Ribbon Proofs for Separation Logic

John Wickerson
University of Cambridge

Dublin Concurrency Workshop, 15 April 2011

Talk outline

1. The problem
2. Towards a solution
3. A big proof
4. Fun with quantifiers
5. What about concurrency?

```
mchunkptr b, p;
idx += ~smallbits & 1; /* Uses next bin if idx empty */
```

$$\left\{\begin{array}{l}\exists\{U_i \mid i\in[0,63)\}, n.\ \mathit{arena}(A_a \uplus (\biguplus_{i=0}^{64}. U_i)_u) * \mathtt{least\_addr} = 5w \\ * \ nw = \lceil \mathtt{bytes} \rceil_w * 8\mathtt{idx} \geq (n+1)w * 2 \leq \mathtt{idx} < 32 * \mathtt{smallmap}_{[\mathtt{idx}]} = 1 \\ * \ \mathop{\ast}_{i=0}^{32}. \mathit{smallbin}_i(U_i) * \mathop{\ast}_{i=0}^{32}. \mathit{treebin}_i(U_{i+32})\end{array}\right\}$$

```
b = smallbin_at(gm, idx);
```

$$\left\{\begin{array}{l}\exists\{U_i \mid i\in[0,63)\}, n.\ \mathit{arena}(A_a \uplus (\biguplus_{i=0}^{64}. U_i)_u) * \mathtt{least\_addr} = 5w \\ * \ nw = \lceil \mathtt{bytes} \rceil_w * 8\mathtt{idx} \geq (n+1)w * 2 \leq \mathtt{idx} < 32 * \mathtt{smallmap}_{[\mathtt{idx}]} = 1 \\ * \ \mathtt{b} = \mathtt{smallbins} + 8\mathtt{idx} * \mathit{bin}(|\mathtt{idx}|, \mathtt{b}, U_{\mathtt{idx}}) * U_{\mathtt{idx}} \neq \{\} \\ * \ \mathop{\ast}_{i\in[0..32)-\mathtt{idx}}. \mathit{smallbin}_i(U_i) * \mathop{\ast}_{i=0}^{32}. \mathit{treebin}_i(U_{i+32})\end{array}\right\}$$

```
// rename U_idx to U_idx++[p+2w->8idx-1w]
```

$$\left\{\begin{array}{l}\exists\{U_i \mid i\in[0,63)\}, p, n.\ \mathit{arena}(A_a \uplus (\biguplus_{i=0}^{64}. U_i)_u \uplus \{p+2w \mapsto_u 8\mathtt{idx} - 1w\}) \\ * \ \mathtt{least\_addr} = 5w * nw = \lceil \mathtt{bytes} \rceil_w * 8\mathtt{idx} \geq (n+1)w * 2 \leq \mathtt{idx} < 32 \\ * \ \mathtt{smallmap}_{[\mathtt{idx}]} = 1 * \mathtt{b} = \mathtt{smallbins} + 8\mathtt{idx} \\ * \ \mathtt{b} \xmapsto{\mathsf{fd}} p * p \xmapsto{\mathsf{bk}} \mathtt{b} * (\mathit{bnode}\,|\mathtt{idx}|)^*(p, \mathtt{b}, U_{\mathtt{idx}} \uplus \{p+2w \mapsto 8\mathtt{idx} - 1w\}) \\ * \ \mathop{\ast}_{i\in[0..32)-\mathtt{idx}}. \mathit{smallbin}_i(U_i) * \mathop{\ast}_{i=0}^{32}. \mathit{treebin}_i(U_{i+32})\end{array}\right\}$$

```
p = b->fd;
```

