Fractional Resources in Unbounded Separation Logic

Thibault Dardinier, Peter Müller, and Alexander J. Summers

Separation logic

```
method caller() {

  a := new ObjectF(5)

  b := new ObjectF(7)

  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

```
method callee(b: Ref)


{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5

  b := new ObjectF(7)

  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

```
method callee(b: Ref)


{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5

  b := new ObjectF(7) // b.f = 7

  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

```
method callee(b: Ref)


{
   ... // reads b.f
}
```

Separation logic

```
method caller() {

   a := new ObjectF(5) // a.f = 5

   b := new ObjectF(7) // b.f = 7

   callee(b)

   assert a.f == 5
   assert b.f == 7
}
```

```
method callee(b: Ref)


{
   ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  a.f
  b := new ObjectF(7) // b.f = 7

  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

```
method callee(b: Ref)


{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5

  b := new ObjectF(7) // b.f = 7

  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

a.f

Full (exclusive) permission
Permits to read **and write** *a.f*

$a.f \mapsto \_$

```
method callee(b: Ref)

{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  b := new ObjectF(7) // b.f = 7
  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

$a.f \mapsto \_$

Full (exclusive) permission
Permits to read **and write** *a.f*

a.f

a.f b.f

```
method callee(b: Ref)


{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  [a.f]
  b := new ObjectF(7) // b.f = 7
  [a.f] [b.f]
  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

a.f ↦ _

Full (exclusive) permission
Permits to read **and write** *a.f*

```
method callee(b: Ref)
  requires [b.f]
  ensures  [b.f]
{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  [a.f]
  b := new ObjectF(7) // b.f = 7
  [a.f] [b.f]
  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

$a.f \mapsto \_$

Full (exclusive) permission
Permits to read **and write** *a.f*

$$\frac{\{P\}\ C\ \{Q\} \qquad mod(C) \cap fv(R) = \emptyset}{\{P * R\}\ C\ \{Q * R\}}\ (Frame)$$

```
method callee(b: Ref)
  requires [b.f]
  ensures  [b.f]
{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  [a.f]
  b := new ObjectF(7) // b.f = 7
  [a.f] [b.f]
  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

Full (exclusive) permission
Permits to read **and write** *a.f*

$$\text{a.f} \mapsto \_$$

$$\frac{\{P\}\ C\ \{Q\} \qquad mod(C) \cap fv(R) = \emptyset}{\{P * R\}\ C\ \{Q * R\}}\ (Frame)$$

```
method callee(b: Ref)
  requires [b.f]
  ensures  [b.f]
{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  [a.f]
  b := new ObjectF(7) // b.f = 7
  [a.f] [b.f]
  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

a.f $\mapsto$ _

Full (exclusive) permission
Permits to read **and write** *a.f*

$$\frac{\{P\}\ C\ \{Q\} \qquad mod(C) \cap fv(R) = \emptyset}{\{P * R\}\ C\ \{Q * R\}}\ (Frame)$$

Disjoint permissions

```
method callee(b: Ref)
  requires [b.f]
  ensures [b.f]
{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5
```

a.f

a.f ↦ _

Full (exclusive) permission
Permits to read **and write** *a.f*

```
  b := new ObjectF(7) // b.f = 7
```

a.f b.f

```
  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

$$\frac{\{P\}\ C\ \{Q\} \qquad mod(C) \cap fv(R) = \emptyset}{\{P * R\}\ C\ \{Q * R\}}\ (\mathit{Frame})$$

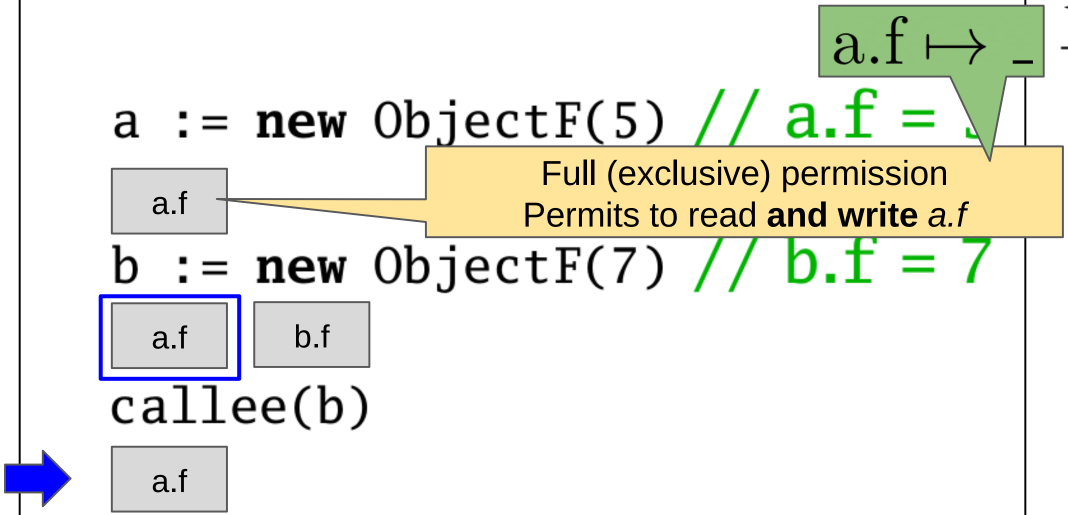
Disjoint permissions

```
method callee(b: Ref)
  requires b.f
  ensures b.f
{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  b := new ObjectF(7) // b.f = 7
  callee(b)
  assert a.f == 5
  assert b.f == 7
}
```



$$\frac{\{P\}\ C\ \{Q\} \qquad mod(C) \cap fv(R) = \varnothing}{\{P * R\}\ C\ \{Q * R\}}\ (Frame)$$

Disjoint permissions

```
method callee(b: Ref)
  requires b.f
  ensures b.f
{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

   a := new ObjectF(5) // a.f = 5
   [a.f]
   b := new ObjectF(7) // b.f = 7
   [a.f] [b.f]
   callee(b)
   [a.f]
   assert a.f == 5
   assert b.f == 7
}
```

$a.f \mapsto \_$

Full (exclusive) permission
Permits to read **and write** *a.f*

$$\frac{\{P\}\ C\ \{Q\} \qquad mod(C) \cap fv(R) = \varnothing}{\{P * R\}\ C\ \{Q * R\}}\ (Frame)$$

Disjoint permissions

```
method callee(b: Ref)
   requires [b.f]
   ensures [b.f]
{
   ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  [a.f]
  b := new ObjectF(7) // b.f = 7
  [a.f] [b.f]
  callee(b)
  [a.f] [b.f]
  assert a.f == 5
  assert b.f == 7
}
```

$a.f \mapsto \_$

Full (exclusive) permission
Permits to read **and write** *a.f*

$$\frac{\{P\}\ C\ \{Q\} \qquad mod(C) \cap fv(R) = \emptyset}{\{P * R\}\ C\ \{Q * R\}}\ (Frame)$$

Disjoint permissions

```
method callee(b: Ref)
  requires [b.f]
  ensures [b.f]
{
  ... // reads b.f
}
```

Separation logic

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  [a.f]
  b := new ObjectF(7) // b.f = 7
  [a.f] [b.f]
  callee(b)
  [a.f] [b.f]
  assert a.f == 5
  assert b.f == 7
}
```

$a.f \mapsto \_$

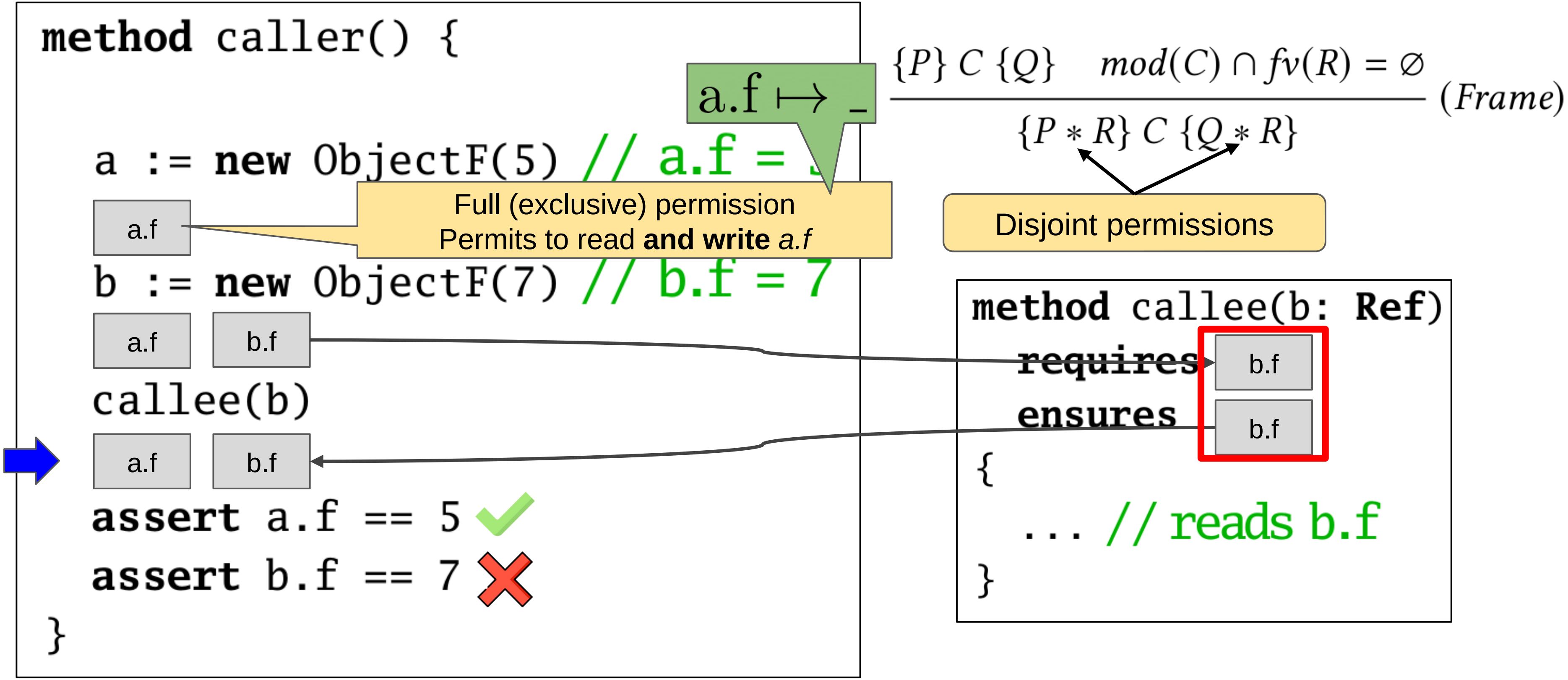
Full (exclusive) permission
Permits to read **and write** *a.f*

$$\frac{\{P\}\ C\ \{Q\} \qquad mod(C) \cap fv(R) = \emptyset}{\{P * R\}\ C\ \{Q * R\}}\ (Frame)$$

Disjoint permissions

```
method callee(b: Ref)
  requires [b.f]
  ensures [b.f]
{
  ... // reads b.f
}
```

Separation logic



Fractional permissions

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  a.f
  b := new ObjectF(7) // b.f = 7
  a.f   b.f
  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

```
method callee(b: Ref)
  requires
  ensures
{
  ... // reads b.f
}
```

Fractional permissions

```
method caller() {

   a := new ObjectF(5) // a.f = 5
   [a.f]
   b := new ObjectF(7) // b.f = 7
   [a.f] [b.f]
   callee(b)

   assert a.f == 5 ✔
   assert b.f == 7 ✘
}
```

```
method callee(b: Ref)
   requires [b.f]
   ensures [b.f]
{
   ... // reads .f
}
```

$b.f \overset{0.5}{\mapsto} \_$

Fractional (non-exclusive) permission
Permits only to read *b.f*

Fractional permissions

```
method caller() {

  a := new ObjectF(5) // a.f = 5
```

a.f

```
  b := new ObjectF(7) // b.f = 7
```

a.f b.f b.f

```
  callee(b)

  assert a.f == 5
  assert b.f == 7
}
```

```
method callee(b: Ref)
  requires
  ensures
{
  ... // reads .f
}
```

requires: b.f

ensures: b.f — $b.f \overset{0.5}{\mapsto} \_$

Fractional (non-exclusive) permission
Permits only to read *b.f*

Fractional permissions

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  [a.f]
  b := new ObjectF(7) // b.f = 7
  [a.f] [b.f | b.f]
  callee(b)
  [a.f] [b.f]
  assert a.f == 5  ✔
  assert b.f == 7  ✘
}
```

```
method callee(b: Ref)
  requires [b.f]
  ensures [b.f]
{
  ... // reads b.f
}
```

$b.f \overset{0.5}{\mapsto} \_$

Fractional (non-exclusive) permission
Permits only to read *b.f*

Fractional permissions

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  b := new ObjectF(7) // b.f = 7
  callee(b)
  assert a.f == 5
  assert b.f == 7
}
```

```
method callee(b: Ref)
  requires b.f
  ensures b.f
{
  ... // reads b.f
}
```

$b.f \overset{0.5}{\mapsto} \_$

Fractional (non-exclusive) permission
Permits only to read *b.f*

Fractional permissions

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  [a.f]
  b := new ObjectF(7) // b.f = 7
  [a.f] [b.f | b.f]
  callee(b)
  [a.f] [b.f | b.f]
  assert a.f == 5 ✔
  assert b.f == 7 ✔
}
```

```
method callee(b: Ref)
  requires [b.f]
  ensures [b.f]
{
  ... // reads b.f
}
```

$b.f \overset{0.5}{\mapsto} \_$

Fractional (non-exclusive) permission
Permits only to read *b.f*

Fractional permissions

```
method caller() {

  a := new ObjectF(5) // a.f = 5
  [a.f]
  b := new ObjectF(7) // b.f = 7
  [a.f] [b.f | b.f]
  callee(b)
  [a.f] [b.f | b.f]
  assert a.f == 5 ✓
  assert b.f == 7 ✓
}
```

$$\text{State} \triangleq \text{Locations} \rightharpoonup \text{Values} \times (\mathbb{Q} \cap (0,1])$$

```
method callee(b: Ref)
  requires b.f
  ensures b.f
{
  ... // reads .f
}
```

$b.f \overset{0.5}{\mapsto} \_$

Fractional (non-exclusive) permission
Permits only to read *b.f*

Fractional permissions

