Sigma) \quad \rightsquigarrow \quad (\exists \vec{y'}.\, \Pi \wedge \mathsf{true} * \Sigma)$$
$$(\text{when } x' \notin \mathsf{FV}(\Pi, \Sigma))$$

$$(\exists \vec{y'} x'.\, x'{=}E \wedge \Pi \wedge \Sigma) \ \rightsquigarrow \ (\exists \vec{y'}.\, (\Pi \wedge \Sigma)[E/x'])$$

Correctness of Abs

Suppose $Abs(\{P_1, ..., P_n\}) = \{Q_1, ..., Q_m\}$. Then, the below implication holds.

$$(P_1 \vee P_2 \vee ... \vee P_n) \Longrightarrow (Q_1 \vee Q_2 \vee ... \vee Q_m).$$

[Question] What about the other $\Longleftarrow$ direction?