Program Verification using Separation Logic

Lecture 2: Simple Automatic Verifier

Hongseok Yang (Queen Mary, Univ. of London)

What's missing to build an automatic verifier?

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

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

$y = 0;$

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

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

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

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

$t = x;$

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

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

$x = *t;$

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

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

$*t = y;$

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

$y = t$

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

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

$\}$

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

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

What's missing to build an automatic verifier?

1. Infer a loop invariant.

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

$y = 0;$

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

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

$\boxed{\begin{array}{l} \{x{\neq}0 \wedge \mathsf{ls}\ (y,0) * \mathsf{ls}\ (x,0)\} \\ \{\exists x'.\, \mathsf{ls}\ (y,0) * (x \mapsto x') * \mathsf{ls}\ (x',0)\} \end{array}}$

$t = x;$

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

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

$x = *t;$

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

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

$*t = y;$

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

$y = t$

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

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

$\}$

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

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

What's missing to build an automatic verifier?

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

a) exposing pointsto.

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

$y = 0;$

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

`while` $(x \neq 0)$ $\{$

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

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

$t = x;$

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

$x = *t;$

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

$*t = y;$

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

$y = t$

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

$\}$

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

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

What's missing to build an automatic verifier?

1. Infer a loop invariant.

2. Massage assertions using Consequence.

   a) exposing pointsto.

   b) removing equality.

$\{\mathsf{ls}\,(x,0)\}$
$y = 0;$
$\mathsf{INV}:\ \mathsf{ls}\,(y,0) * \mathsf{ls}\,(x,0)$
`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.

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

$y = 0;$

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

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

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

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

$t = x;$

$\{\exists x'.\ t{=}x$ … $)\}$

$\{\exists x'.\ \mathsf{ls}\ (y$ …

$x = *t;$

$\{\exists x'.\ x{=}x$ … $0)\}$

$\{\mathsf{ls}\ (y,0) *$ …

$*t = y;$

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

$y = t$

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

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

$\}$

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

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

What's missing to build an automatic verifier?

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.

Today's Lecture

- Understand the abstraction and rearrangement.
- Should be able to build a basic program verifier.
- Will use the term "static analysis", instead of "automatic program verifier".

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 sep. implication and no univ. quan.
- Built-in list-seg predicate without alpha.
- E.g. $y = z \wedge \mathsf{ls}\,(x, 0) * \mathsf{ls}\,(y, 0)$
  $\exists v' w'.\, \mathsf{ls}\,(x, v') * y \mapsto v' * v' \mapsto w'$

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

Rearrangement +
Proof rule in sep. logic +
Abstraction

- Algorithm:
  - Abstractly run a program with sym. heaps.
  - Accumulate all the obtained sym. heaps.
  - Repeat until no changes.

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

```
h = 0;
```

$\boxed{emp}$

$\boxed{h=0 \wedge emp}$

```
while (nondet)
{
```

$\boxed{h=0 \wedge emp}$

```
    t = new(1);
```

$\boxed{\exists t'.\, h=0 \wedge t \mapsto t'}$

```
    *t=h;
```

$\boxed{h=0 \wedge t \mapsto 0}$

```
    h=t;
```

$\boxed{\exists h'.\, h=t \wedge h'=0 \wedge t \mapsto h'}$

```
    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;

}
```

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’. 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=t ∧ t↦0
   t=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

create

```
h = 0;
```

$\boxed{emp}$

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

```
while (nondet)
{
```

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

```
  t = new(1);
```

$\boxed{\exists t'.\, h=0 \wedge t \mapsto t'}$

```
  *t=h;
```

$\boxed{h=0 \wedge t \mapsto 0}$

```
  h=t;
```

$\boxed{h=t \wedge t \mapsto 0}$

```
  t=0;
```

$\boxed{t=0 \wedge h \mapsto 0}$

```
}
```

create

emp

```
h = 0;
```

$h{=}0 \land emp$ $\quad$ $t{=}0 \land h{\mapsto}0$

```
while (nondet)
{
```

$h{=}0 \land emp$ $\quad$ $t{=}0 \land h{\mapsto}0$

```
  t = new(1);
```

$\exists t'.\ h{=}0 \land t{\mapsto}t'$ $\quad$ $\exists t't''.\ t'{=}0 \land t{\mapsto}t'' * h{\mapsto}0$

```
  *t=h;
```