```
method caller() {

   a := new ObjectF(5) // a.f = 5
   [a.f]
   b := new ObjectF(7) // b.f = 7
   [a.f] [b.f | b.f]
   callee(b)
   [a.f] [b.f | b.f]
   assert a.f == 5 ✔
   assert b.f == 7 ✔
}
```

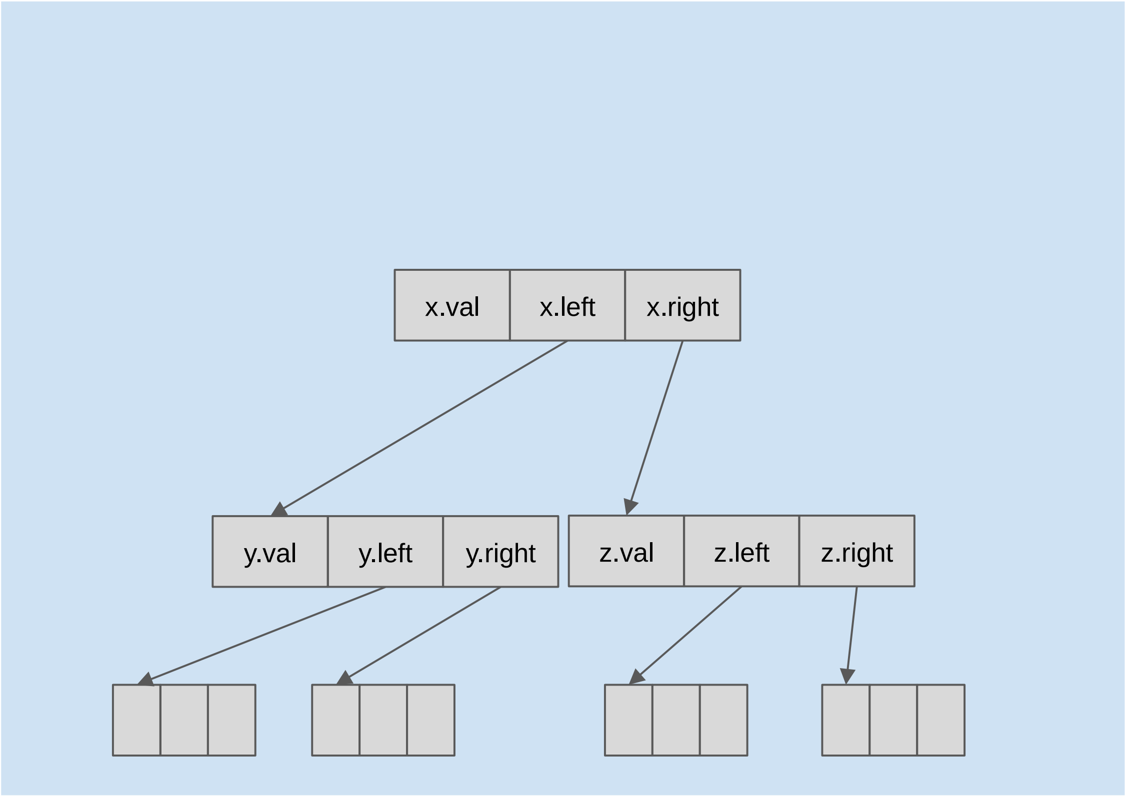
```
method callee(b: Ref)
   requires [b.f]
   ensures [b.f]
{
   ... // reads .f
}
```

$b.f \overset{0.5}{\mapsto} \_$

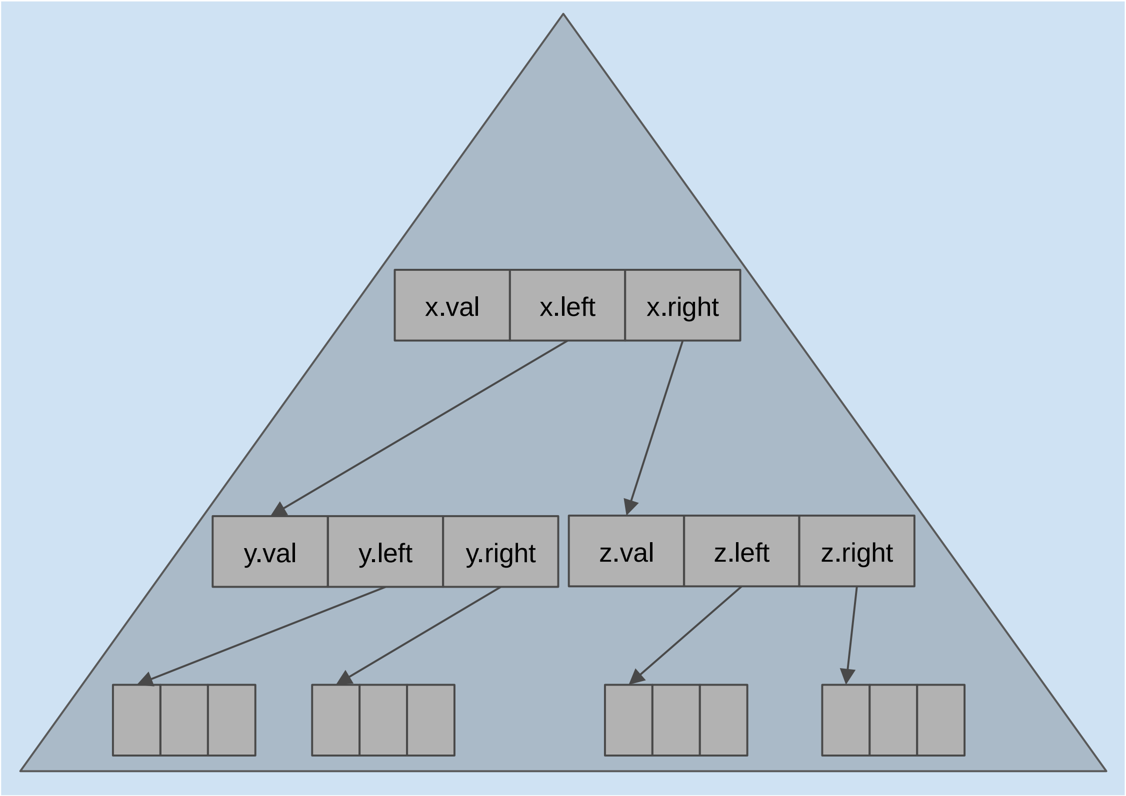
Fractional (non-exclusive) permission
Permits only to read *b.f*

$$\text{State} \triangleq \text{Locations} \rightharpoonup \text{Values} \times (\mathbb{Q} \cap (0,1])$$

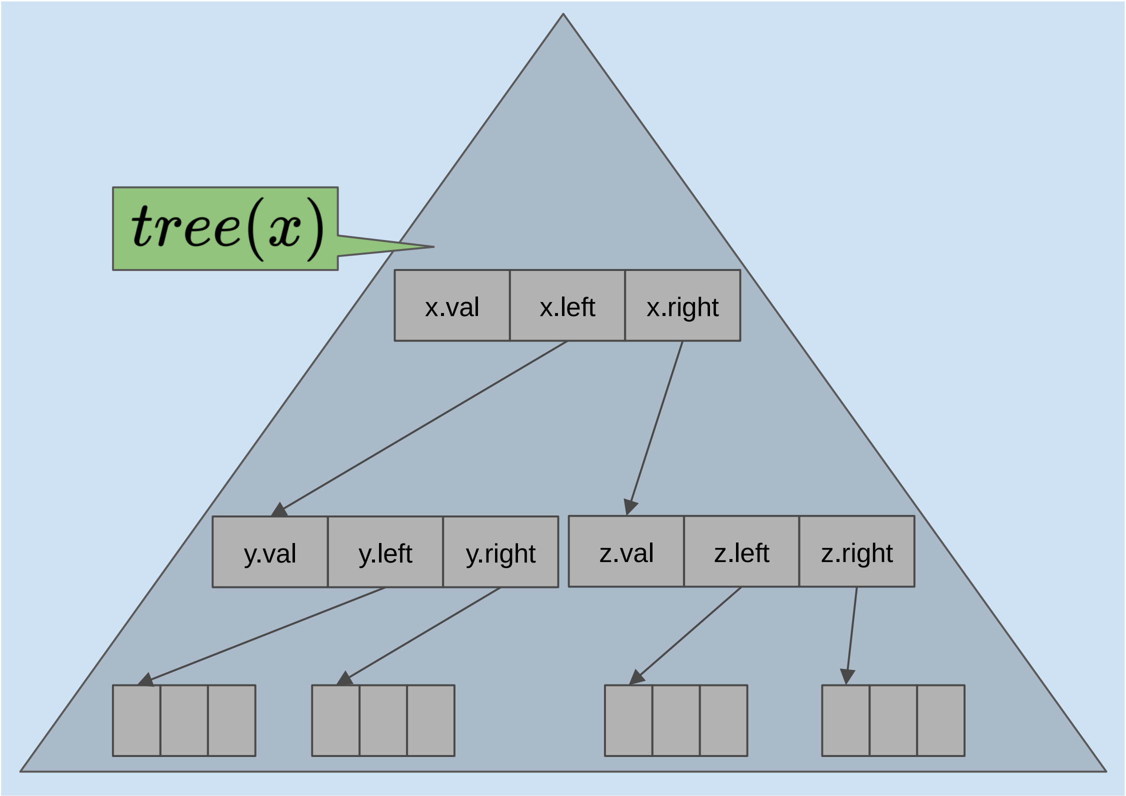
(Fractional) resources, informally



(Fractional) resources, informally

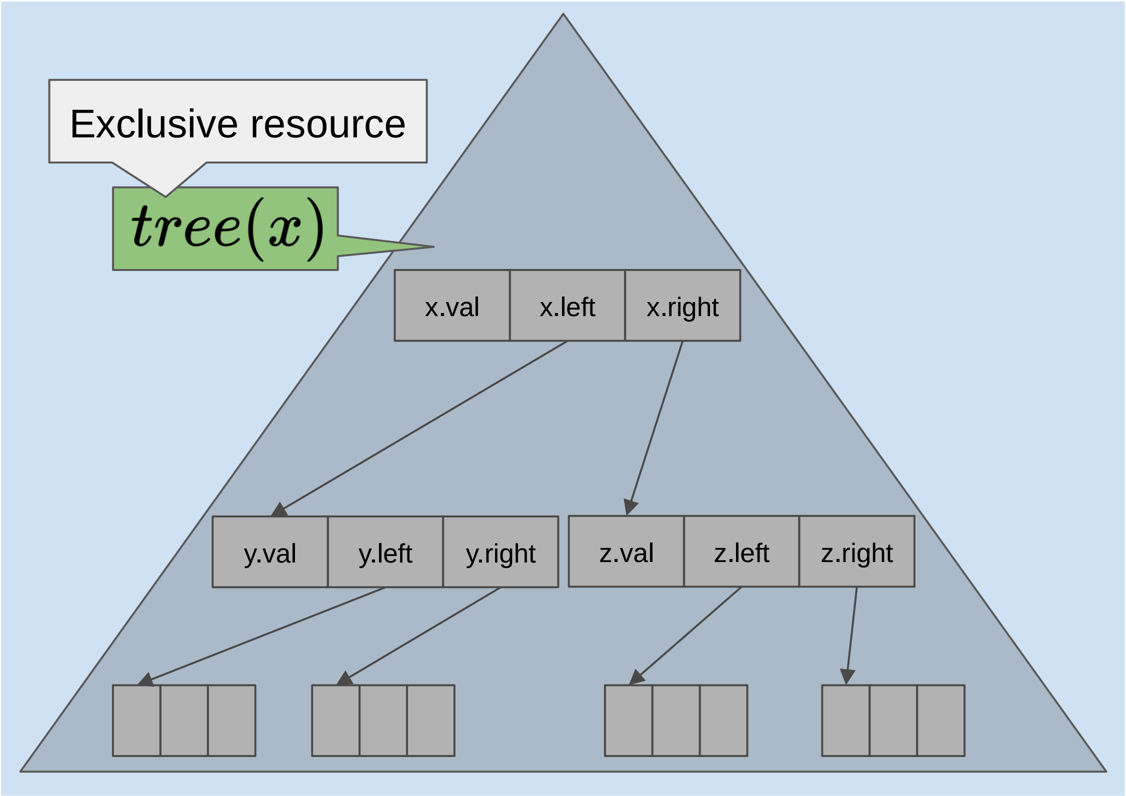


(Fractional) resources, informally



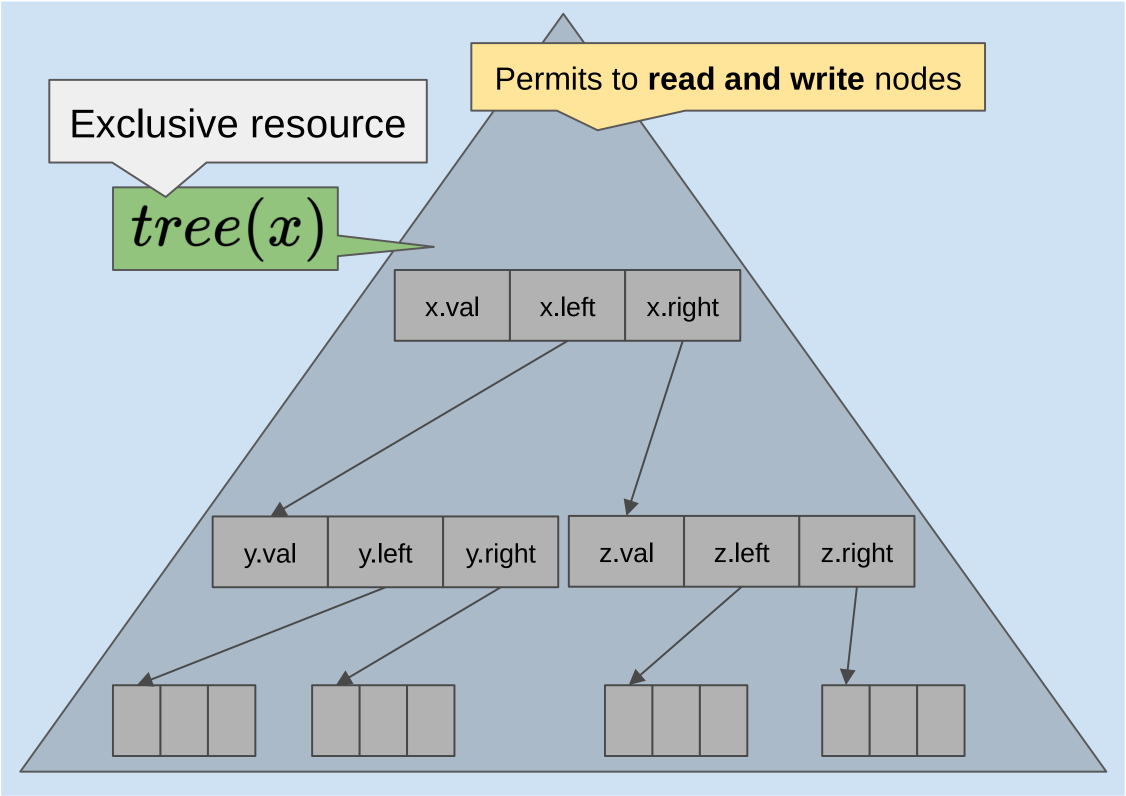
$$tree(x) \triangleq (x \neq \texttt{null} \Rightarrow x.val \mapsto \_ *$$
$$(\exists x_l.\, x.left \mapsto x_l * tree(x_l)) * (\exists x_r.\, x.right \mapsto x_r * tree(x_r)))$$

(Fractional) resources, informally



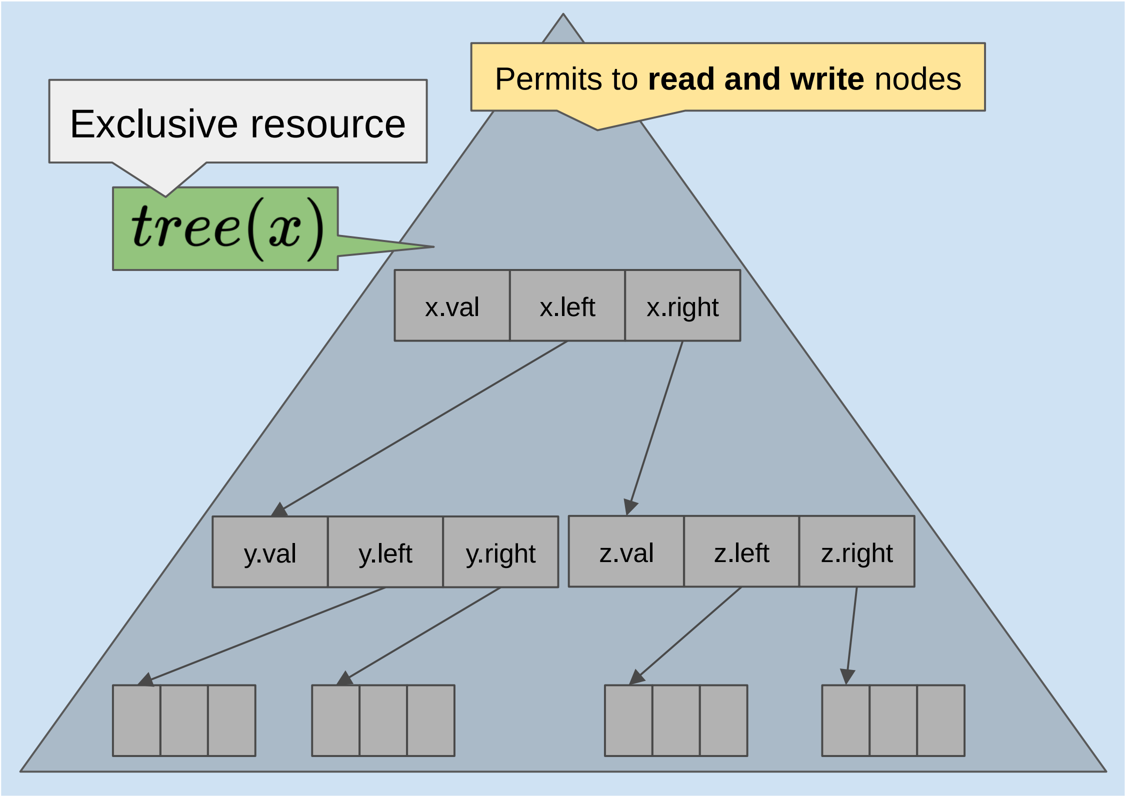
$$tree(x) \triangleq (x \neq \texttt{null} \Rightarrow x.val \mapsto \_ *$$
$$(\exists x_l.\, x.left \mapsto x_l * tree(x_l)) * (\exists x_r.\, x.right \mapsto x_r * tree(x_r)))$$

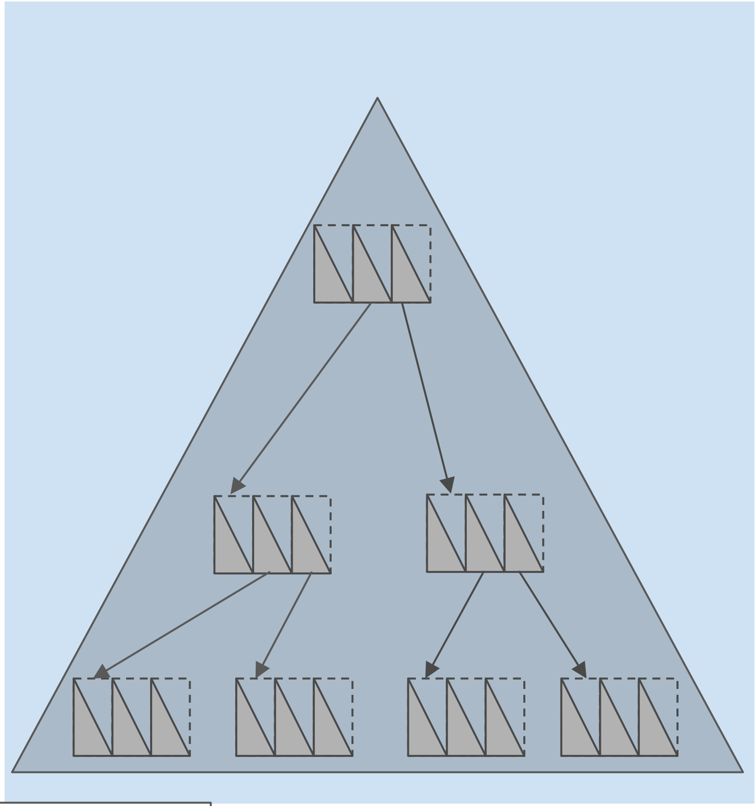
(Fractional) resources, informally



$$tree(x) \triangleq (x \neq \texttt{null} \Rightarrow x.val \mapsto \_ *$$
$$(\exists x_l.\, x.left \mapsto x_l * tree(x_l)) * (\exists x_r.\, x.right \mapsto x_r * tree(x_r)))$$

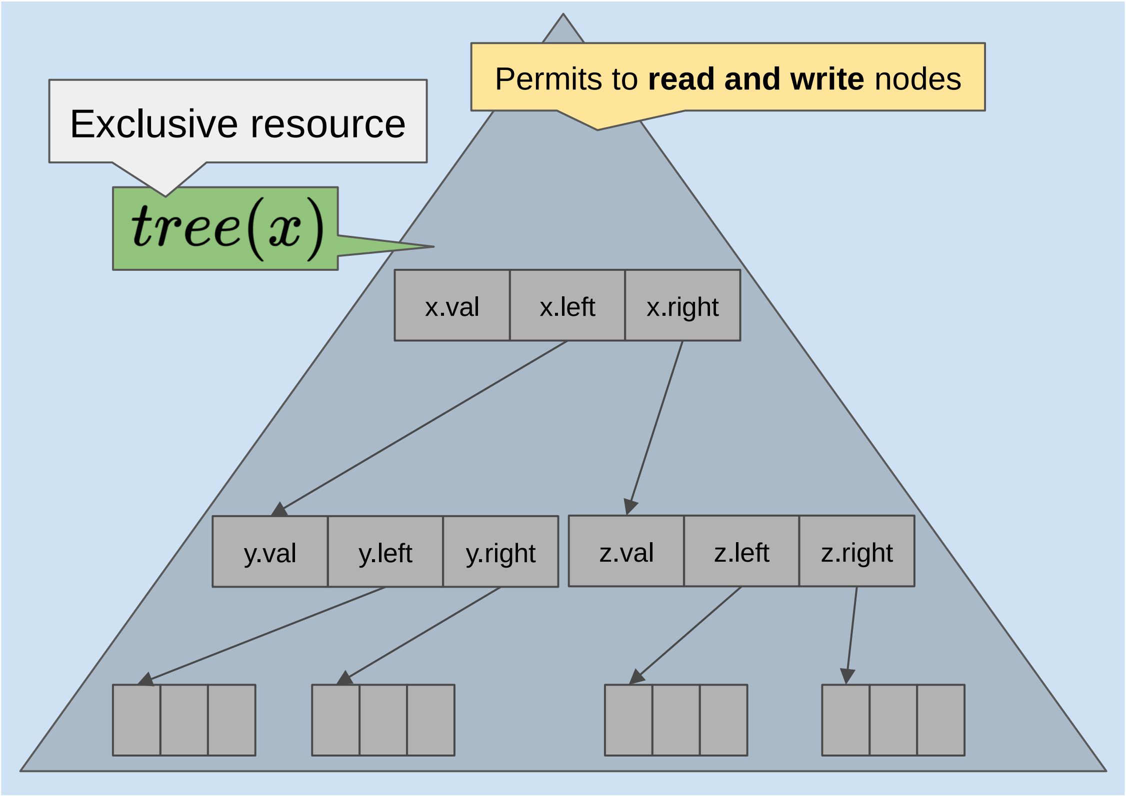
(Fractional) resources, informally

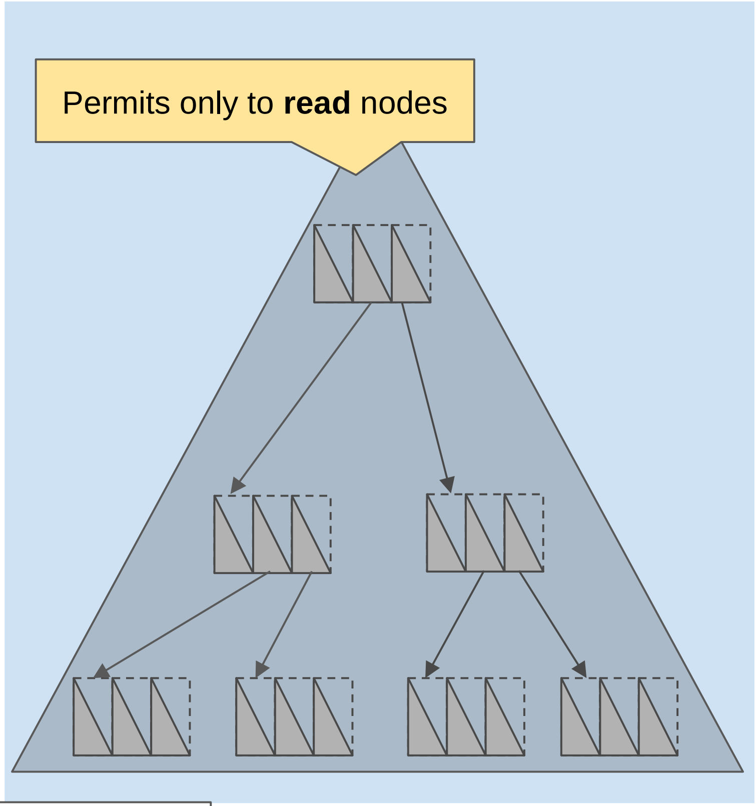




$$tree(x) \triangleq (x \neq \texttt{null} \Rightarrow x.val \mapsto \_ *$$
$$(\exists x_l.\, x.left \mapsto x_l * tree(x_l)) * (\exists x_r.\, x.right \mapsto x_r * tree(x_r)))$$

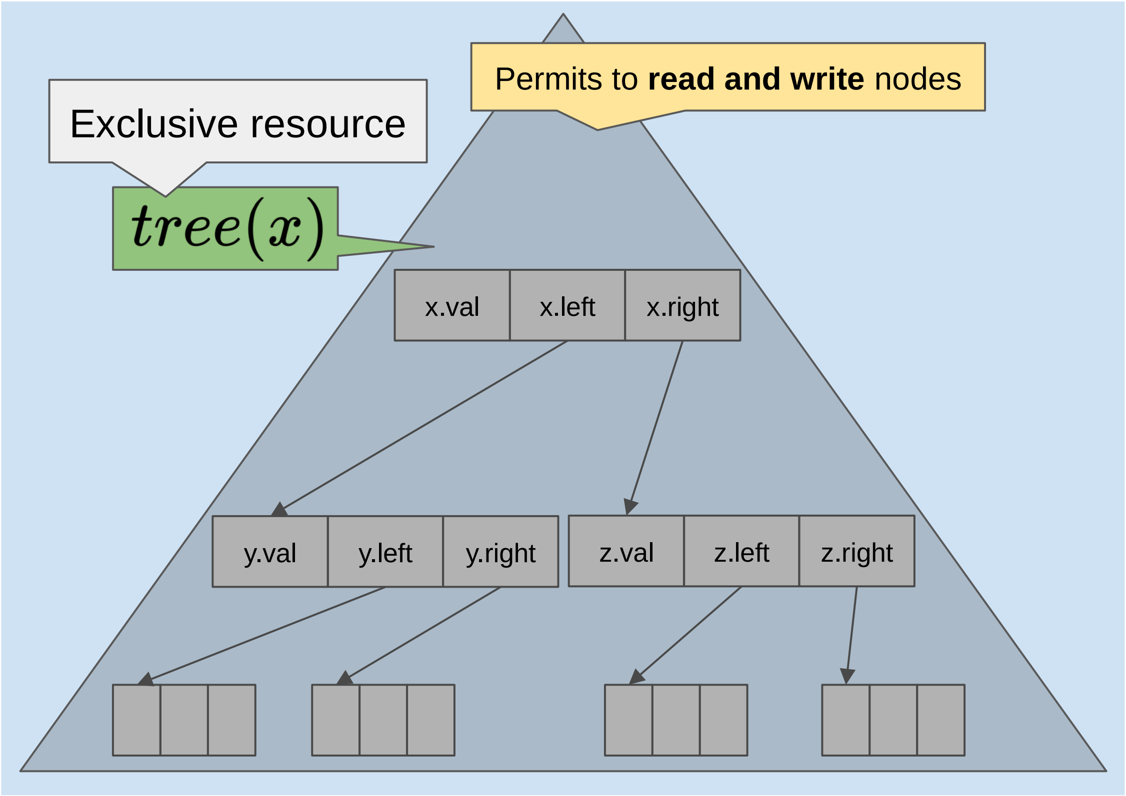
(Fractional) resources, informally

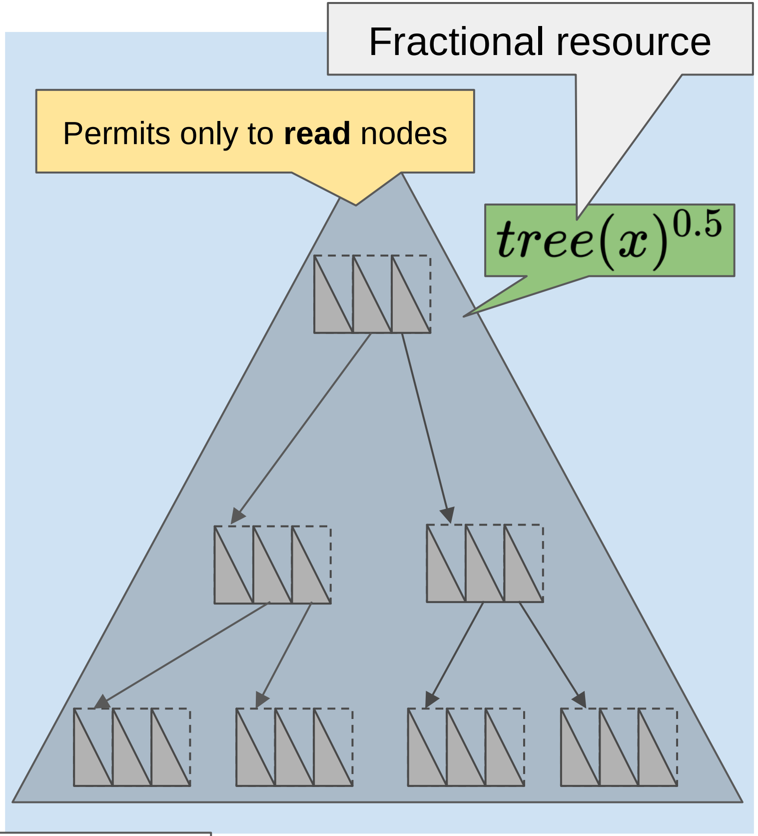




$$tree(x) \triangleq (x \neq \texttt{null} \Rightarrow x.val \mapsto \_ *$$
$$(\exists x_l.\, x.left \mapsto x_l * tree(x_l)) * (\exists x_r.\, x.right \mapsto x_r * tree(x_r)))$$

(Fractional) resources, informally





$$tree(x) \triangleq (x \neq \texttt{null} \Rightarrow x.val \mapsto \_ *$$
$$(\exists x_l.\, x.left \mapsto x_l * tree(x_l)) * (\exists x_r.\, x.right \mapsto x_r * tree(x_r)))$$

Using fractional resources