$$\left\{\begin{array}{l}\exists\{U_i \mid i\in[0,63)\}, n, F.\ \mathit{arena}(A_a \uplus (\biguplus_{i=0}^{64}. U_i)_u \uplus \{\mathtt{p}+2w \mapsto_u 8\mathtt{idx} - 1w\}) \\ * \ \mathtt{least\_addr} = 5w * nw = \lceil \mathtt{bytes} \rceil_w * 8\mathtt{idx} \geq (n+1)w * 2 \leq \mathtt{idx} < 32 \\ * \ \mathtt{smallmap}_{[\mathtt{idx}]} = 1 * \mathtt{b} = \mathtt{smallbins} + 8\mathtt{idx} \\ * \ \mathtt{b} \xmapsto{\mathsf{fd}} \mathtt{p} * \mathtt{p} \xmapsto{\mathsf{bk}} \mathtt{b} * \frac{1}{2}(\mathtt{p} \xmapsto{\mathsf{size}} 8\mathtt{idx}) * \mathtt{p} \xmapsto{\mathsf{fd}} F * F \xmapsto{\mathsf{bk}} \mathtt{p} * (\mathit{bnode}\,|\mathtt{idx}|)^*(F, \mathtt{b}, U_{\mathtt{idx}}) \\ * \ \mathop{\ast}_{i\in[0..32)-\mathtt{idx}}. \mathit{smallbin}_i(U_i) * \mathop{\ast}_{i=0}^{32}. \mathit{treebin}_i(U_{i+32})\end{array}\right\}$$

```
//assert(chunksize(p) == small_index2size(idx));
unlink_first_small_chunk(gm, b, p, idx);
```

$$\left\{\begin{array}{l}\exists\{U_i \mid i\in[0,63)\}, n.\ \mathit{arena}(A_a \uplus (\biguplus_{i=0}^{64}. U_i)_u \uplus \{\mathtt{p}+2w \mapsto_u 8\mathtt{idx} - 1w\}) \\ * \ \mathtt{least\_addr} = 5w * nw = \lceil \mathtt{bytes} \rceil_w * 8\mathtt{idx} \geq (n+1)w * 2 \leq \mathtt{idx} < 32 \\ * \ \frac{1}{2}(\mathtt{p} \xmapsto{\mathsf{size}} 8\mathtt{idx}) * \mathtt{p} \xmapsto{\mathsf{fd}} \_ * \mathtt{p} \xmapsto{\mathsf{bk}} \_ * \mathop{\ast}_{i=0}^{32}. \mathit{smallbin}_i(U_i) * \mathop{\ast}_{i=0}^{32}. \mathit{treebin}_i(U_{i+32})\end{array}\right\}$$

$$\left\{\begin{array}{l}\exists\{U_i \mid i\in[0,63)\}, B_1, B_2, n.\ \mathit{coallesced}(A_a \uplus (\biguplus_{i=0}^{64}. U_i)_u \uplus \{\mathtt{p}+2w \mapsto_u 8\mathtt{idx} - 1w\}) \\ * \ \mathtt{start} \xmapsto{\mathsf{prevfoot}} \_ * \mathtt{start} \xmapsto{\mathsf{pinuse}} 1 * \mathit{ublock}(\mathtt{top}, \mathtt{top} + \mathtt{topsize}, \_) \\ * \ \mathit{block}^*(\mathtt{start}, \mathtt{p}, B_1) * \mathit{ublock}(\mathtt{p}, \mathtt{p} + 8\mathtt{idx}, \{\mathtt{p}+2w \mapsto_u 8\mathtt{idx} - 1w\}) \\ * \ \mathit{block}^*(\mathtt{p} + 8\mathtt{idx}, \mathtt{top}, B_2) * B_1 \uplus B_2 = A_a \uplus (\biguplus_{i=0}^{64}. U_i)_u \\ * \ \mathtt{least\_addr} = 5w * nw = \lceil \mathtt{bytes} \rceil_w * 8\mathtt{idx} \geq (n+1)w * 2 \leq \mathtt{idx} < 32 \\ * \ \frac{1}{2}(\mathtt{p} \xmapsto{\mathsf{size}} 8\mathtt{idx}) * \mathtt{p} \xmapsto{\mathsf{fd}} \_ * \mathtt{p} \xmapsto{\mathsf{bk}} \_ * \mathop{\ast}_{i=0}^{32}. \mathit{smallbin}_i(U_i) * \mathop{\ast}_{i=0}^{32}. \mathit{treebin}_i(U_{i+32})\end{array}\right\}$$