$h{=}0 \land t{\mapsto}0$

```
  h=t;
```

$h{=}t \land t{\mapsto}0$

```
  t=0;
```

$t{=}0 \land h{\mapsto}0$

```
}
```

create

```
h = 0;
while (nondet)
{
  t = new(1);
  *t=h;
  h=t;
  t=0;
}
```

$\boxed{\mathrm{emp}}$

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

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

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

$\boxed{h=0 \wedge t \mapsto 0}$

$\boxed{h=t \wedge t \mapsto 0}$

$\boxed{t=0 \wedge h \mapsto 0}$

create

```
h = 0;
while (nondet)
{
   t = new(1);
   *t=h;
   h=t;
   t=0;
}
```

$\boxed{emp}$

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

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

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

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

$\boxed{h=t \wedge t\mapsto 0}$

$\boxed{t=0 \wedge h\mapsto 0}$

create

emp

```
h = 0;
```

$h=0 \land emp$ $\quad$ $t=0 \land h \mapsto 0$

```
while (nondet)
{
```

$h=0 \land emp$ $\quad$ $t=0 \land h \mapsto 0$

```
    t = new(1);
```

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

```
    *t=h;
```

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

```
    h=t;
```

$h=t \land t \mapsto 0$

```
    t=0;
```

$t=0 \land h \mapsto 0$

```
}
```

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;
```

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

```
  t=0;
```

$t=0 \wedge h \mapsto 0$

```
}
```

create

emp

```
h = 0;
```

$h=0 \wedge emp$ | $t=0 \wedge h \mapsto 0$

```
while (nondet)
{
```

$h=0 \wedge emp$ | $t=0 \wedge h \mapsto 0$

```
  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;
```

$h=t \wedge t \mapsto 0$ | $\exists h'. h=t \wedge ls\ (t,0)$

```
  t=0;
```

$t=0 \wedge h \mapsto 0$

```
}
```

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;
```

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

```
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;
```

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

```
}
```

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;
```

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

```
}
```

create

emp

```
h = 0;
```

$h=0 \wedge emp$ | $t=0 \wedge h \mapsto 0$ | $t=0 \wedge ls(h,0)$

```
while (nondet)
{
```

$h=0 \wedge emp$ | $t=0 \wedge h \mapsto 0$

```
  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;
```

$h=t \wedge t \mapsto 0$ | $h=t \wedge ls(t,0)$

```
  t=0;
```

$t=0 \wedge h \mapsto 0$ | $t=0 \wedge ls(h,0)$

```
}
```

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;
```

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

```
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;
```

$h=t \wedge t \mapsto 0$ | $h=t \wedge ls(t,0)$

```
  t=0;
```

$t=0 \wedge h \mapsto 0$ | $t=0 \wedge ls(h,0)$ | $t=0 \wedge ls(h,0)$

```
}
```

create

emp

h = 0;

$h=0 \wedge emp$ | $t=0 \wedge h \mapsto 0$ | $t=0 \wedge ls(h,0)$

while (nondet)
{

$h=0 \wedge emp$ | $t=0 \wedge h \mapsto 0$

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;

$h=t \wedge t \mapsto 0$ | $h=t \wedge ls(t,0)$

t=0;

$t=0 \wedge h \mapsto 0$ | $t=0 \wedge ls(h,0)$ | $t=0 \wedge ls(h,0)$

}

$h=0 \wedge emp$ | $t=0 \wedge h \mapsto 0$ | $t=0 \wedge ls(h,0)$

create

emp

```
h = 0;
```

$h=0 \land emp$ | $t=0 \land h \mapsto 0$ | $t=0 \land ls(h,0)$

```
while (nondet)
{
```

$h=0 \land emp$ | $t=0 \land h \mapsto 0$

```
  t = new(1);
```

```
  h=t;
```

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

```
  t=0;
```

$\exists t'. t=0 \land h=t' \land t' \mapsto 0$ | $\exists t'. t=0 \land h=t' \land ls(t',0)$ | $t=0 \land ls(h,0)$

```
}
```