```
method processTree(x: Ref) {
  {tree(x)^π}
  if (x != null) {
          print(x.val)                    ||    print(x.val)
          processTree(x.left)             ||    processTree(x.left)
          processTree(x.right)            ||    processTree(x.right)
  }
  {tree(x)^π}
}
```

Using fractional resources

```
method processTree(x: Ref) {
  {tree(x)^π}
  if (x != null) {

          print(x.val)                 ||    print(x.val)
          processTree(x.left)          ||    processTree(x.left)
          processTree(x.right)         ||    processTree(x.right)

  }
  {tree(x)^π}
}
```

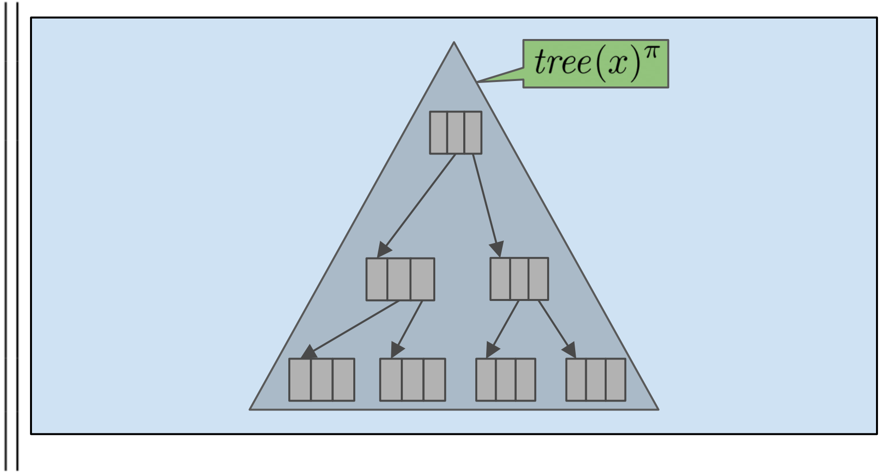
Taken from "Logical Reasoning for Disjoint Permissions", Xuan-Bach Le and Aquinas Hobor (ESOP'18)

Using fractional resources

```
method processTree(x: Ref) {
  {tree(x)^π}
  if (x != null) {

              print(x.val)
              processTree(x.left)
              processTree(x.right)

  }
  {tree(x)^π}
}
```



Using fractional resources

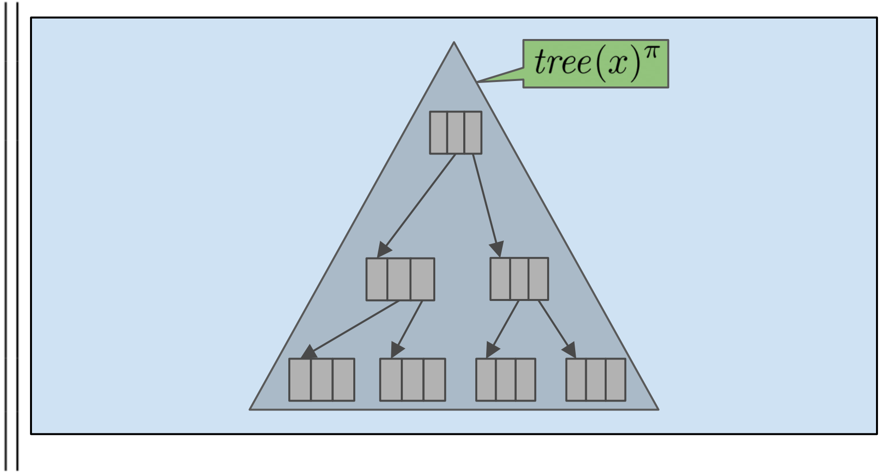
```
method processTree(x: Ref) {
  {tree(x)^π}
  if (x != null) {
    {tree(x)^π * x ≠ null}



              print(x.val)
              processTree(x.left)
              processTree(x.right)



  }
  {tree(x)^π}
}
```



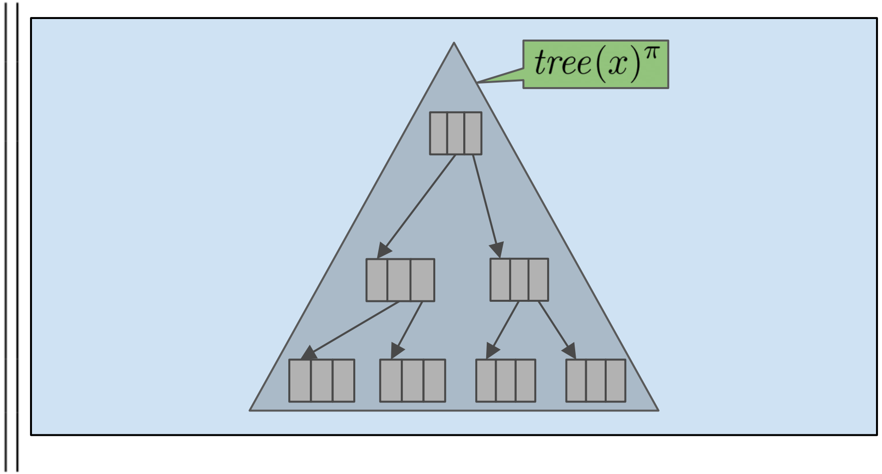
Using fractional resources

```
method processTree(x: Ref) {
  {tree(x)^π}
  if (x != null) {
    {tree(x)^π * x ≠ null}

        print(x.val)
        processTree(x.left)
        processTree(x.right)

  }
  {tree(x)^π}
}
```

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$



Using fractional resources

```
method processTree(x: Ref) {
```

$\{tree(x)^{\pi}\}$

```
  if (x != null) {
```

$\{tree(x)^{\pi} * x \neq null\}$

$\{(tree(x)^{\frac{\pi}{2}} * x \neq null) * (tree(x)^{\frac{\pi}{2}} * x \neq null)\}$

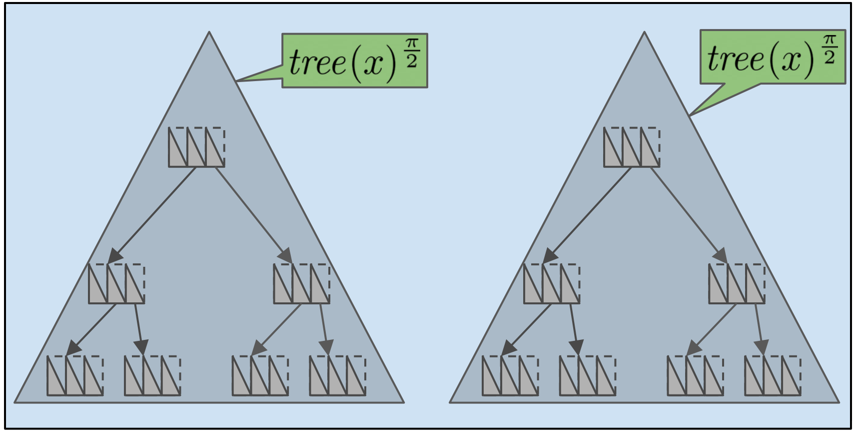
1. Split

```
        print(x.val)
        processTree(x.left)
        processTree(x.right)
  }
```

$\{tree(x)^{\pi}\}$

```
}
```

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$



Using fractional resources

method processTree(x: **Ref**) {

$\{tree(x)^{\pi}\}$

if (x != **null**) {

$\{tree(x)^{\pi} * x \neq null\}$

$\{(tree(x)^{\frac{\pi}{2}} * x \neq null) * (tree(x)^{\frac{\pi}{2}} * x \neq null)\}$

1. Split

$\{tree(x)^{\frac{\pi}{2}} * x \neq null\}$

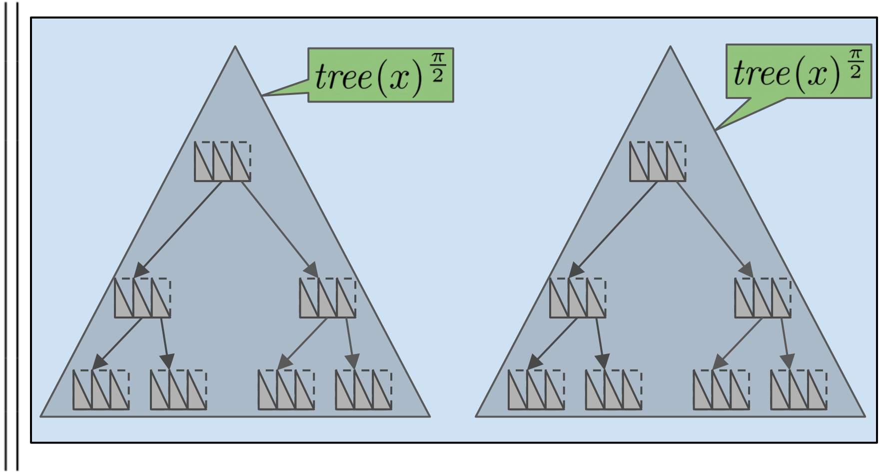
```
print(x.val)
processTree(x.left)
processTree(x.right)
```

}

$\{tree(x)^{\pi}\}$

}

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$



Using fractional resources

```
method processTree(x: Ref) {
  {tree(x)^π}
  if (x != null) {
    {tree(x)^π * x ≠ null}
    {(tree(x)^(π/2) * x ≠ null) * (tree(x)^(π/2) * x ≠ null)}
```

1. Split

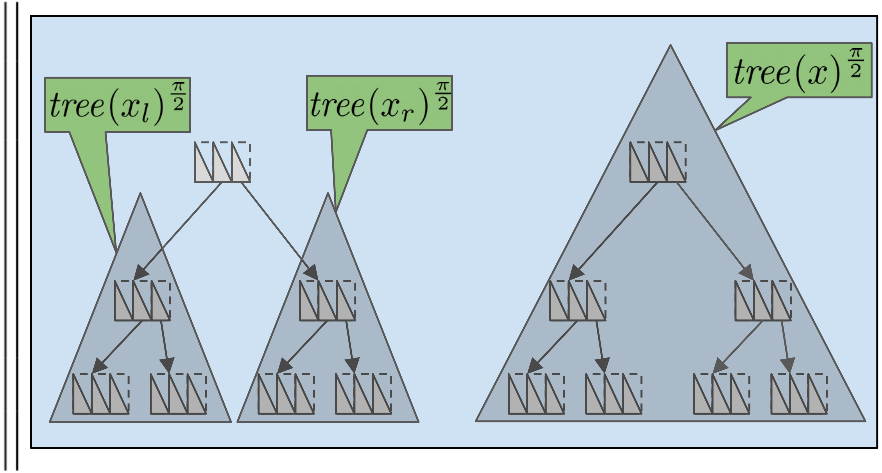
$\{tree(x)^{\frac{\pi}{2}} * x \neq null\}$

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

2. Distribute

```
    print(x.val)
    processTree(x.left)
    processTree(x.right)
  }
  {tree(x)^π}
}
```

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$



Using fractional resources

```
method processTree(x: Ref) {
```

$\{tree(x)^{\pi}\}$

```
  if (x != null) {
```

$\{tree(x)^{\pi} * x \neq null\}$

$\{(tree(x)^{\frac{\pi}{2}} * x \neq null) * (tree(x)^{\frac{\pi}{2}} * x \neq null)\}$

1. Split

$\{tree(x)^{\frac{\pi}{2}} * x \neq null\}$

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

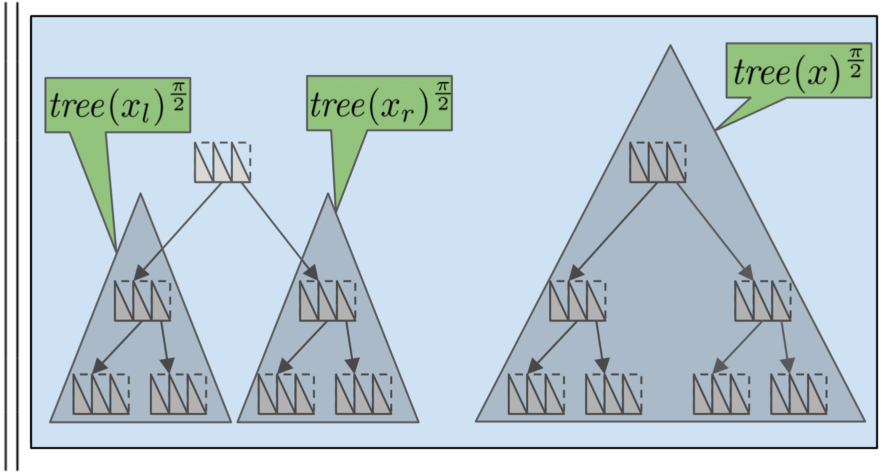
2. Distribute

```
    print(x.val)
    processTree(x.left)
    processTree(x.right)
  }
```

$\{tree(x)^{\pi}\}$

```
}
```

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$



Using fractional resources

```
method processTree(x: Ref) {
```
$\{tree(x)^{\pi}\}$
```
  if (x != null) {
```
$\{tree(x)^{\pi} * x \neq null\}$

$\{(tree(x)^{\frac{\pi}{2}} * x \neq null) * (tree(x)^{\frac{\pi}{2}} * x \neq null)\}$

1. Split

$\{tree(x)^{\frac{\pi}{2}} * x \neq null\}$

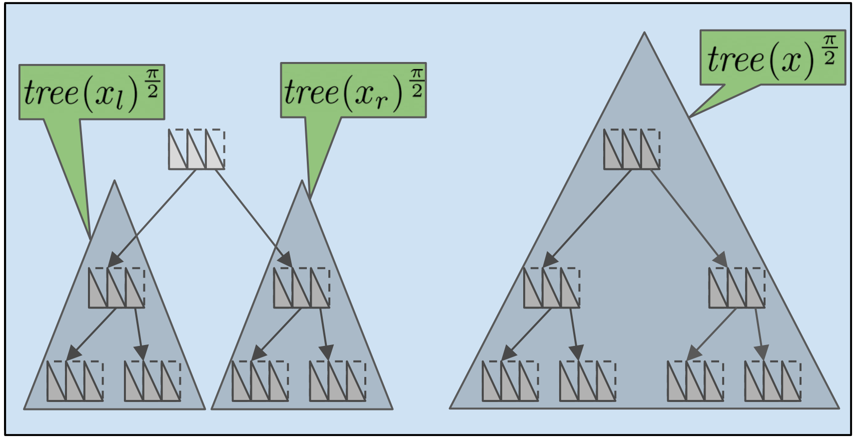
$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

2. Distribute

```
    print(x.val)
    processTree(x.left)
    processTree(x.right)
  }
```
$\{tree(x)^{\pi}\}$
```
}
```

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$

||



Using fractional resources

```
method processTree(x: Ref) {
```
$\{tree(x)^{\pi}\}$
```
  if (x != null) {
```
$\{tree(x)^{\pi} * x \neq null\}$

$\{(tree(x)^{\frac{\pi}{2}} * x \neq null) * (tree(x)^{\frac{\pi}{2}} * x \neq null)\}$

1. Split

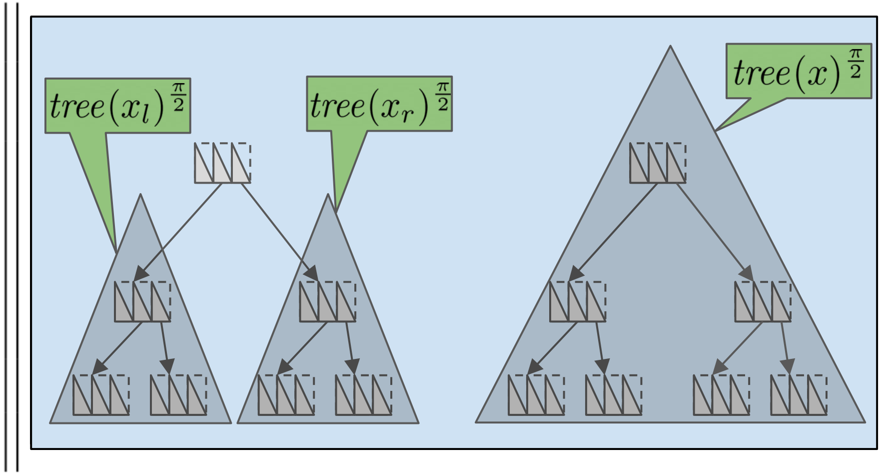
$\{tree(x)^{\frac{\pi}{2}} * x \neq null\}$

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

2. Distribute

```
    print(x.val)
    processTree(x.left)
    processTree(x.right)
  }
```
$\{tree(x)^{\pi}\}$
```
}
```

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$



Using fractional resources

```
method processTree(x: Ref) {
```

$\{tree(x)^{\pi}\}$

```
  if (x != null) {
```

$\{tree(x)^{\pi} * x \neq null\}$

$\{(tree(x)^{\frac{\pi}{2}} * x \neq null) * (tree(x)^{\frac{\pi}{2}} * x \neq null)\}$

1. Split

$\{tree(x)^{\frac{\pi}{2}} * x \neq null\}$

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

2. Distribute

```
    print(x.val)
    processTree(x.left)
    processTree(x.right)
```

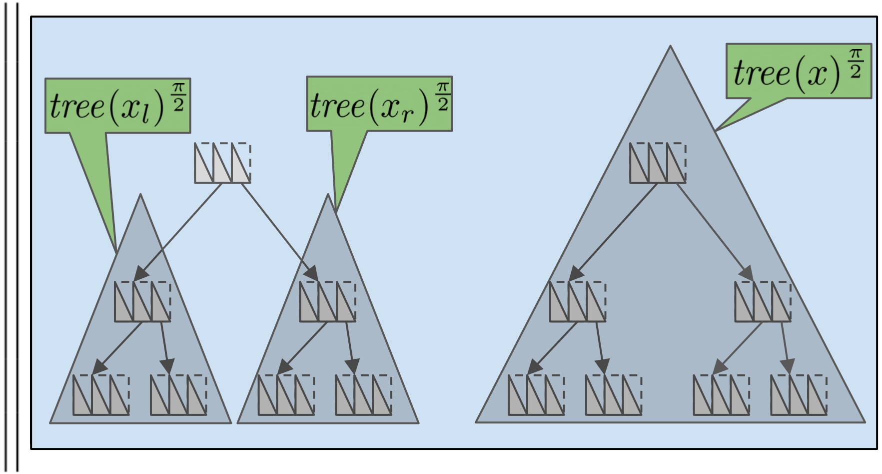
$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