A simple program

$$\{x \mapsto 2 * y \mapsto 3 * z \mapsto 4\}$$

$$[x] := 3$$

$$\{x \mapsto 3 * y \mapsto 3 * z \mapsto 4\}$$

$$[y] := 4$$

$$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 4\}$$

$$[z] := 5$$

$$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 5\}$$

A simple program

$\{x \mapsto 2 * y \mapsto 3 * z \mapsto 4\}$

$\{x \mapsto 2\}$

$[x] := 3$

$\{x \mapsto 3\}$

$\{x \mapsto 3 * y \mapsto 3 * z \mapsto 4\}$

$\{y \mapsto 3\}$

$[y] := 4$

$\{y \mapsto 4\}$

$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 4\}$

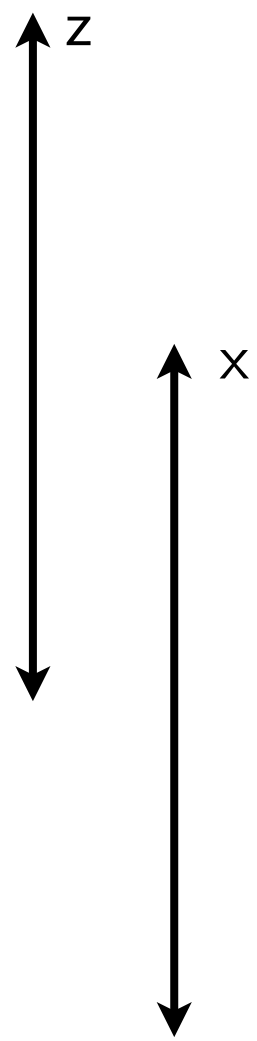
$\{z \mapsto 4\}$

$[z] := 5$

$\{z \mapsto 5\}$

$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 5\}$

A simple program



$$\{x \mapsto 2 * y \mapsto 3 * z \mapsto 4\}$$

$$[x] := 3$$

$$\{x \mapsto 3 * y \mapsto 3 * z \mapsto 4\}$$

$$[y] := 4$$

$$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 4\}$$

$$[z] := 5$$

$$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 5\}$$

Talk outline

1. The problem
2. Towards a solution
3. A big proof
4. Fun with quantifiers
5. What about concurrency?

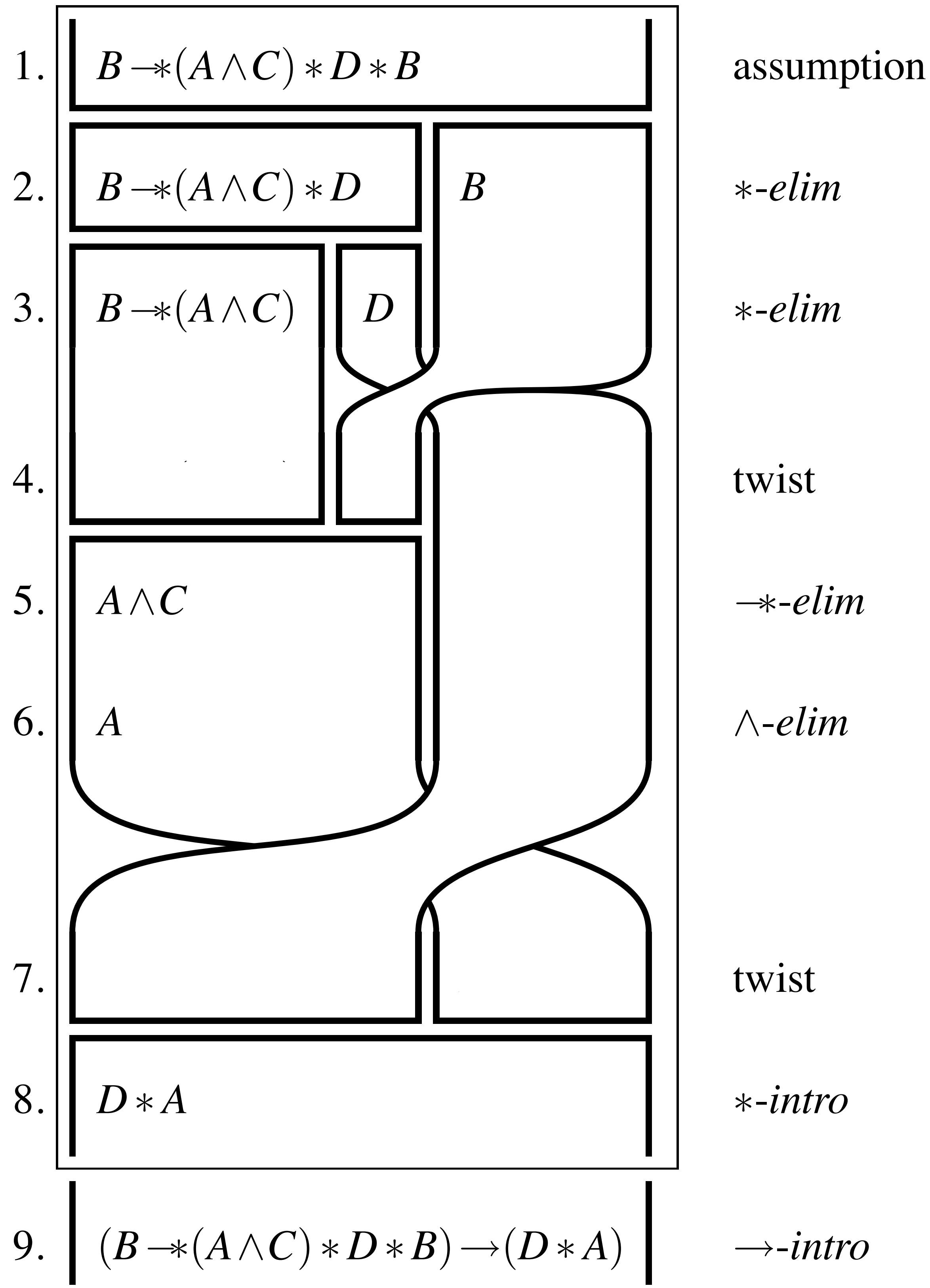
Box proofs

1	$P \to (Q \to R)$	
2	$P \to Q$	
3	P	
4	Q	$\to\mathcal{E}(2,3)$
5	$Q \to R$	$\to\mathcal{E}(1,3)$
6	R	$\to\mathcal{E}(5,4)$
7	$P \to R$	$\to\mathcal{I}$
8	$(P \to Q) \to (P \to R)$	$\to\mathcal{I}$