$h=0 \land emp$ | $t=0 \land h \mapsto 0$ | $t=0 \land ls(h,0)$

The output is a proof sketch of

$\{ emp \}create\{ (h=0 \land emp) \lor (t=0 \land h \mapsto 0) \lor (t=0 \land ls(h,0)) \}$

Abstract Semantics of Atomic Command A

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

Abstract Semantics of Atomic Command A

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

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

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

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

omic Command A

SymHeap) $\cup$ {err}

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

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

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

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

omic Command A

SymHeap) $\cup$ {err}

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

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

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

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

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

omic Command A

SymHeap) $\cup$ {err}

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

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

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

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

$$\Downarrow$$

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

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

omic Command A

SymHeap) $\cup$ {err}

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

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

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

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

$$\Downarrow$$

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

Abstract Semantics of Atomic Command A

$$(\!|A|\!) \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap}) \cup \{\mathsf{err}\}$$
$$(\!|A|\!) \quad = \quad \mathsf{Abs} \circ \boxed{\mathsf{RuleApply}_A} \circ \mathsf{Rearr}_A$$

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

[Step2] Apply the rule.

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

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

$$\Downarrow$$

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

Abstract Semantics of Atomic Command A

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

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

[Step2] Apply the rule.

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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

Abstract Semantics of Atomic Command A

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

[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.

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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

Abstract Semantics of Atomic Command A

$$(\!|A|\!) \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap}) \cup \{\mathsf{err}\}$$
$$(\!|A|\!) \quad = \quad \boxed{\mathsf{Abs}} \circ \mathsf{RuleApply}_A \circ \mathsf{Rearr}_A$$

[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.

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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

$$\Downarrow$$

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

Abstract Semantics of Atomic Command A

$$\llparenthesis A \rrparenthesis \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap}) \cup \{\mathsf{err}\}$$
$$\llparenthesis A \rrparenthesis \quad = \quad \boxed{\mathsf{Abs}} \circ \mathsf{RuleApply}_A \circ \mathsf{Rearr}_A$$

[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.

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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

$$\Downarrow$$

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

Abstract Semantics of Atomic Command A

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

[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.

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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

$$\Downarrow$$

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

Abstract Semantics of Atomic Command A

$$\begin{array}{rcl} (\!|A|\!) & : & \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap}) \cup \{\mathsf{err}\} \\ (\!|A|\!) & = & \boxed{\mathsf{Abs}} \circ \mathsf{RuleApply}_A \circ \mathsf{Rearr}_A \end{array}$$

[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.

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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

$$\Downarrow$$

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

Abstract Semantics of Atomic Command A

$$(\!|A|\!) \quad : \quad \mathsf{SymHeap} \to \mathcal{P}(\mathsf{SymHeap}) \cup \{\mathsf{err}\}$$
$$(\!|A|\!) \quad = \quad \boxed{\mathsf{Abs}} \circ \mathsf{RuleApply}_A \circ \mathsf{Rearr}_A$$

[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.

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

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

$$\Downarrow$$

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

$$\Downarrow$$

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

$$\Downarrow$$

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

Abstract Semantics of Atomic Command A

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

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

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

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

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

ls $(x, 0)$

$\Downarrow$

err

$\Downarrow$

err

$\Downarrow$

err

Abstract Semantics of Atomic Command A

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

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

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

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

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

$\exists z'.$

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

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

$\{$ $\}$

ls $(x, 0)$

$\Downarrow$

err

$\Downarrow$

err

$\Downarrow$

err

Abstract Semantics of Atomic Command A

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

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

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

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

ls $(x, 0)$

$\Downarrow$

err

$\Downarrow$

err

$\Downarrow$

err

Abstract Semantics of Atomic Command A

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

[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.

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

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

$\Downarrow$

err

$\Downarrow$

err

$\Downarrow$

err

Abstract Semantics of Atomic Command A

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

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

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

[Step2] Apply the rule.

[Step3] Abstract symbolic heaps.

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

$\exists z'.$

$\{ \quad \exists z'.$

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

$\{$

ls $(x, 0)$

err

err

err

Reference

- Distefano et al's TACAS 2006 paper.
- Berdine et al's APLAS 2005 paper.