```
  }
```

$\{tree(x)^{\pi}\}$

```
}
```

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$



Using fractional resources

```
method processTree(x: Ref) {
```

$\{tree(x)^{\pi}\}$

```
  if (x != null) {
```

$\{tree(x)^{\pi} * x \neq null\}$

$\{(tree(x)^{\frac{\pi}{2}} * x \neq null) * (tree(x)^{\frac{\pi}{2}} * x \neq null)\}$

1. Split

$\{tree(x)^{\frac{\pi}{2}} * x \neq null\}$

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

2. Distribute

```
    print(x.val)
    processTree(x.left)
    processTree(x.right)
```

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

$\{tree(x)^{\frac{\pi}{2}}\}$

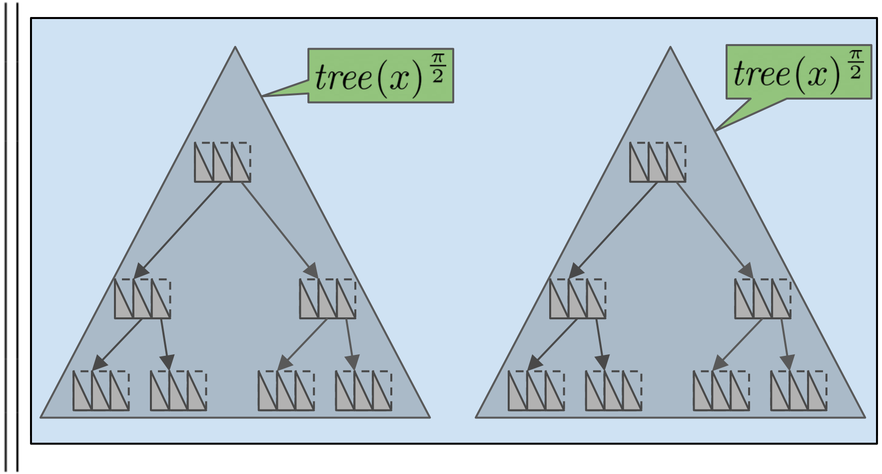
3. Factorise

```
  }
```

$\{tree(x)^{\pi}\}$

```
}
```

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$



Using fractional resources

```
method processTree(x: Ref) {
```

$\{tree(x)^{\pi}\}$

```
  if (x != null) {
```

$\{tree(x)^{\pi} * x \neq null\}$

$\{(tree(x)^{\frac{\pi}{2}} * x \neq null) * (tree(x)^{\frac{\pi}{2}} * x \neq null)\}$

1. Split

$\{tree(x)^{\frac{\pi}{2}} * x \neq null\}$

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

2. Distribute

```
    print(x.val)
    processTree(x.left)
    processTree(x.right)
```

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

$\{tree(x)^{\frac{\pi}{2}}\}$

3. Factorise

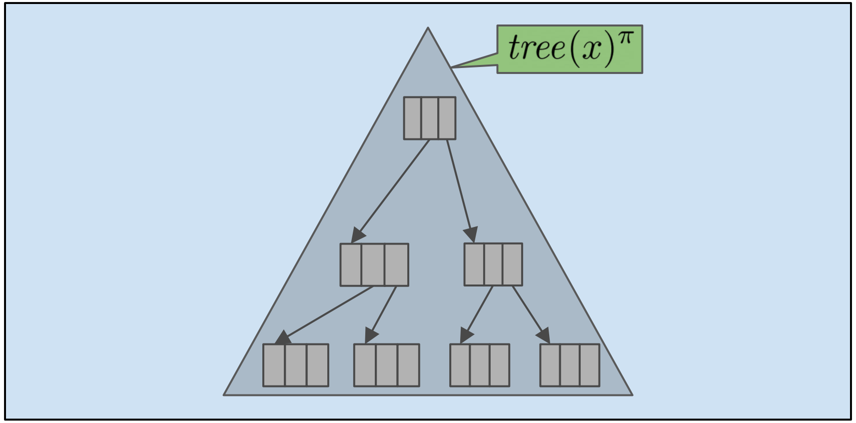
$\{tree(x)^{\frac{\pi}{2}} * tree(x)^{\frac{\pi}{2}}\}$

```
  }
```

$\{tree(x)^{\pi}\}$

```
}
```

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \boxed{\{Q_1 * Q_2\}}}\ (Parallel)$$



Using fractional resources

```
method processTree(x: Ref) {
```

$\{tree(x)^{\pi}\}$

```
  if (x != null) {
```

$\{tree(x)^{\pi} * x \neq null\}$

$\{(tree(x)^{\frac{\pi}{2}} * x \neq null) * (tree(x)^{\frac{\pi}{2}} * x \neq null)\}$

1. Split

$\{tree(x)^{\frac{\pi}{2}} * x \neq null\}$

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

2. Distribute

```
    print(x.val)
    processTree(x.left)
    processTree(x.right)
```

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

$\{tree(x)^{\frac{\pi}{2}}\}$

3. Factorise

$\{tree(x)^{\frac{\pi}{2}} * tree(x)^{\frac{\pi}{2}}\}$

```
  }
```

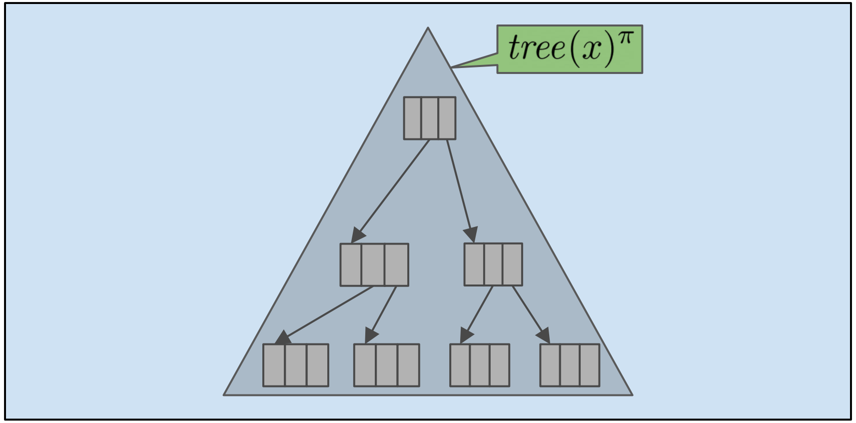
$\{tree(x)^{\pi}\}$

```
}
```

4. Combine

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \boxed{\{Q_1 * Q_2\}}}\ (Parallel)$$

$\parallel$



Using fractional resources

```
method processTree(x: Ref) {
```

$\{tree(x)^{\pi}\}$

```
  if (x != null) {
```

$\{tree(x)^{\pi} * x \neq null\}$

$\{(tree(x)^{\frac{\pi}{2}} * x \neq null) * (tree(x)^{\frac{\pi}{2}} * x \neq null)\}$

1. Split

$\{tree(x)^{\frac{\pi}{2}} * x \neq null\}$

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

2. Distribute

```
    print(x.val)
    processTree(x.left)
    processTree(x.right)
```

$\{x.val \overset{\frac{\pi}{2}}{\mapsto} \_ * \ldots * tree(x_l)^{\frac{\pi}{2}} * tree(x_r)^{\frac{\pi}{2}}\}$

$\{tree(x)^{\frac{\pi}{2}}\}$

3. Factorise

$\{tree(x)^{\frac{\pi}{2}} * tree(x)^{\frac{\pi}{2}}\}$

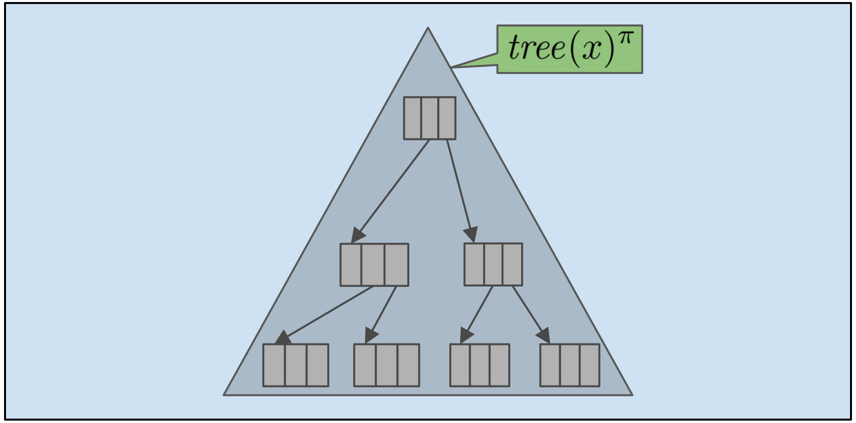
```
  }
```

$\{tree(x)^{\pi}\}$