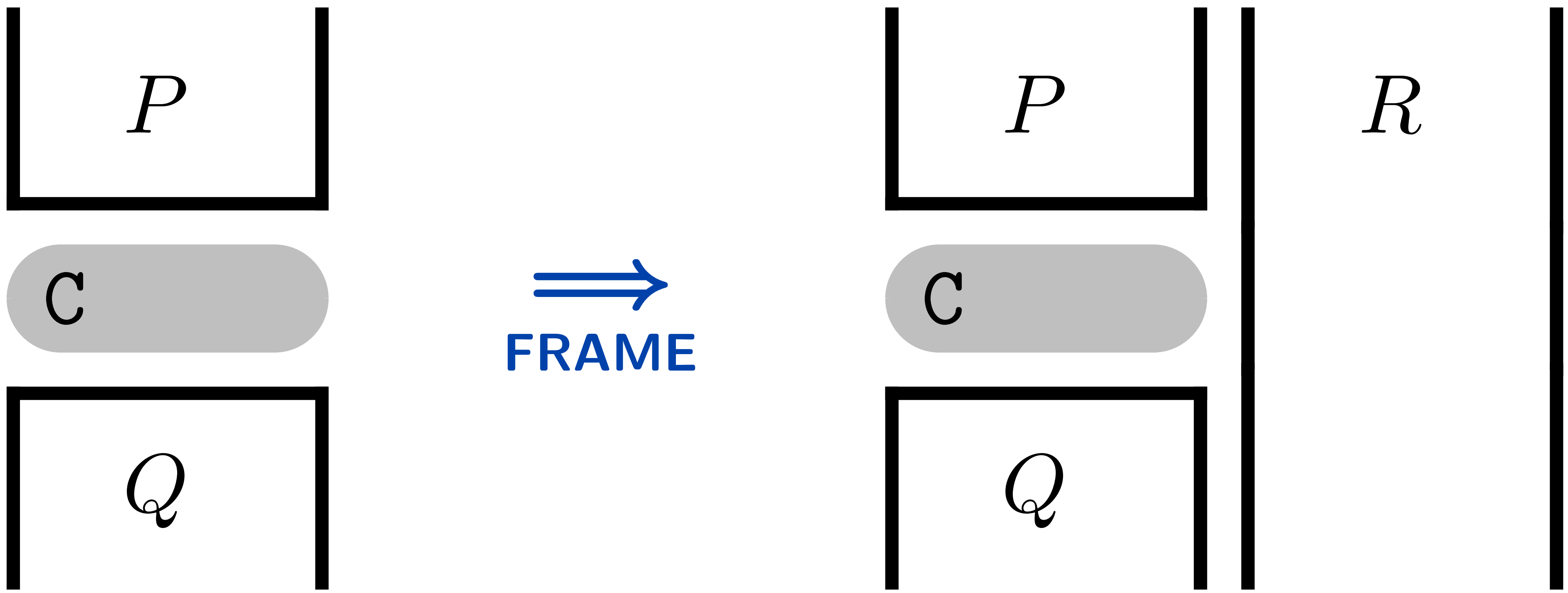
Ribbon proofs

1.	A	assumption
2.	$A \mathrel{-\!\!*} B$	assumption
3.	B	$\mathrel{-\!\!*}$-*elim*
4.	$(A \mathrel{-\!\!*} B) \mathrel{-\!\!*} B$	$\mathrel{-\!\!*}$-*intro*
5.	$A \to ((A \mathrel{-\!\!*} B) \mathrel{-\!\!*} B)$	$\to$-*intro*