```
}
```

4. Combine

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$



Is this proof outline actually correct?

Is this proof outline actually correct?

It depends on the meaning of fractional resources.

The meaning(s) of fractional resources

The meaning(s) of fractional resources

Semantic multiplication

Studied in theoretical papers

The meaning(s) of fractional resources

Semantic multiplication

Studied in theoretical papers

Previous proof outline ❌

The meaning(s) of fractional resources

Semantic multiplication

Studied in theoretical papers

$$A^{\pi}$$

Previous proof outline ✗

The meaning(s) of fractional resources

Semantic multiplication

Studied in theoretical papers

$$A^{\pi}$$

$$\text{State} \triangleq \text{Locations} \rightharpoonup \text{Values} \times (\mathbb{Q} \cap (0,1])$$

$$h \models A^{\pi} \text{ iff there exists } h_A \text{ such that}$$
$$h = \pi \odot h_A \text{ and } h_A \models A$$

Previous proof outline ✗

The meaning(s) of fractional resources

Semantic multiplication

Studied in theoretical papers

$$A^\pi$$

$$\text{State} \triangleq \text{Locations} \rightharpoonup \text{Values} \times (\mathbb{Q} \cap (0, 1])$$

$$h \models A^\pi \text{ iff there exists } h_A \text{ such that}$$

$$h = \boxed{\pi \odot h_A} \text{ and } h_A \models A$$

All permission amounts are multiplied by π

Previous proof outline ✗

The meaning(s) of fractional resources

Semantic multiplication

Studied in theoretical papers

$$A^{\pi}$$

$$\text{State} \triangleq \text{Locations} \rightharpoonup \text{Values} \times (\mathbb{Q} \cap (0,1])$$

$$h \models A^{\pi} \text{ iff there exists } h_A \text{ such that}$$
$$h = \boxed{\pi \odot h_A} \text{ and } h_A \models A$$

All permission amounts are multiplied by π

Previous proof outline ❌

Syntactic multiplication

Implemented in automatic separation logic verifiers (e.g, VeriFast, Viper…)

The meaning(s) of fractional resources

Semantic multiplication

Studied in theoretical papers

$$A^\pi$$

$$\text{State} \triangleq \text{Locations} \rightharpoonup \text{Values} \times (\mathbb{Q} \cap (0,1])$$

$$h \models A^\pi \text{ iff there exists } h_A \text{ such that}$$
$$h = \boxed{\pi \odot h_A} \text{ and } h_A \models A$$

All permission amounts are multiplied by π

Previous proof outline ✗

Syntactic multiplication

Implemented in automatic separation logic verifiers (e.g, VeriFast, Viper…)

Previous proof outline ✓

The meaning(s) of fractional resources

Semantic multiplication

Studied in theoretical papers

$$A^\pi$$

$$\text{State} \triangleq \text{Locations} \rightharpoonup \text{Values} \times (\mathbb{Q} \cap (0,1])$$

$$h \models A^\pi \text{ iff there exists } h_A \text{ such that}$$
$$h = \boxed{\pi \odot h_A} \text{ and } h_A \models A$$

All permission amounts are multiplied by π

Previous proof outline ✗

Syntactic multiplication

Implemented in automatic separation logic verifiers (e.g, VeriFast, Viper…)

$$\pi \cdot A$$

Previous proof outline ✓

The meaning(s) of fractional resources

Semantic multiplication

Studied in theoretical papers

$$A^\pi$$

$$\mathrm{State} \triangleq \mathrm{Locations} \rightharpoonup \mathrm{Values} \times (\mathbb{Q} \cap (0,1])$$

$$h \models A^\pi \text{ iff there exists } h_A \text{ such that}$$
$$h = \boxed{\pi \odot h_A} \text{ and } h_A \models A$$

All permission amounts are multiplied by π

Previous proof outline ✗

Syntactic multiplication

Implemented in automatic separation logic verifiers (e.g, VeriFast, Viper…)

$$\pi \cdot A$$

$$0.5 \cdot (l_1 \mapsto v_1 * l_2 \mapsto v_2)$$
$$\triangleq 0.5 \cdot (l_1 \mapsto v_1) * 0.5 \cdot (l_2 \mapsto v_2)$$
$$\triangleq (l_1 \overset{0.5}{\mapsto} v_1) * (l_2 \overset{0.5}{\mapsto} v_2)$$

Previous proof outline ✓

This work

This work

- We discovered a discrepancy between two notions of fractional resources
  - **Syntactic** multiplication: Rules implemented in automated verifiers, no formal foundation
  - **Semantic** multiplication: Theoretical foundation, shortcomings
- We present and formalise a new logic: **unbounded separation logic**
  - Formal foundation for the **syntactic** multiplication
  - Eliminates shortcomings from the **semantic** multiplication
- In-depth study of **combinability** in unbounded separation logic
- Reasoning principles for (co)inductive predicates
- *Unbounded separation logic* as a formal foundation for automatic verifiers
  - Justifies the rules used
  - Shows how to extend them to other constructs

This work

- We discovered a discrepancy between two notions of fractional resources
  - **Syntactic** multiplication: Rules implemented in automated verifiers, no formal foundation
  - **Semantic** multiplication: Theoretical foundation, shortcomings
- We present and formalise a new logic: **unbounded separation logic**
  - Formal foundation for the **syntactic** multiplication
  - Eliminates shortcomings from the **semantic** multiplication
- In-depth study of **combinability** in unbounded separation logic
- Reasoning principles for (co)inductive predicates
- *Unbounded separation logic* as a formal foundation for automatic verifiers
  - Justifies the rules used
  - Shows how to extend them to other constructs

Semantic multiplication ≠ syntactic multiplication (1/2)

$$tree(x)^{0.5}$$

Semantic

$$0.5 \cdot tree(x)$$

Syntactic

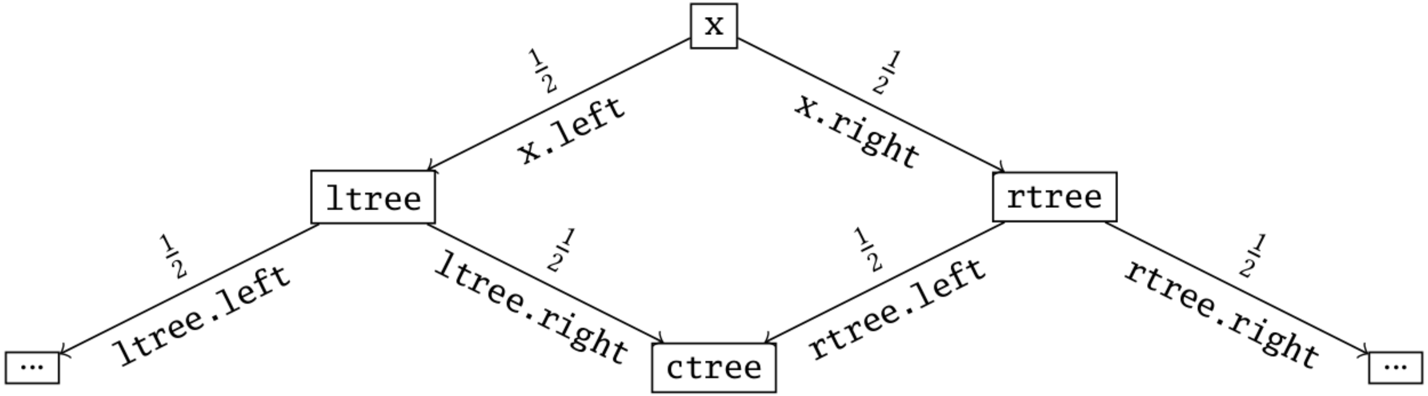
Semantic multiplication ≠ syntactic multiplication (1/2)

$$tree(x)^{0.5}$$

Semantic

$$0.5 \cdot tree(x)$$

Syntactic



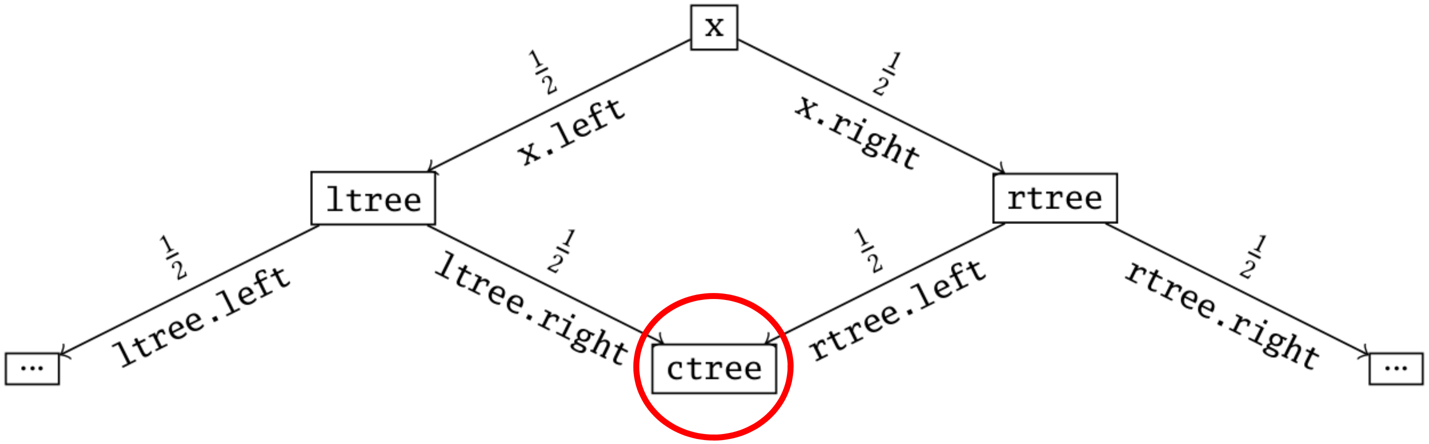
Semantic multiplication ≠ syntactic multiplication (1/2)

$$tree(x)^{0.5}$$

Semantic

$$0.5 \cdot tree(x)$$

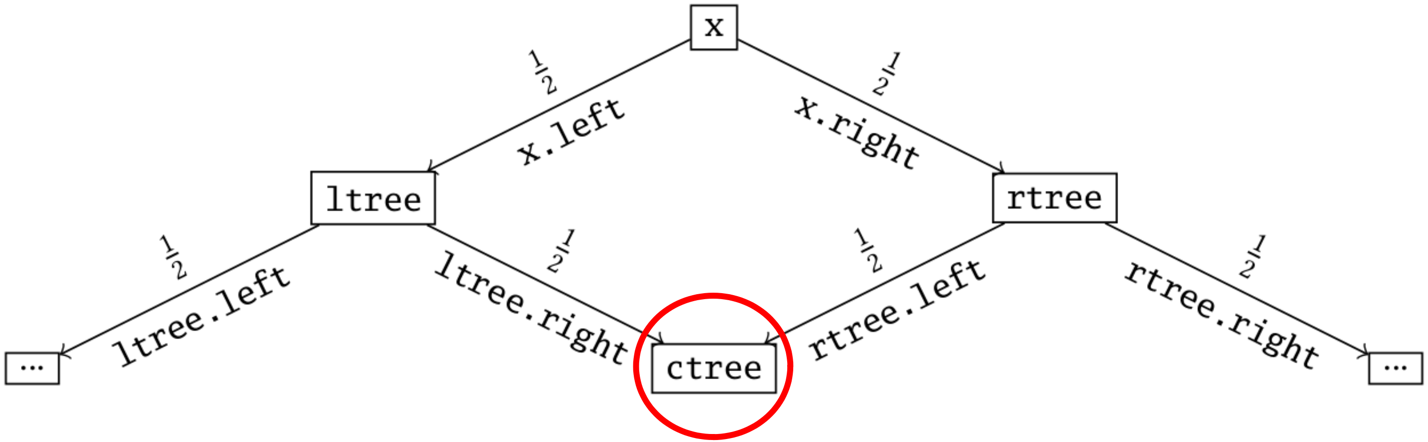
Syntactic



Semantic multiplication ≠ syntactic multiplication (1/2)

$tree(x)^{0.5}$ ✗ — Semantic

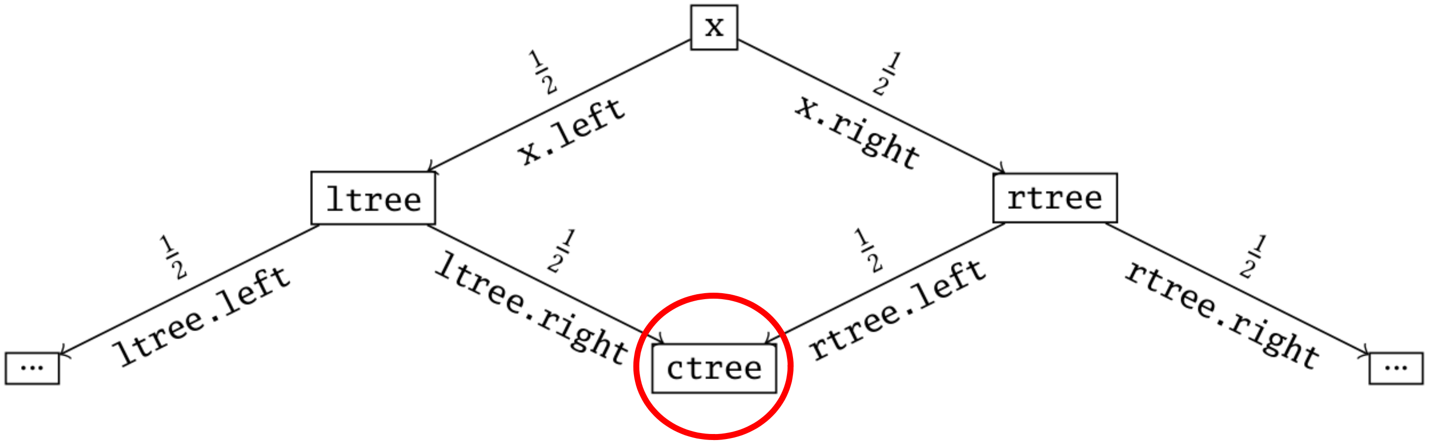
$0.5 \cdot tree(x)$ — Syntactic



Semantic multiplication ≠ syntactic multiplication (1/2)

$tree(x)^{0.5}$ ✗ — Semantic

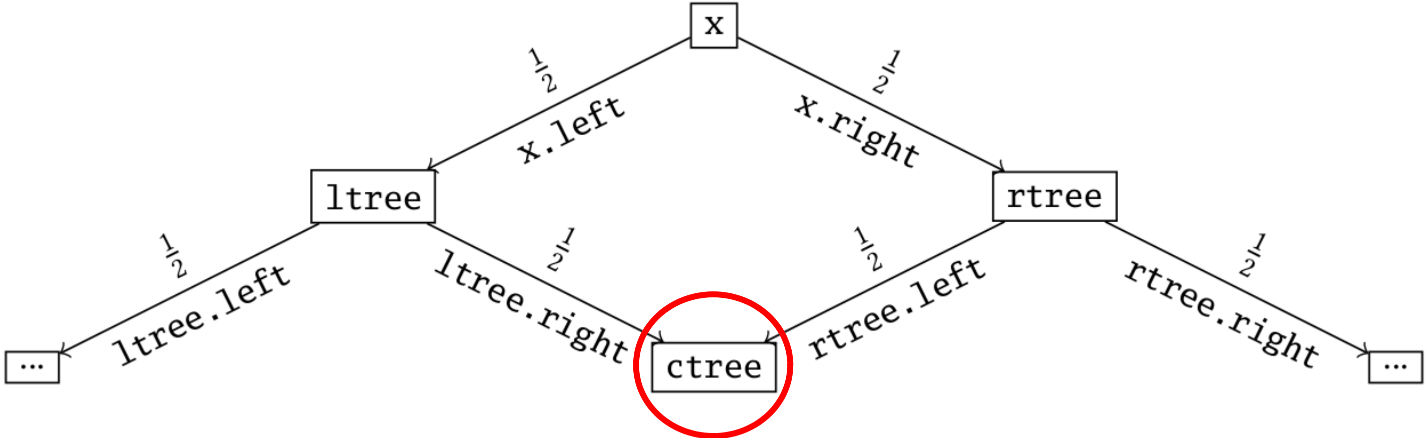
$0.5 \cdot tree(x)$ ✓ — Syntactic



Semantic multiplication ≠ syntactic multiplication (1/2)

$tree(x)^{0.5}$ ✗ — Semantic

$0.5 \cdot tree(x)$ ✓ — Syntactic



$$tree(x) \triangleq (x \neq \texttt{null} \Rightarrow \ldots * tree(rtree) * \ldots * tree(ltree))$$

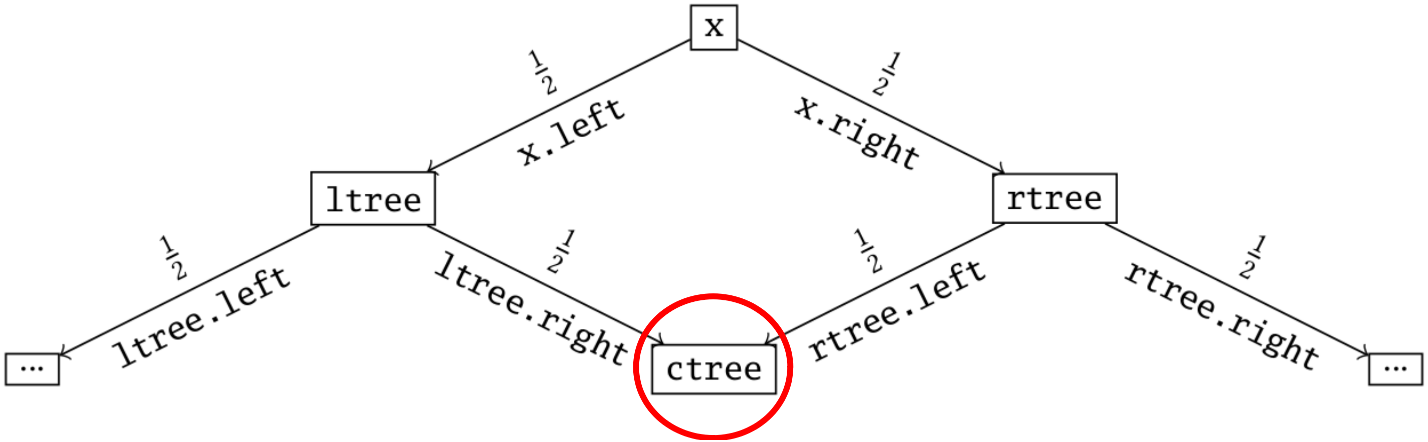
$$0.5 \cdot tree(x) \triangleq (x \neq \texttt{null} \Rightarrow \ldots * 0.5 \cdot tree(rtree) * \ldots * 0.5 \cdot tree(ltree))$$

Syntactic — Syntactic — Syntactic

Semantic multiplication ≠ syntactic multiplication (1/2)

$tree(x)^{0.5}$ ✗ — Semantic

$0.5 \cdot tree(x)$ ✓ — Syntactic



$$tree(x) \triangleq (x \neq \text{null} \Rightarrow \ldots * tree(rtree) * \ldots * tree(ltree))$$

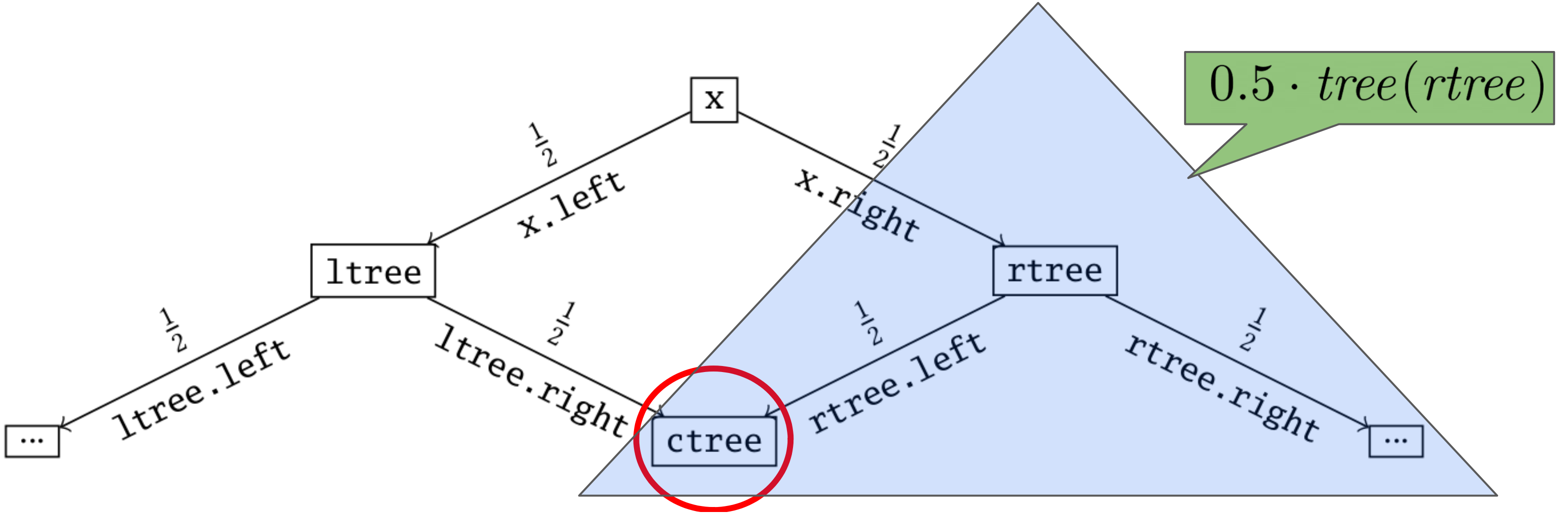
$$0.5 \cdot tree(x) \triangleq (x \neq \text{null} \Rightarrow \ldots * 0.5 \cdot tree(rtree) * \ldots * 0.5 \cdot tree(ltree))$$

Syntactic — Syntactic — Syntactic

Semantic multiplication ≠ syntactic multiplication (1/2)

$tree(x)^{0.5}$ ✗ — Semantic

$0.5 \cdot tree(x)$ ✓ — Syntactic



$$tree(x) \triangleq (x \neq \text{null} \Rightarrow \ldots * tree(rtree) * \ldots * tree(ltree))$$

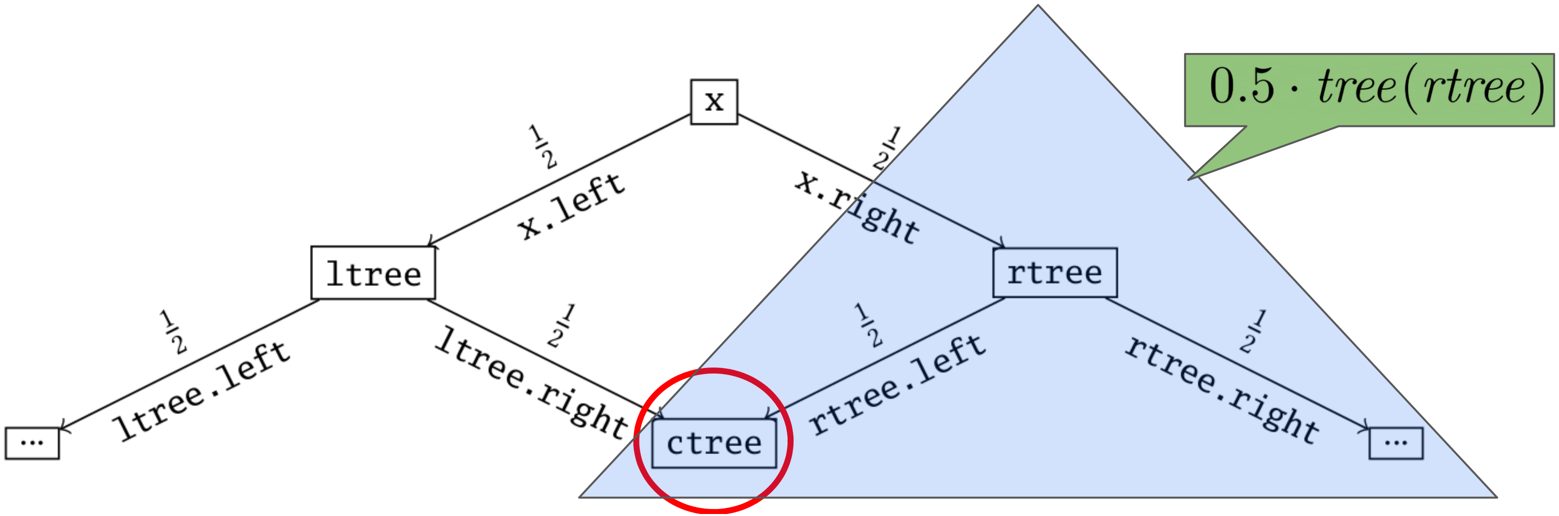
$$0.5 \cdot tree(x) \triangleq (x \neq \text{null} \Rightarrow \ldots * 0.5 \cdot tree(rtree) * \ldots * 0.5 \cdot tree(ltree))$$

Syntactic — Syntactic — Syntactic

Semantic multiplication ≠ syntactic multiplication (1/2)

$tree(x)^{0.5}$ ✗ — Semantic

$0.5 \cdot tree(x)$ ✓ — Syntactic



$$tree(x) \triangleq (x \neq \text{null} \Rightarrow \ldots * tree(rtree) * \ldots * tree(ltree))$$

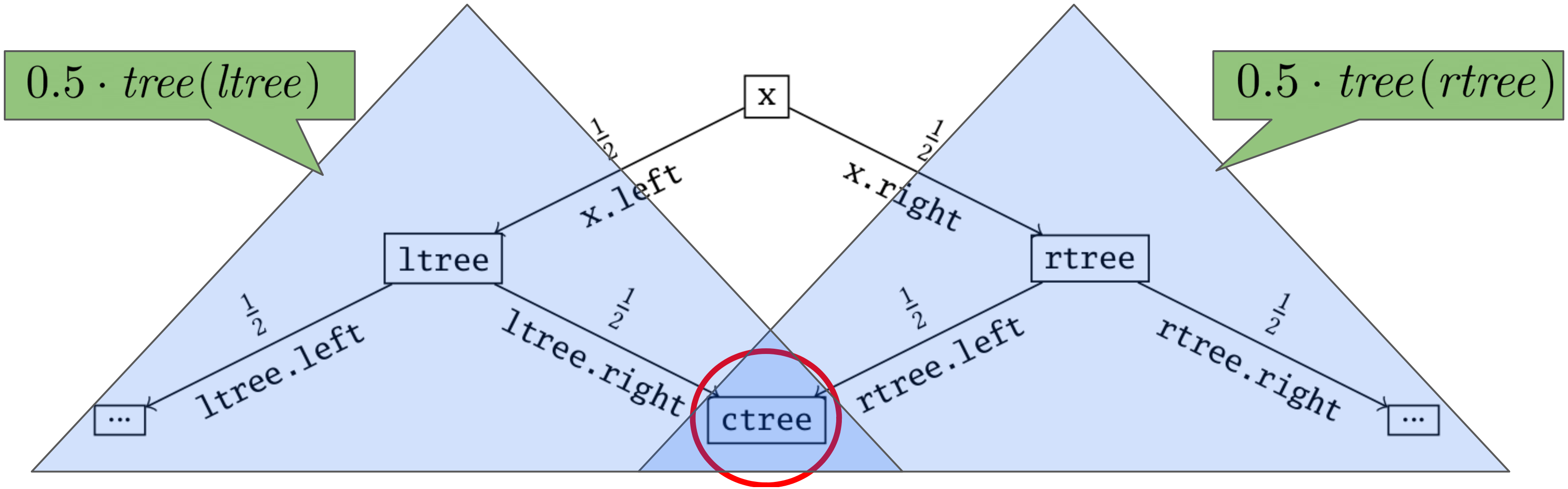
$$0.5 \cdot tree(x) \triangleq (x \neq \text{null} \Rightarrow \ldots * 0.5 \cdot tree(rtree) * \ldots * 0.5 \cdot tree(ltree))$$

Syntactic — Syntactic — Syntactic

Semantic multiplication ≠ syntactic multiplication (1/2)

$tree(x)^{0.5}$ ✗ — Semantic

$0.5 \cdot tree(x)$ ✓ — Syntactic



$$tree(x) \triangleq (x \neq \text{null} \Rightarrow \ldots * tree(rtree) * \ldots * tree(ltree))$$