Ribbon proofs



Ribbon proofs for commands



* providing R doesn't mention any variables that C might modify.

A simple program

$\{x \mapsto 2 * y \mapsto 3 * z \mapsto 4\}$

[x]:=3

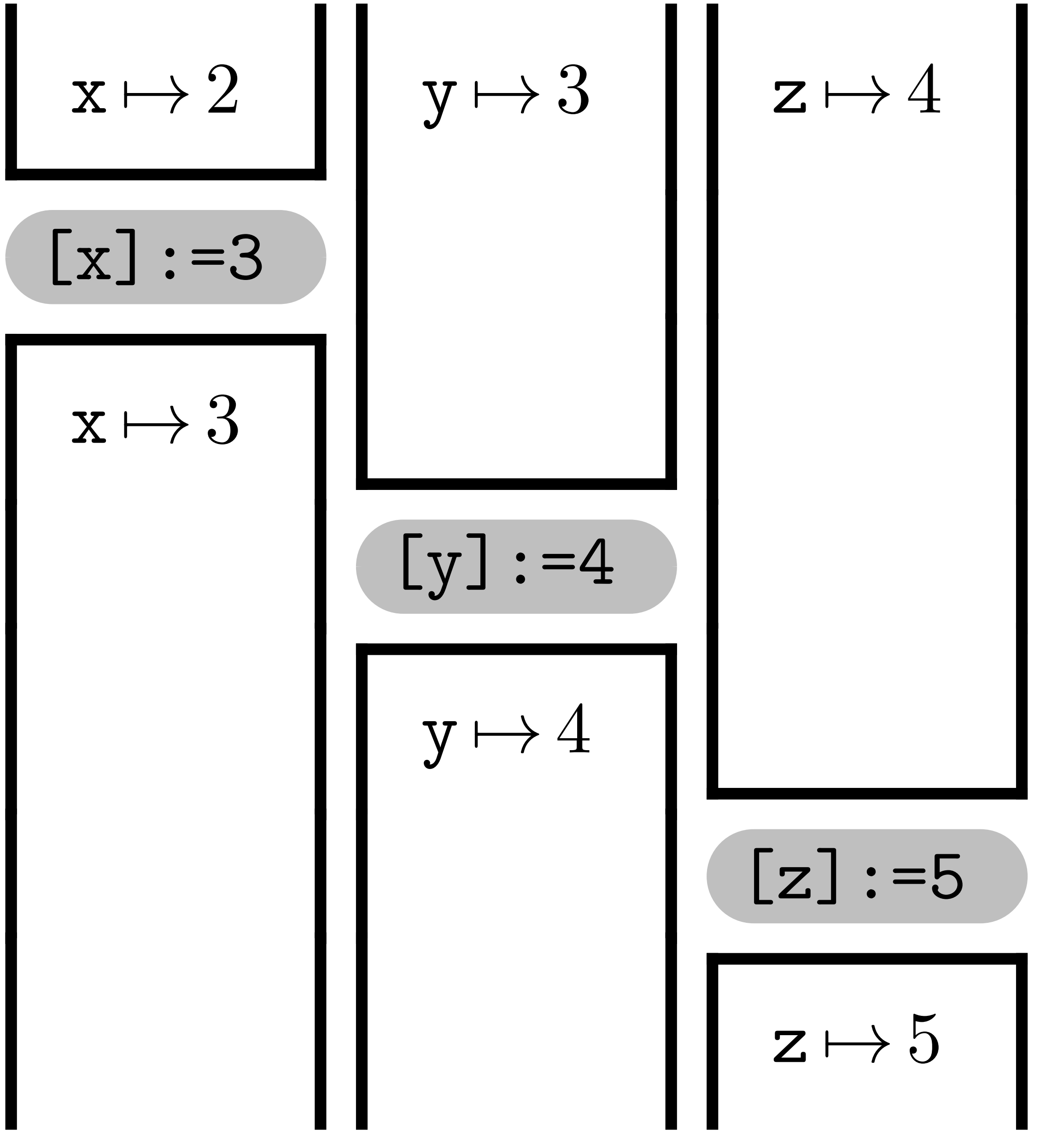
$\{x \mapsto 3 * y \mapsto 3 * z \mapsto 4\}$

[y]:=4

$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 4\}$

[z]:=5

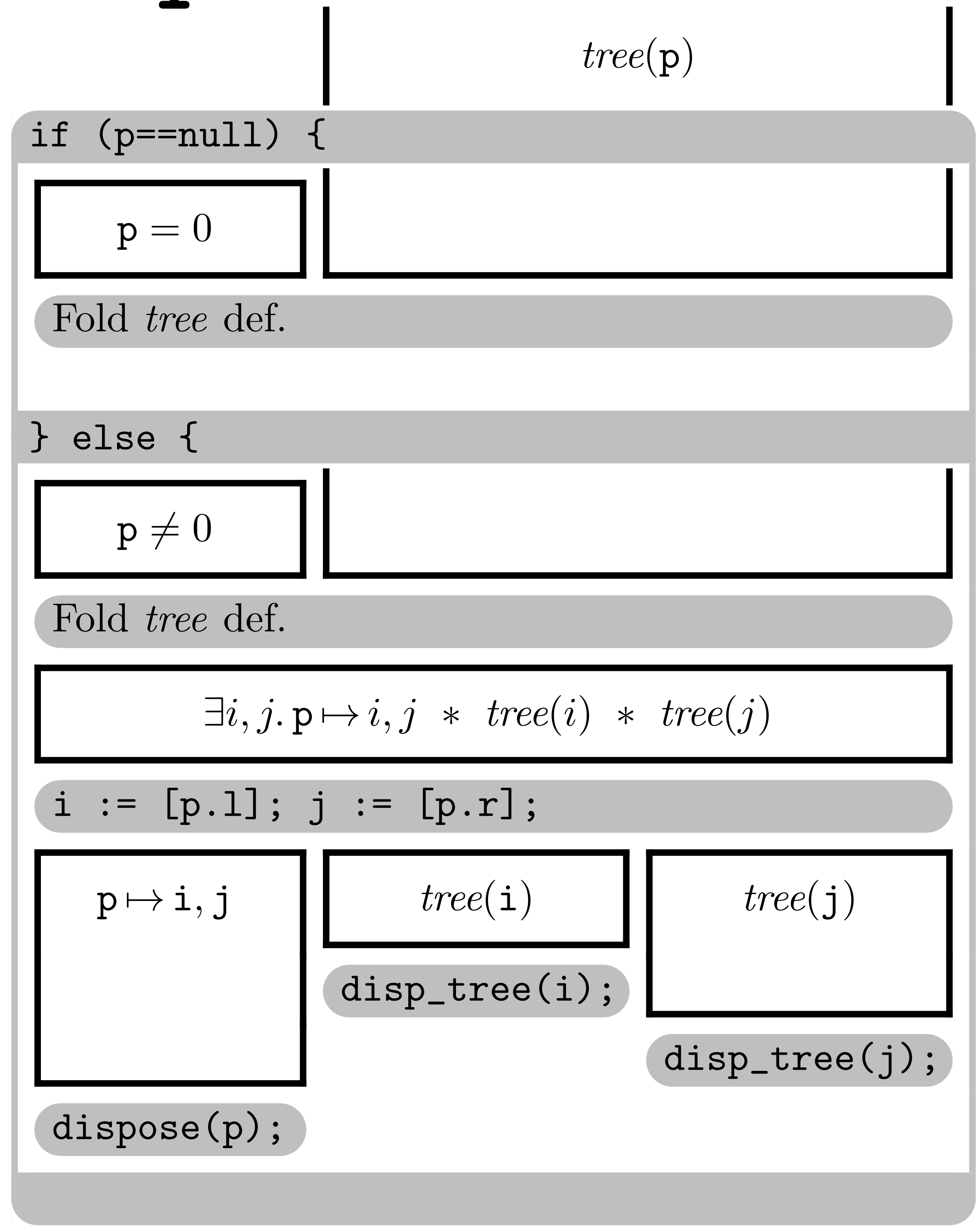
$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 5\}$