$$0.5 \cdot tree(x) \triangleq (x \neq \text{null} \Rightarrow \ldots * 0.5 \cdot tree(rtree) * \ldots * 0.5 \cdot tree(ltree))$$

Syntactic — Syntactic — Syntactic

Semantic multiplication ≠ syntactic multiplication (2/2)

Syntactic

Semantic

$$0.5 \cdot (l_1 \mapsto v_1 * l_2 \mapsto v_2) \not\models (l_1 \mapsto v_1 * l_2 \mapsto v_2)^{0.5}$$

$$(l_1 \overset{0.5}{\mapsto} v_1) * (l_2 \overset{0.5}{\mapsto} v_2)$$

Semantic multiplication ≠ syntactic multiplication (2/2)

Syntactic

l_1 and l_2 **cannot** be aliases

Semantic

$$0.5 \cdot (l_1 \mapsto v_1 * l_2 \mapsto v_2) \not\models (l_1 \mapsto v_1 * l_2 \mapsto v_2)^{0.5}$$

$$(l_1 \overset{0.5}{\mapsto} v_1) * (l_2 \overset{0.5}{\mapsto} v_2)$$

Semantic multiplication ≠ syntactic multiplication (2/2)

Syntactic

l_1 and l_2 **cannot** be aliases

Semantic

$$0.5 \cdot (l_1 \mapsto v_1 * l_2 \mapsto v_2) \not\models (l_1 \mapsto v_1 * l_2 \mapsto v_2)^{0.5}$$

$$(l_1 \overset{0.5}{\mapsto} v_1) * (l_2 \overset{0.5}{\mapsto} v_2)$$

l_1 and l_2 **can** be aliases

Semantic multiplication ≠ syntactic multiplication (2/2)

Syntactic

l_1 and l_2 **cannot** be aliases

Semantic

$$0.5 \cdot (l_1 \mapsto v_1 * l_2 \mapsto v_2) \not\models (l_1 \mapsto v_1 * l_2 \mapsto v_2)^{0.5}$$

$$(l_1 \overset{0.5}{\mapsto} v_1) * (l_2 \overset{0.5}{\mapsto} v_2)$$

l_1 and l_2 **can** be aliases

$$(l_1 \mapsto v_1)^{0.5} * (l_2 \mapsto v_2)^{0.5}$$

Semantic

Semantic

Cannot factorise!

$$A^\pi * B^\pi \not\models (A * B)^\pi$$

Summary

	Semantic multiplication	Syntactic multiplication
Factorisability ($*$)		
Distributivity ($*$)		

Factorise

$$A^\pi * B^\pi \longrightarrow (A * B)^\pi$$

Summary

	Semantic multiplication	**Syntactic multiplication**
Factorisability ($*$)	❌	
Distributivity ($*$)		

$$A^\pi * B^\pi \xrightarrow{\text{Factorise}} (A * B)^\pi$$

Summary

	Semantic multiplication	**Syntactic multiplication**
Factorisability ($*$)	✗	
Distributivity ($*$)		

Factorise →

$$A^{\pi} * B^{\pi} \rightleftarrows (A * B)^{\pi}$$

← Distribute

Summary

	Semantic multiplication	**Syntactic multiplication**
Factorisability ($*$)	✗	
Distributivity ($*$)	✓	

$$A^{\pi} * B^{\pi} \underset{\text{Distribute}}{\overset{\text{Factorise}}{\rightleftarrows}} (A * B)^{\pi}$$

Summary

	Semantic multiplication	Syntactic multiplication
Factorisability (*)	✗	✓
Distributivity (*)	✓	✓

$$A^{\pi} * B^{\pi} \underset{\text{Distribute}}{\overset{\text{Factorise}}{\rightleftarrows}} (A * B)^{\pi}$$

Summary

	Semantic multiplication	Syntactic multiplication
Factorisability (∗)	✗	✓
Distributivity (∗)	✓	✓
Factorisability (– ∗)		
Distributivity (– ∗)		

Separating implication (magic wand)

$$A^{\pi} * B^{\pi} \underset{\text{Distribute}}{\overset{\text{Factorise}}{\rightleftarrows}} (A * B)^{\pi}$$

Summary

	Semantic multiplication	**Syntactic multiplication**
Factorisability ($*$)	✗	✓
Distributivity ($*$)	✓	✓
Factorisability ($-*$)	✓	
Distributivity ($-*$)	✗	

Separating implication (magic wand)

Factorise

$$A^\pi * B^\pi \rightleftarrows (A * B)^\pi$$

Distribute

Summary

has shortcomings

	Semantic multiplication	**Syntactic multiplication**
Factorisability (∗)	✗	✓
Distributivity (∗)	✓	✓
Factorisability (–∗)	✓	
Distributivity (–∗)	✗	

Separating implication (magic wand)

Factorise

$$A^\pi * B^\pi \rightleftarrows (A * B)^\pi$$

Distribute

Summary

has shortcomings

	Semantic multiplication	**Syntactic multiplication**
Factorisability (∗)	✗	✓
Distributivity (∗)	✓	✓
Factorisability (–∗)	✓	?
Distributivity (–∗)	✗	?

Unsupported

Separating implication (magic wand)

Factorise

$$A^{\pi} * B^{\pi} \rightleftarrows (A * B)^{\pi}$$

Distribute

Summary

has shortcomings

no theoretical foundation

	Semantic multiplication	**Syntactic multiplication**
Factorisability ($*$)	✗	✓
Distributivity ($*$)	✓	✓
Factorisability ($-*$)	✓	?
Distributivity ($-*$)	✗	?

Unsupported

Separating implication (magic wand)

Factorise

$$A^{\pi} * B^{\pi} \rightleftarrows (A * B)^{\pi}$$

Distribute

Unbounded separation logic

has shortcomings

no theoretical foundation

	Semantic multiplication	Syntactic multiplication
Factorisability (*)	✗	✓
Distributivity (*)	✓	✓
Factorisability (–*)	✓	?
Distributivity (–*)	✗	?

In **bounded** separation logic

Unbounded separation logic

has shortcomings

no theoretical foundation

	Semantic multiplication	Syntactic multiplication
Factorisability (∗)	✗	✓
Distributivity (∗)	✓	✓
Factorisability (−∗)	✓	?
Distributivity (−∗)	✗	?

In **bounded** separation logic

(Syntactic) multiplication
✓
✓
✓
✓

In **unbounded** separation logic

provides a theoretical foundation

Unbounded separation logic: Intuition

Semantic

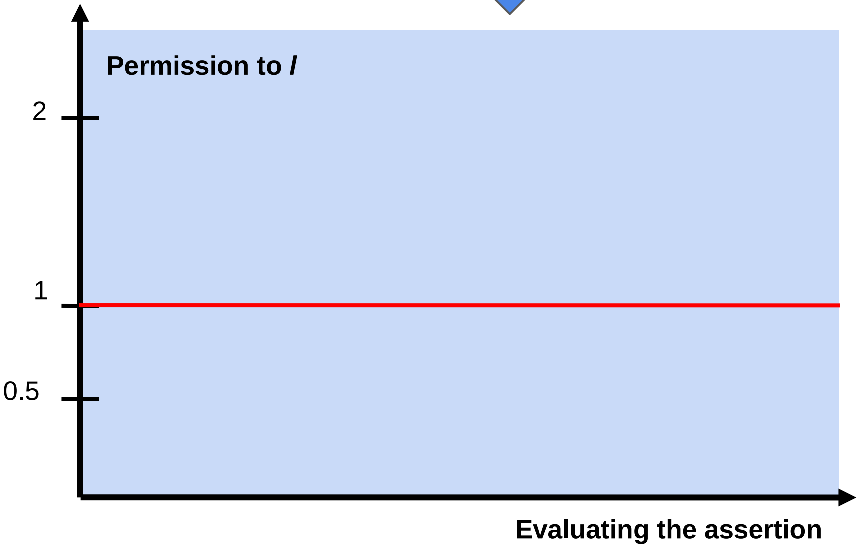
$$0.5 \cdot (l \mapsto v * l \mapsto v) \not\models (l \mapsto v * l \mapsto v)^{0.5}$$

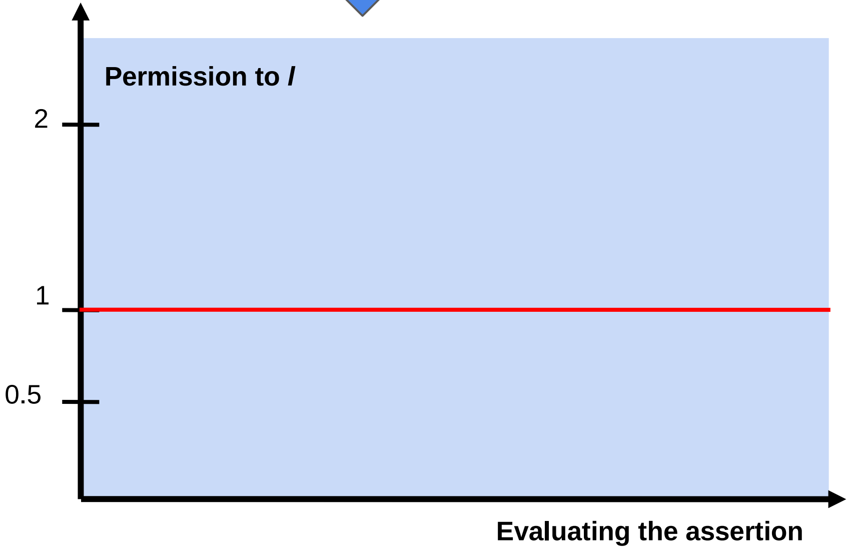
Syntactic

$$0.5 \cdot (l \mapsto v) * 0.5 \cdot (l \mapsto v)$$

Syntactic

Syntactic





Unbounded separation logic: Intuition

Semantic

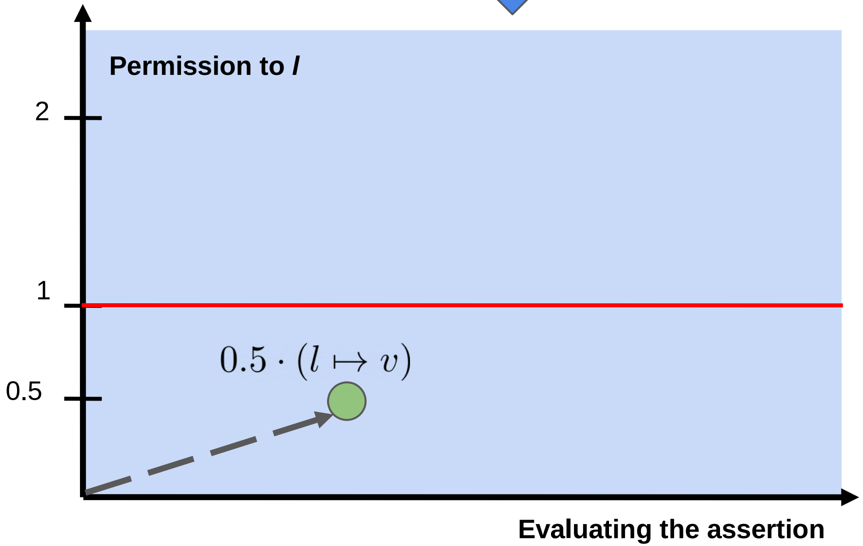
$$0.5 \cdot (l \mapsto v * l \mapsto v) \not\models (l \mapsto v * l \mapsto v)^{0.5}$$

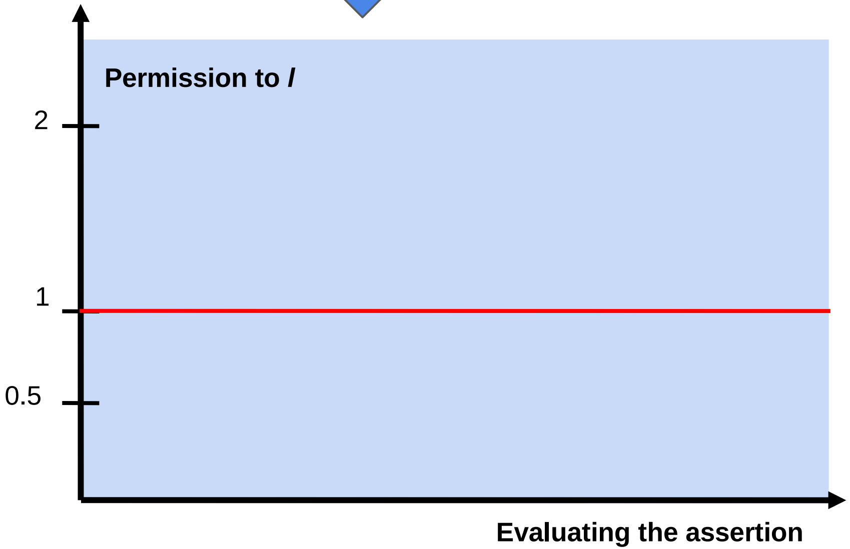
Syntactic

$$0.5 \cdot (l \mapsto v) * 0.5 \cdot (l \mapsto v)$$

Syntactic

Syntactic





Unbounded separation logic: Intuition

Semantic

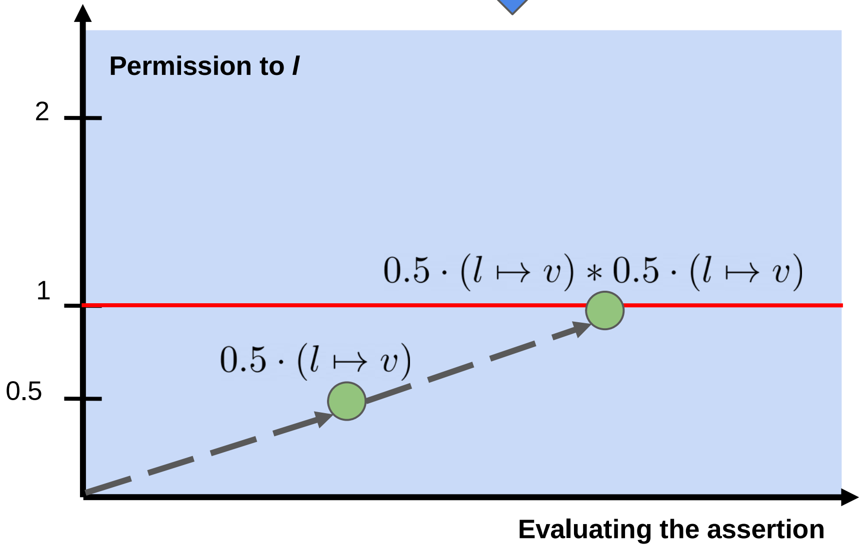
$$0.5 \cdot (l \mapsto v * l \mapsto v) \not\models (l \mapsto v * l \mapsto v)^{0.5}$$

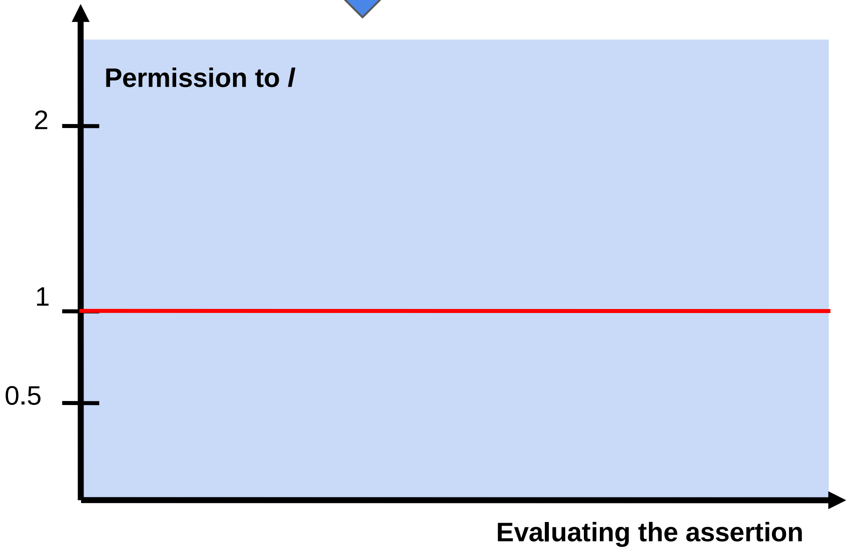
Syntactic

$$0.5 \cdot (l \mapsto v) * 0.5 \cdot (l \mapsto v)$$

Syntactic

Syntactic

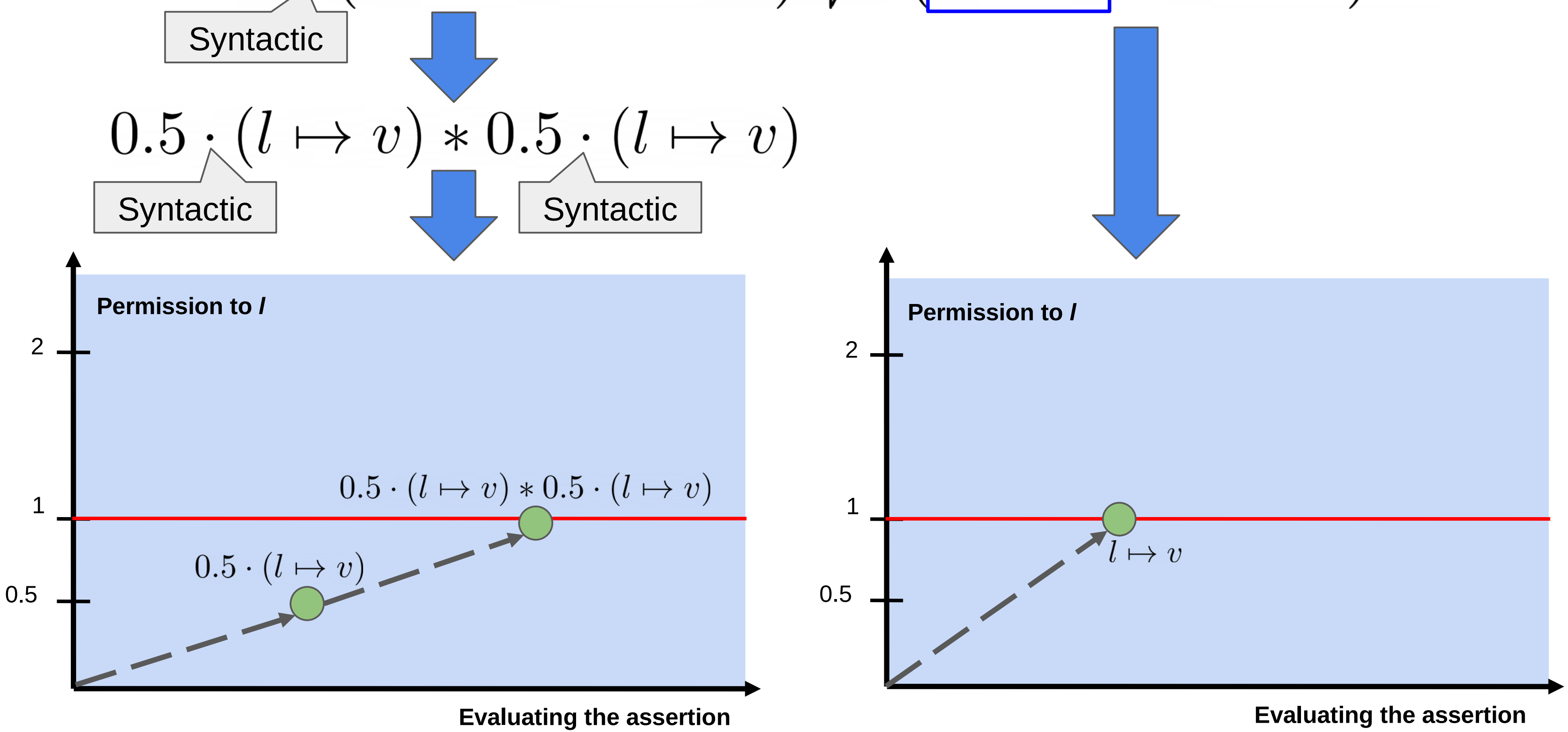




Unbounded separation logic: Intuition

Semantic

$$0.5 \cdot (l \mapsto v * l \mapsto v) \not\models (l \mapsto v * l \mapsto v)^{0.5}$$



Unbounded separation logic: Intuition

$$0.5 \cdot (l \mapsto v * l \mapsto v) \not\models (l \mapsto v * l \mapsto v)^{0.5}$$

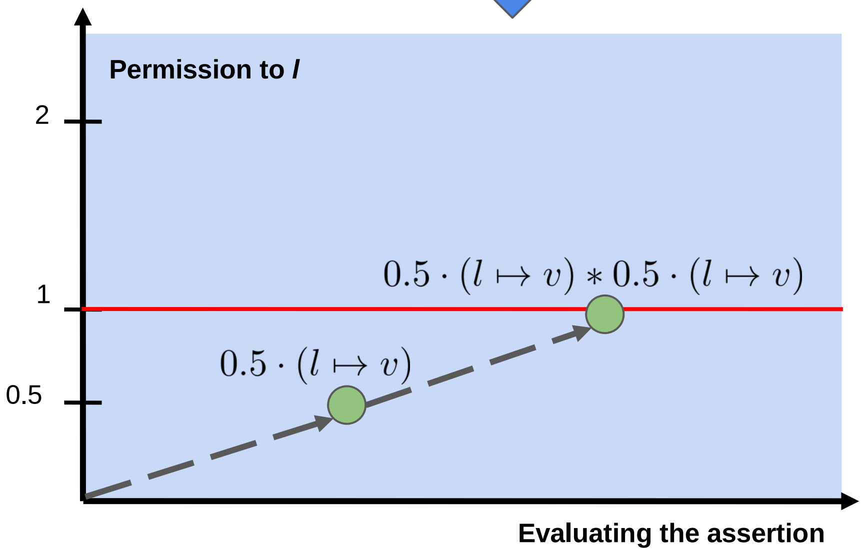
Syntactic

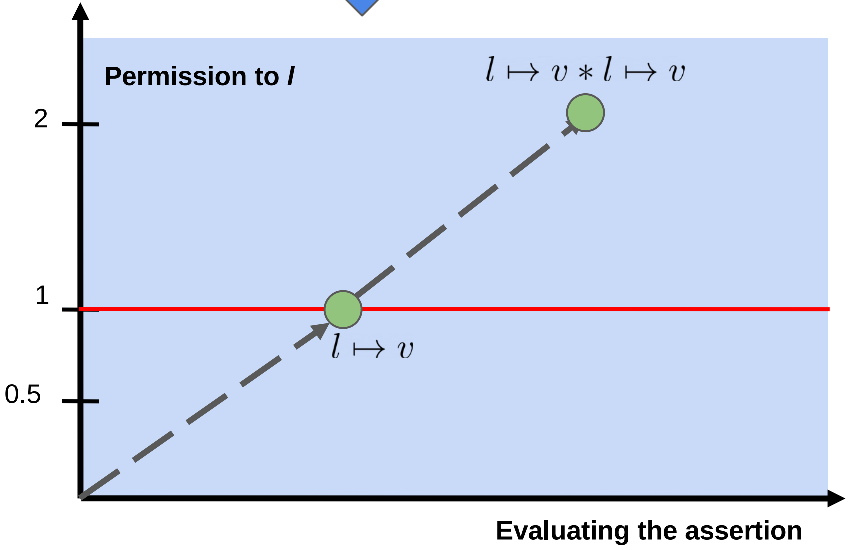
Semantic

$$0.5 \cdot (l \mapsto v) * 0.5 \cdot (l \mapsto v)$$

Syntactic

Syntactic





Unbounded separation logic: Intuition

$$0.5 \cdot (l \mapsto v * l \mapsto v) \not\models (l \mapsto v * l \mapsto v)^{0.5}$$

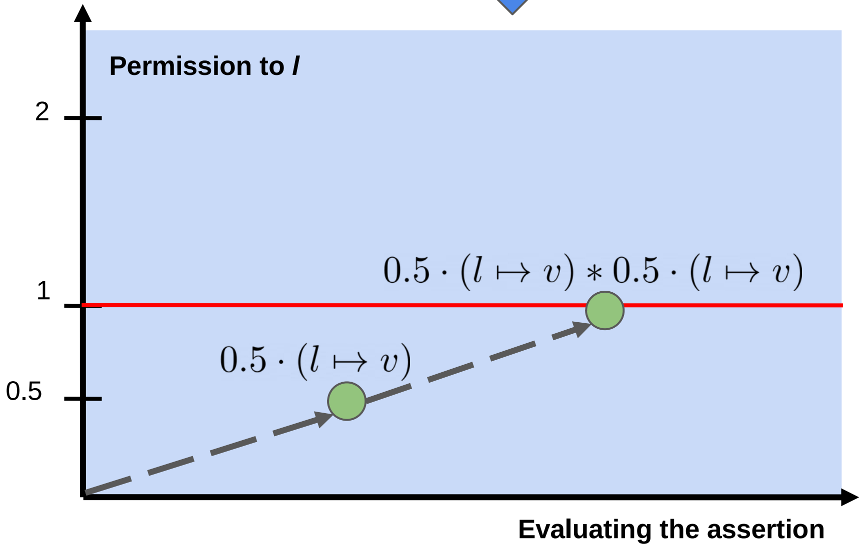
Syntactic

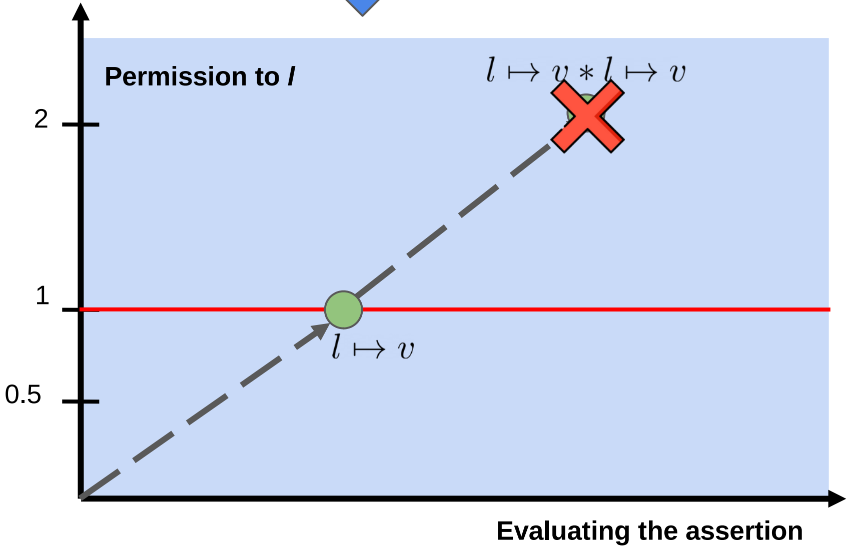
Semantic

$$0.5 \cdot (l \mapsto v) * 0.5 \cdot (l \mapsto v)$$

Syntactic

Syntactic





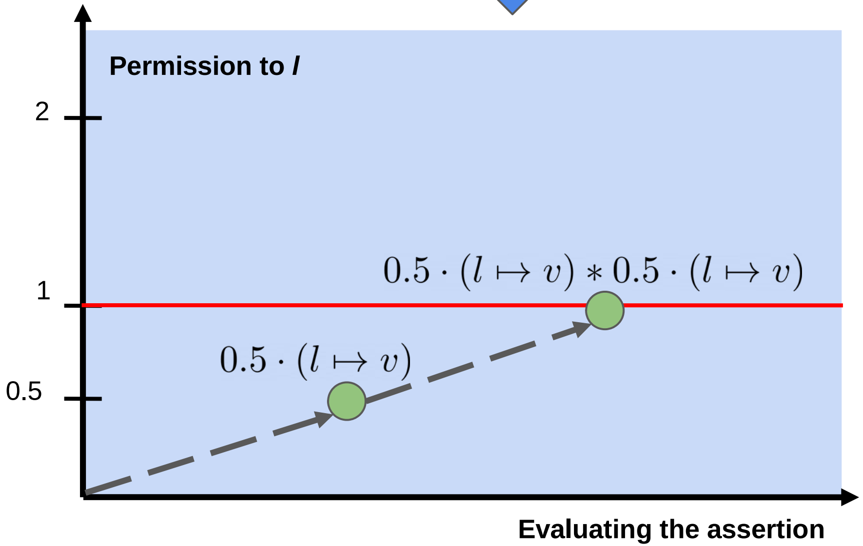
Unbounded separation logic: Intuition

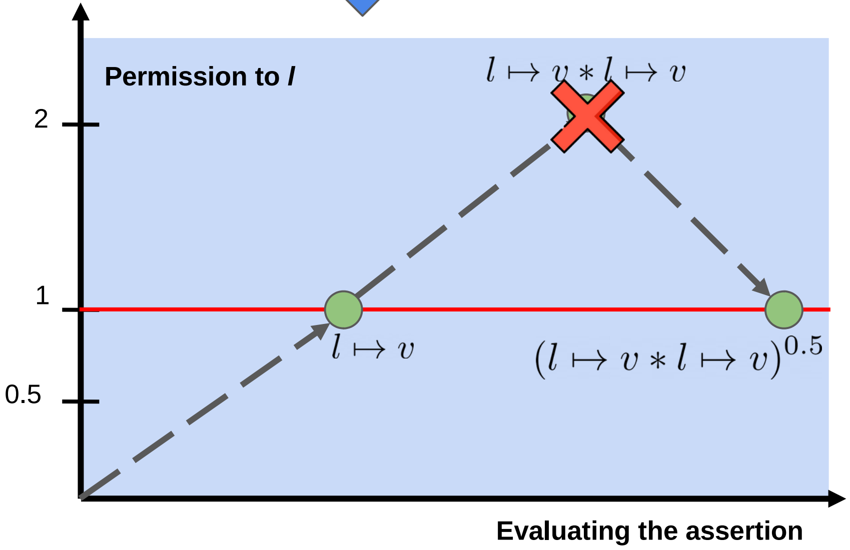
$$0.5 \cdot (l \mapsto v * l \mapsto v) \not\models (l \mapsto v * l \mapsto v)^{0.5}$$

Syntactic | Semantic

$$0.5 \cdot (l \mapsto v) * 0.5 \cdot (l \mapsto v)$$

Syntactic | Syntactic





Unbounded separation logic: Intuition

Semantic

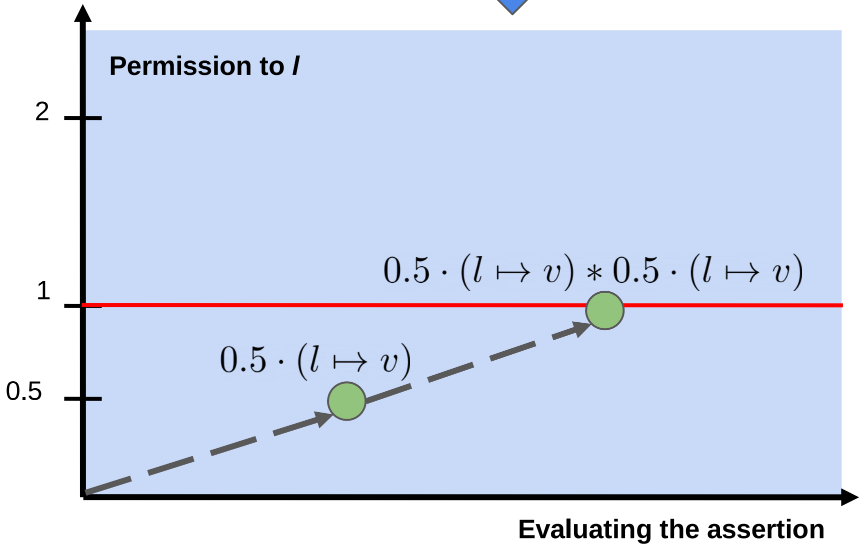
$$0.5 \cdot (l \mapsto v * l \mapsto v) \not\models (l \mapsto v * l \mapsto v)^{0.5}$$

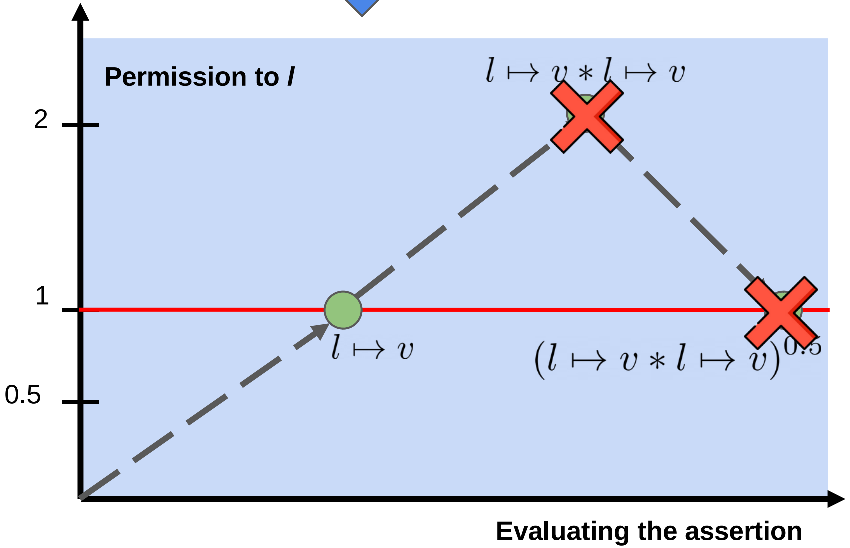
Syntactic

$$0.5 \cdot (l \mapsto v) * 0.5 \cdot (l \mapsto v)$$

Syntactic

Syntactic





Unbounded separation logic: Intuition

$$0.5 \cdot (l \mapsto v * l \mapsto v) \not\models (l \mapsto v * l \mapsto v)^{0.5}$$

Syntactic

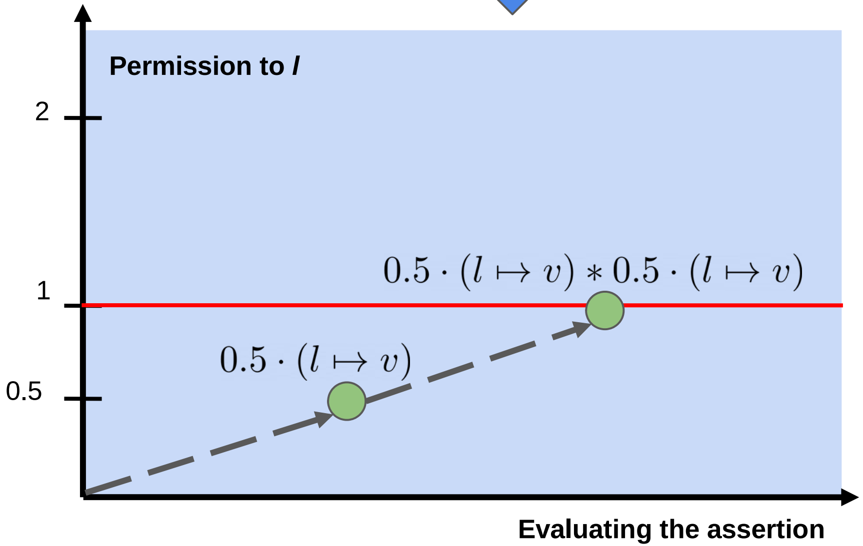
Semantic

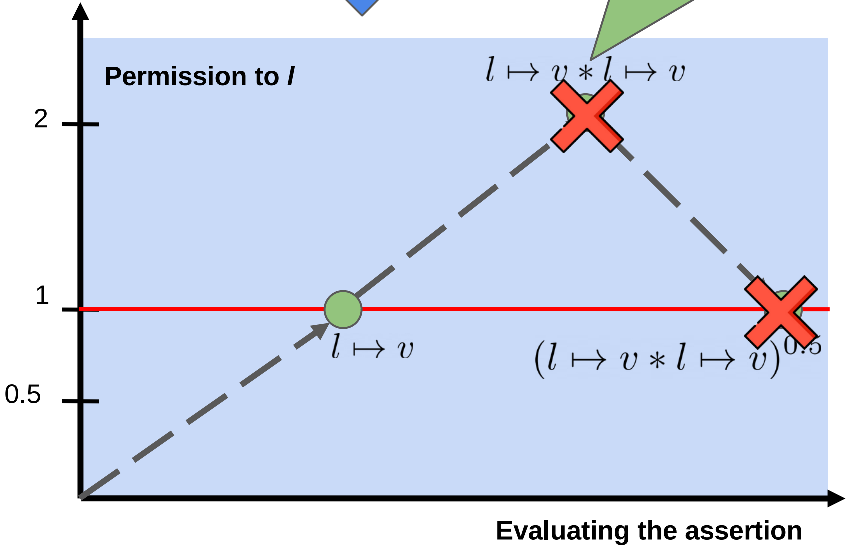
$$0.5 \cdot (l \mapsto v) * 0.5 \cdot (l \mapsto v)$$

Syntactic

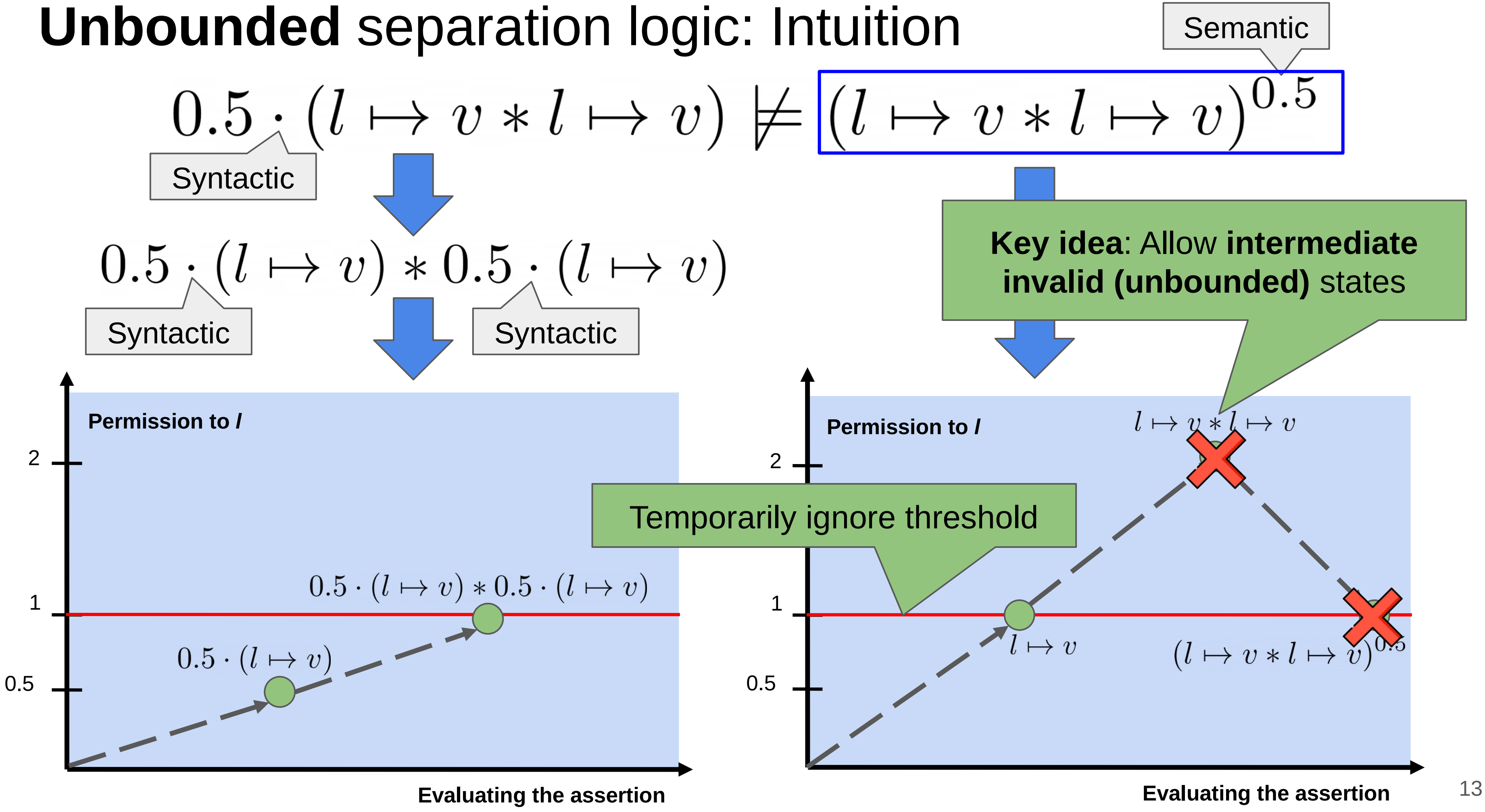
Syntactic

Key idea: Allow **intermediate invalid (unbounded)** states





Unbounded separation logic: Intuition



Unbounded separation logic (simplified)

Unbounded separation logic (simplified)

1. *Temporarily* allow **unbounded** states in the assertion logic

Unbounded separation logic (simplified)

1. *Temporarily* allow **unbounded** states in the assertion logic

2. Reimpose boundedness at statement boundaries

Unbounded separation logic (simplified)

1. *Temporarily* allow **unbounded** states in the assertion logic

$$BoundedState \triangleq Locations \rightharpoonup Value \times (\mathbb{Q} \cap (0, 1])$$

2. Reimpose boundedness at statement boundaries

Unbounded separation logic (simplified)

1. *Temporarily* allow **unbounded** states in the assertion logic