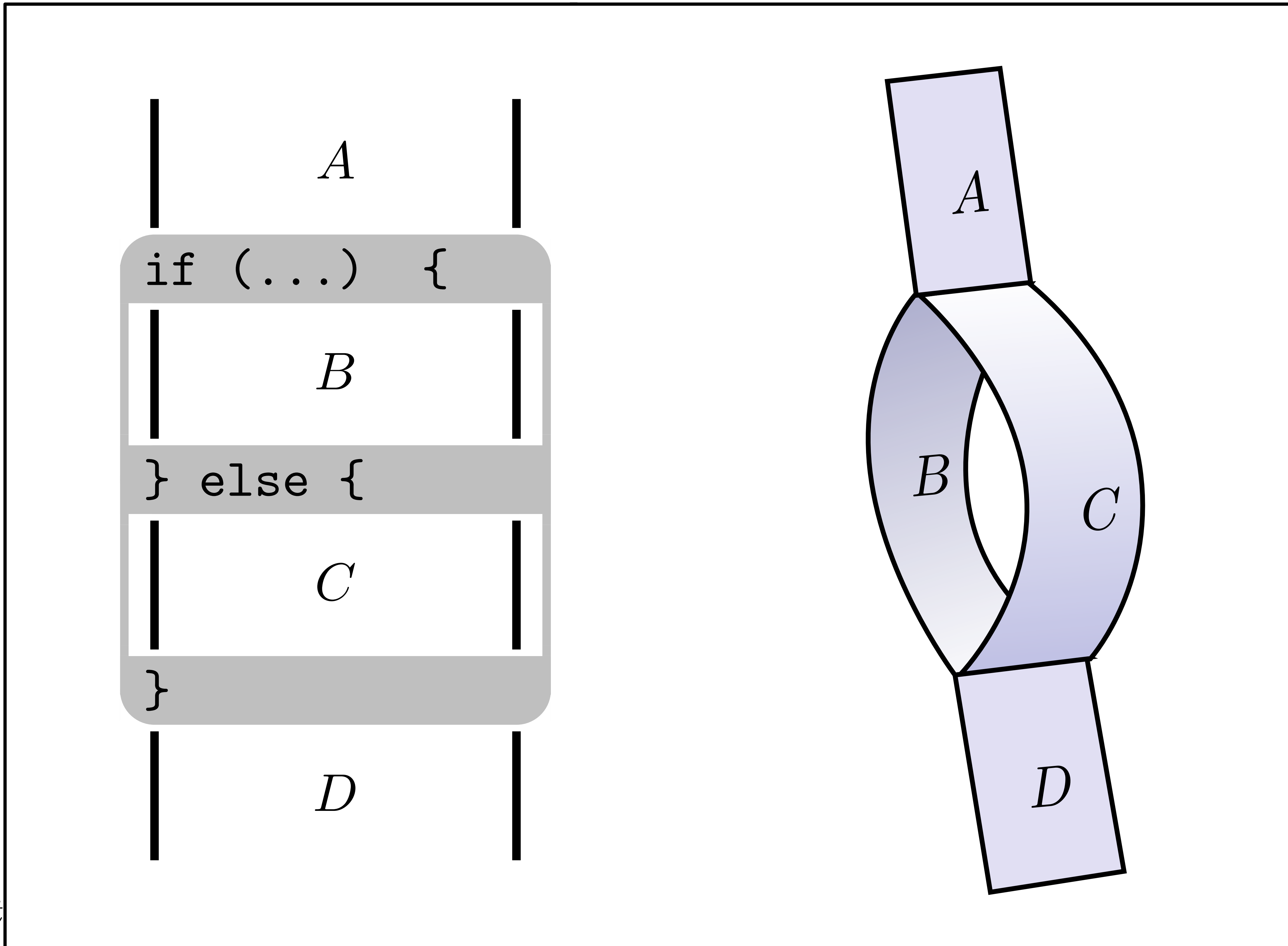


Example: `disp-tree`

```
disp_tree(p) {
  local i,j;
  if (p==null) {
    /* skip */
  } else {
    i := [p.l];
    j := [p.r];
    disp_tree(i);
    disp_tree(j);
    dispose(p);
  }
}
```

$$tree(x) \iff x = 0 \wedge emp \vee \exists i, j.\, x \mapsto i, j * tree(i) * tree(j)$$