$$BoundedState \triangleq Locations \rightharpoonup Value \times (\mathbb{Q} \cap (0,1])$$

2. Reimpose boundedness at statement boundaries

Unbounded separation logic (simplified)

1. *Temporarily* allow **unbounded** states in the assertion logic

$$BoundedState \triangleq Locations \rightharpoonup Value \times \boxed{(\mathbb{Q} \cap (0,1])}$$

$$State \triangleq Locations \rightharpoonup Value \times \boxed{\mathbb{Q}^+}$$

2. Reimpose boundedness at statement boundaries

Unbounded separation logic (simplified)

1. *Temporarily* allow **unbounded** states in the assertion logic

$$\mathit{BoundedState} \triangleq \mathit{Locations} \rightharpoonup \mathit{Value} \times \boxed{(\mathbb{Q} \cap (0,1])}$$

$$\mathit{State} \triangleq \mathit{Locations} \rightharpoonup \mathit{Value} \times \boxed{\mathbb{Q}^{+}}$$

2. Reimpose boundedness at statement boundaries

$$\{P\}C\{Q\} \iff (\forall h.\, h \models P \Rightarrow \ldots)$$

Unbounded separation logic (simplified)

1. *Temporarily* allow **unbounded** states in the assertion logic

$$BoundedState \triangleq Locations \rightharpoonup Value \times \boxed{(\mathbb{Q} \cap (0, 1])}$$

$$\Downarrow$$

$$State \triangleq Locations \rightharpoonup Value \times \boxed{\mathbb{Q}^+}$$

2. Reimpose boundedness at statement boundaries

$$\{P\}C\{Q\} \iff (\forall h.\, h \models P \Rightarrow \ldots)$$

$$\Downarrow$$

$$\{P\}C\{Q\} \iff (\forall h.\, h \models P \land \boxed{h \in BoundedState} \Rightarrow \ldots)$$

Theoretical foundation for the syntactic multiplication

Theorem: In unbounded separation logic,

$$h \models \pi \cdot A \iff (\exists h_A .\, h_A \models A \land h = \pi \odot h_A)$$

Syntactic

Theoretical foundation for the syntactic multiplication

Theorem: In unbounded separation logic,

$$h \models \pi \cdot A \iff (\exists h_A .\, h_A \models A \land h = \pi \odot h_A)$$

Syntactic

Same definition as the semantic multiplication. The difference is in the **state model** (*bounded* vs. *unbounded*).

What about the frame rule?

$$\frac{\{P\}\ C\ \{Q\} \quad mod(C) \cap fv(R) = \emptyset}{\{P * R\}\ C\ \{Q * R\}}\ (Frame)$$

What about the frame rule?

$$\frac{\{P\}\ C\ \{Q\} \qquad mod(C) \cap fv(R) = \emptyset}{\{P * R\}\ C\ \{Q * R\}}\ (Frame)$$

Theorem: The frame rule holds in unbounded separation logic.

What about the frame rule?

$$\frac{\{P\}\ C\ \{Q\} \quad mod(C) \cap fv(R) = \emptyset}{\{P * R\}\ C\ \{Q * R\}}\ (\mathit{Frame})$$

2. Reimpose boundedness at statement boundaries

Theorem: The frame rule holds in unbounded separation logic.

Factorisation and distribution in unbounded separation logic

$$\frac{}{\alpha \cdot (\beta \cdot A) \equiv (\alpha \times \beta) \cdot A}\ (\mathit{DotDot}) \qquad \frac{}{\pi \cdot (\exists x.\, A) \equiv \exists x.\, (\pi \cdot A)}\ (\mathit{DotExists})$$

$$\frac{}{\pi \cdot (A \mathrel{-\!\!*} B) \equiv (\pi \cdot A) \mathrel{-\!\!*} (\pi \cdot B)}\ (\mathit{DotWand}) \qquad \frac{}{\pi \cdot (A \Rightarrow B) \equiv (\pi \cdot A) \Rightarrow (\pi \cdot B)}\ (\mathit{DotImp})$$

$$\frac{}{A \models B \Longleftrightarrow \pi \cdot A \models \pi \cdot B}\ (\mathit{DotPos}) \qquad \frac{}{\pi \cdot (\forall x.\, A) \equiv \forall x.\, (\pi \cdot A)}\ (\mathit{DotForall})$$

$$\frac{}{\pi \cdot (A \wedge B) \equiv (\pi \cdot A) \wedge (\pi \cdot B)}\ (\mathit{DotAnd}) \qquad \frac{}{\pi \cdot (A \vee B) \equiv (\pi \cdot A) \vee (\pi \cdot B)}\ (\mathit{DotOr}) \qquad \frac{}{1 \cdot A \equiv A}\ (\mathit{DotFull})$$

$$\frac{\mathit{pure}(A)}{\pi \cdot A \equiv A}\ (\mathit{DotPure}) \qquad \frac{}{\pi \cdot (A * B) \equiv (\pi \cdot A) * (\pi \cdot B)}\ (\mathit{DotStar}) \qquad \frac{}{(\alpha + \beta) \cdot A \models (\alpha \cdot A) * (\beta \cdot A)}\ (\mathit{Split})$$

Factorisation and distribution in unbounded separation logic

$$\frac{}{\alpha \cdot (\beta \cdot A) \equiv (\alpha \times \beta) \cdot A}\ (DotDot) \qquad \frac{}{\pi \cdot (\exists x.\, A) \equiv \exists x.\, (\pi \cdot A)}\ (DotExists)$$

$$\frac{}{\pi \cdot (A \mathrel{-\!\!*} B) \equiv (\pi \cdot A) \mathrel{-\!\!*} (\pi \cdot B)}\ (DotWand) \qquad \frac{}{\pi \cdot (A \Rightarrow B) \equiv (\pi \cdot A) \Rightarrow (\pi \cdot B)}\ (DotImp)$$

$$\frac{}{A \models B \Longleftrightarrow \pi \cdot A \models \pi \cdot B}\ (DotPos) \qquad \frac{}{\pi \cdot (\forall x.\, A) \equiv \forall x.\, (\pi \cdot A)}\ (DotForall)$$

$$\frac{}{\pi \cdot (A \wedge B) \equiv (\pi \cdot A) \wedge (\pi \cdot B)}\ (DotAnd) \qquad \frac{}{\pi \cdot (A \vee B) \equiv (\pi \cdot A) \vee (\pi \cdot B)}\ (DotOr) \qquad \frac{}{1 \cdot A \equiv A}\ (DotFull)$$

$$\frac{pure(A)}{\pi \cdot A \equiv A}\ (DotPure) \qquad \frac{}{\pi \cdot (A * B) \equiv (\pi \cdot A) * (\pi \cdot B)}\ (DotStar) \qquad \frac{}{(\alpha + \beta) \cdot A \models (\alpha \cdot A) * (\beta \cdot A)}\ (Split)$$

Factorisation and distribution in unbounded separation logic

$$\frac{}{\alpha \cdot (\beta \cdot A) \equiv (\alpha \times \beta) \cdot A}\ (DotDot) \qquad \frac{}{\pi \cdot (\exists x.\, A) \equiv \exists x.\, (\pi \cdot A)}\ (DotExists)$$

$$\frac{}{\pi \cdot (A \twoheadrightarrow B) \equiv (\pi \cdot A) \twoheadrightarrow (\pi \cdot B)}\ (DotWand) \qquad \frac{}{\pi \cdot (A \Rightarrow B) \equiv (\pi \cdot A) \Rightarrow (\pi \cdot B)}\ (DotImp)$$

$$\frac{}{A \models B \iff \pi \cdot A \models \pi \cdot B}\ (DotPos) \qquad \frac{}{\pi \cdot (\forall x.\, A) \equiv \forall x.\, (\pi \cdot A)}\ (DotForall)$$

$$\frac{}{\pi \cdot (A \wedge B) \equiv (\pi \cdot A) \wedge (\pi \cdot B)}\ (DotAnd) \qquad \frac{}{\pi \cdot (A \vee B) \equiv (\pi \cdot A) \vee (\pi \cdot B)}\ (DotOr) \qquad \frac{}{1 \cdot A \equiv A}\ (DotFull)$$

$$\frac{pure(A)}{\pi \cdot A \equiv A}\ (DotPure) \qquad \frac{}{\pi \cdot (A * B) \equiv (\pi \cdot A) * (\pi \cdot B)}\ (DotStar) \qquad \frac{}{(\alpha + \beta) \cdot A \models (\alpha \cdot A) * (\beta \cdot A)}\ (Split)$$

Factorisation and distribution in unbounded separation logic

$$\frac{}{\alpha \cdot (\beta \cdot A) \equiv (\alpha \times \beta) \cdot A}\ (DotDot) \qquad \frac{}{\pi \cdot (\exists x.\, A) \equiv \exists x.\, (\pi \cdot A)}\ (DotExists)$$

$$\frac{}{\pi \cdot (A \mathrel{-\!\!*} B) \equiv (\pi \cdot A) \mathrel{-\!\!*} (\pi \cdot B)}\ (DotWand) \qquad \frac{}{\pi \cdot (A \Rightarrow B) \equiv (\pi \cdot A) \Rightarrow (\pi \cdot B)}\ (DotImp)$$

$$\frac{}{A \models B \iff \pi \cdot A \models \pi \cdot B}\ (DotPos) \qquad \frac{}{\pi \cdot (\forall x.\, A) \equiv \forall x.\, (\pi \cdot A)}\ (DotForall)$$

$$\frac{}{\pi \cdot (A \wedge B) \equiv (\pi \cdot A) \wedge (\pi \cdot B)}\ (DotAnd) \qquad \frac{}{\pi \cdot (A \vee B) \equiv (\pi \cdot A) \vee (\pi \cdot B)}\ (DotOr) \qquad \frac{}{1 \cdot A \equiv A}\ (DotFull)$$

$$\frac{pure(A)}{\pi \cdot A \equiv A}\ (DotPure) \qquad \frac{}{\pi \cdot (A * B) \equiv (\pi \cdot A) * (\pi \cdot B)}\ (DotStar) \qquad \frac{}{(\alpha + \beta) \cdot A \models (\alpha \cdot A) * (\beta \cdot A)}\ (Split)$$

Factorisation and distribution in unbounded separation logic

The syntactic multiplication can be extended to support fractional magic wands

$$\frac{}{\ldots \cdot (\exists x.\, A) \equiv \exists x.\, (\pi \cdot A)}\ (DotExists)$$

$$\frac{}{\pi \cdot (A \mathrel{-\!\!*} B) \equiv (\pi \cdot A) \mathrel{-\!\!*} (\pi \cdot B)}\ (DotWand) \qquad \frac{}{\pi \cdot (A \Rightarrow B) \equiv (\pi \cdot A) \Rightarrow (\pi \cdot B)}\ (DotImp)$$

$$\frac{}{A \models B \Longleftrightarrow \pi \cdot A \models \pi \cdot B}\ (DotPos) \qquad \frac{}{\pi \cdot (\forall x.\, A) \equiv \forall x.\, (\pi \cdot A)}\ (DotForall)$$

$$\frac{}{\pi \cdot (A \wedge B) \equiv (\pi \cdot A) \wedge (\pi \cdot B)}\ (DotAnd) \qquad \frac{}{\pi \cdot (A \vee B) \equiv (\pi \cdot A) \vee (\pi \cdot B)}\ (DotOr) \qquad \frac{}{1 \cdot A \equiv A}\ (DotFull)$$

$$\frac{pure(A)}{\pi \cdot A \equiv A}\ (DotPure) \qquad \frac{}{\pi \cdot (A * B) \equiv (\pi \cdot A) * (\pi \cdot B)}\ (DotStar) \qquad \frac{}{(\alpha + \beta) \cdot A \models (\alpha \cdot A) * (\beta \cdot A)}\ (Split)$$

Using fractional resources

```
method processTree(x: Ref) {
  {tree(x)^π}
  if (x != null) {
    {tree(x)^π * x ≠ null}
    {(tree(x)^(π/2) * x ≠ null) * (tree(x)^(π/2) * x ≠ null)}
        {tree(x)^(π/2) * x ≠ null}
        {x.val ↦^(π/2) _ * ... * tree(x_l)^(π/2) * tree(x_r)^(π/2)}
        print(x.val)
        processTree(x.left)
        processTree(x.right)

        {x.val ↦^(π/2) _ * ... * tree(x_l)^(π/2) * tree(x_r)^(π/2)}
        {tree(x)^(π/2)}
    {tree(x)^(π/2) * tree(x)^(π/2)}
  }
  {tree(x)^π}
}
```

1. Split

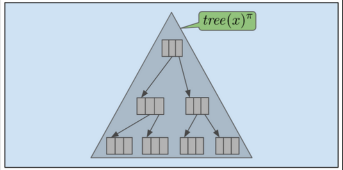
2. Distribute

3. Factorise

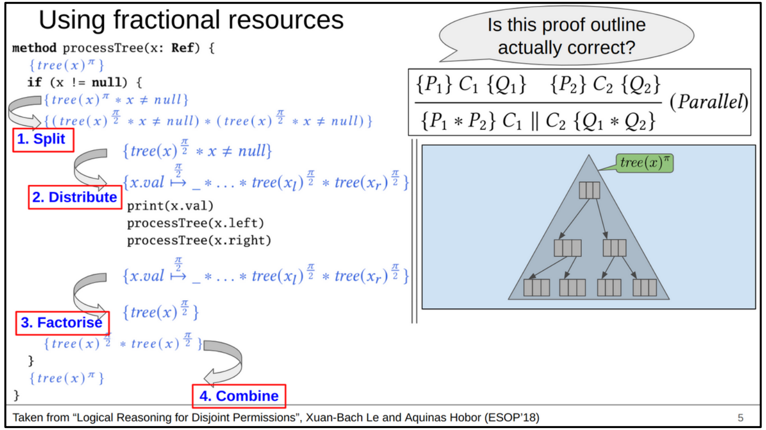
4. Combine

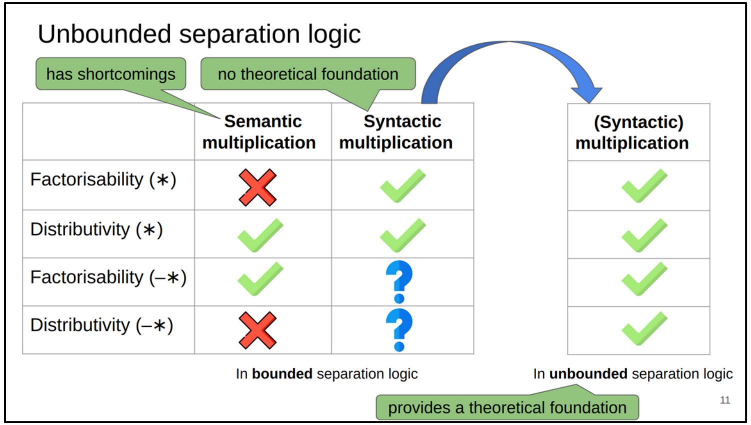
Is this proof outline actually correct?

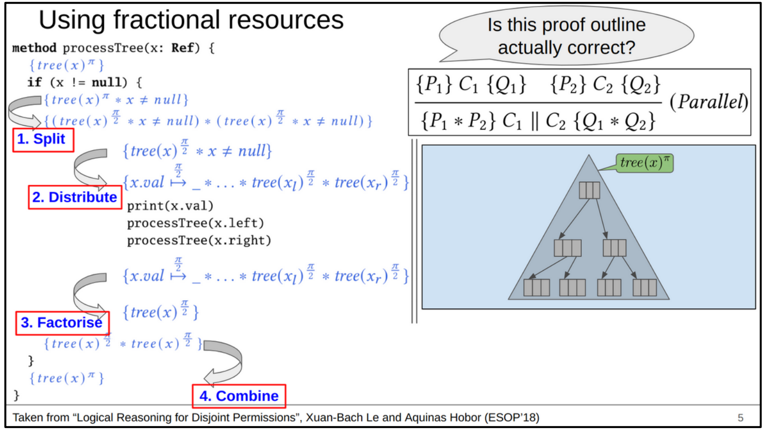
$$\frac{\{P_1\}\ C_1\ \{Q_1\} \quad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \parallel C_2\ \{Q_1 * Q_2\}}\ (Parallel)$$

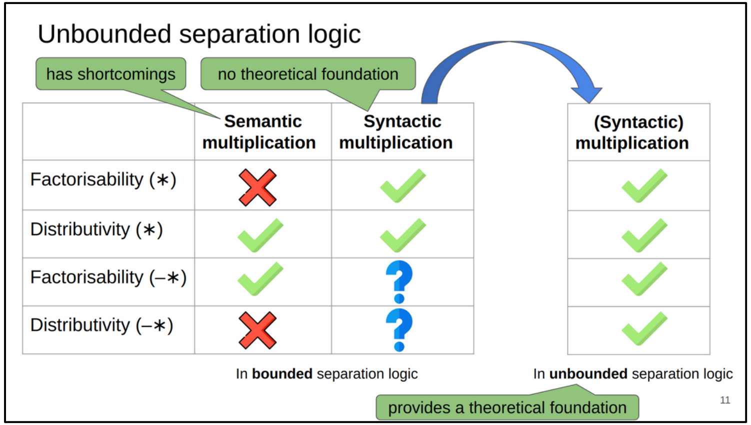


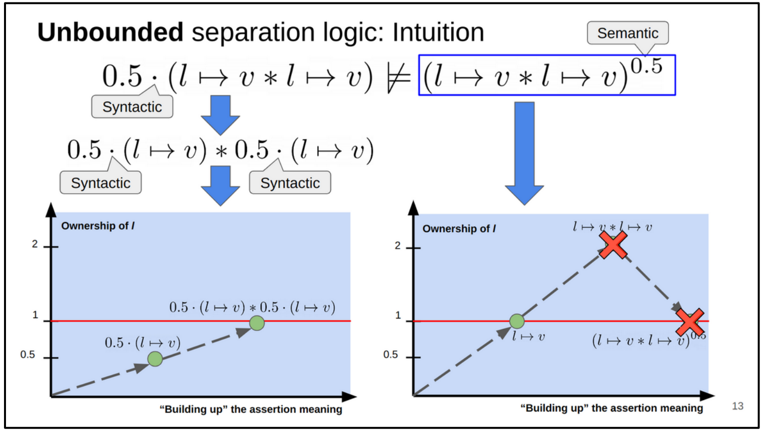
Taken from "Logical Reasoning for Disjoint Permissions", Xuan-Bach Le and Aquinas Hobor (ESOP'18) 5

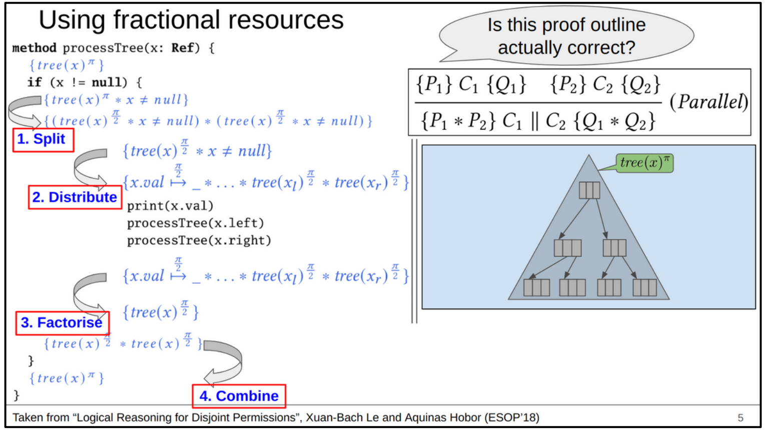

Using fractional resources
Is this proof outline actually correct?
method processTree(x: Ref) {
if (x != null) {
1. Split
2. Distribute
print(x.val)
processTree(x.left)
processTree(x.right)
3. Factorise
4. Combine
(Parallel)
Taken from "Logical Reasoning for Disjoint Permissions", Xuan-Bach Le and Aquinas Hobor (ESOP'18)
5

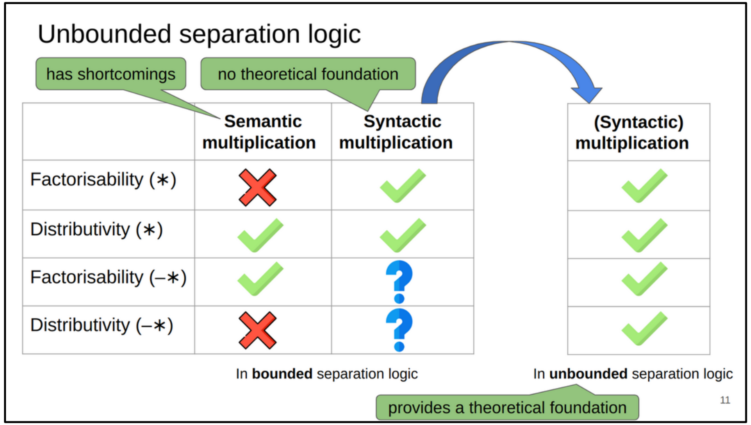

Unbounded separation logic
has shortcomings
no theoretical foundation
Semantic multiplication
Syntactic multiplication
(Syntactic) multiplication
Factorisability (∗)
Distributivity (∗)
Factorisability (−∗)
Distributivity (−∗)
In bounded separation logic
In unbounded separation logic
provides a theoretical foundation
11

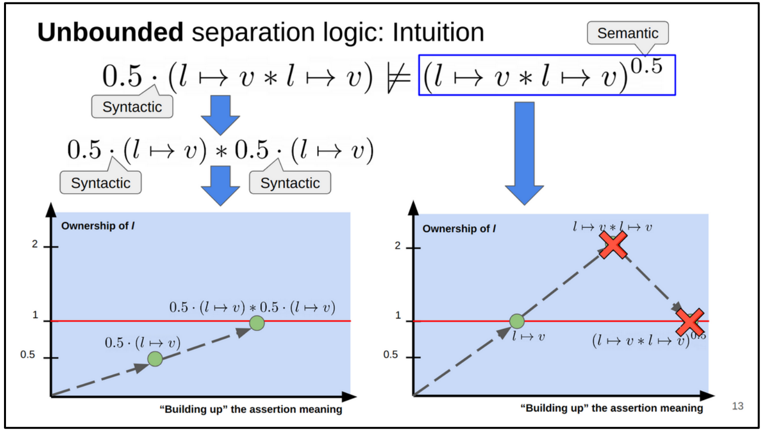

Using fractional resources
Is this proof outline actually correct?
method processTree(x: Ref) {
if (x != null) {
1. Split
2. Distribute
print(x.val)
processTree(x.left)
processTree(x.right)
3. Factorise
4. Combine
(Parallel)
Taken from "Logical Reasoning for Disjoint Permissions", Xuan-Bach Le and Aquinas Hobor (ESOP'18)
5


Unbounded separation logic
has shortcomings
no theoretical foundation
Semantic multiplication
Syntactic multiplication
(Syntactic) multiplication
Factorisability (∗)
Distributivity (∗)
Factorisability (−∗)
Distributivity (−∗)
In bounded separation logic
In unbounded separation logic
provides a theoretical foundation
11


Unbounded separation logic: Intuition
Semantic
Syntactic
Syntactic
Syntactic
Ownership of l
"Building up" the assertion meaning
Ownership of l
"Building up" the assertion meaning
13



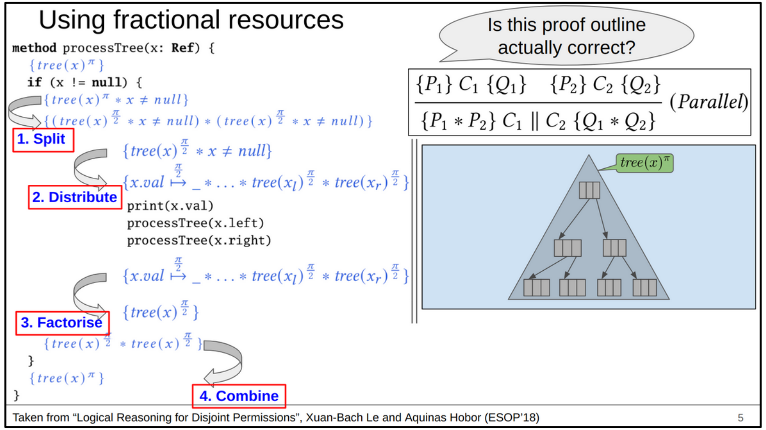


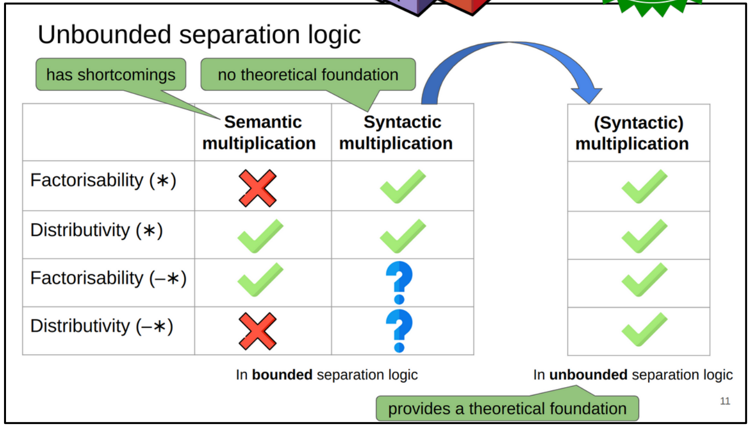


More in the paper:

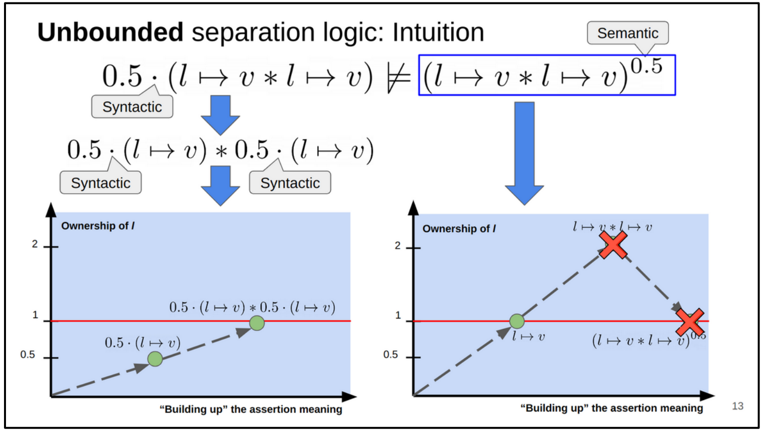
- combinability (step 4)
- reasoning principles for (co)inductive predicates
- unbounded separation logic as a formal foundation for automatic verifiers

Thank you for your attention!





	Semantic multiplication	Syntactic multiplication	(Syntactic) multiplication
Factorisability (∗)	✗	✓	✓
Distributivity (∗)	✓	✓	✓
Factorisability (−∗)	✓	?	✓
Distributivity (−∗)	✗	?	✓
	In **bounded** separation logic		In **unbounded** separation logic



More in the paper:

- ❖ combinability (step 4)
- ❖ reasoning principles for (co)inductive predicates
- ❖ unbounded separation logic as a formal foundation for automatic verifiers