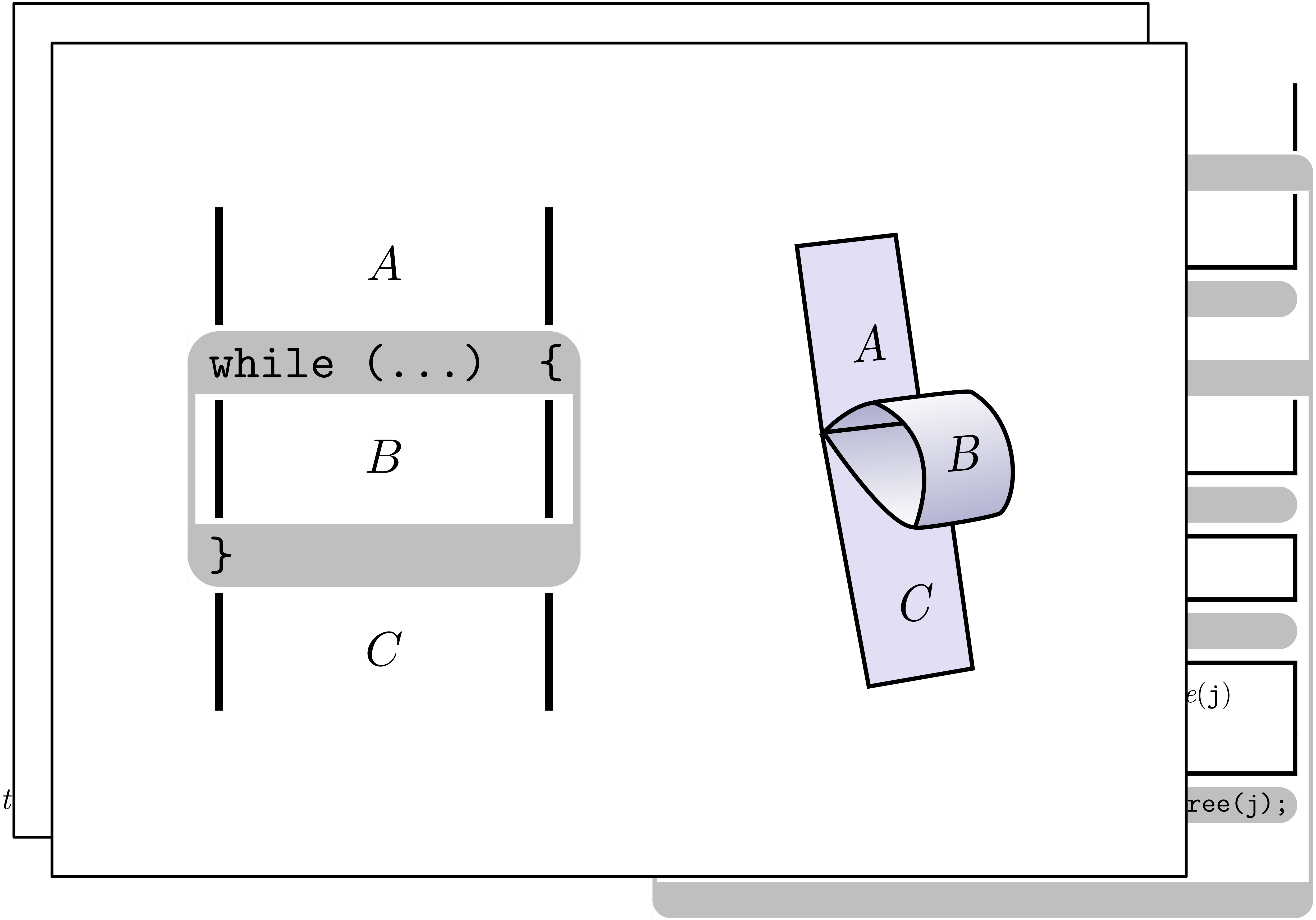


j)

tree(j)

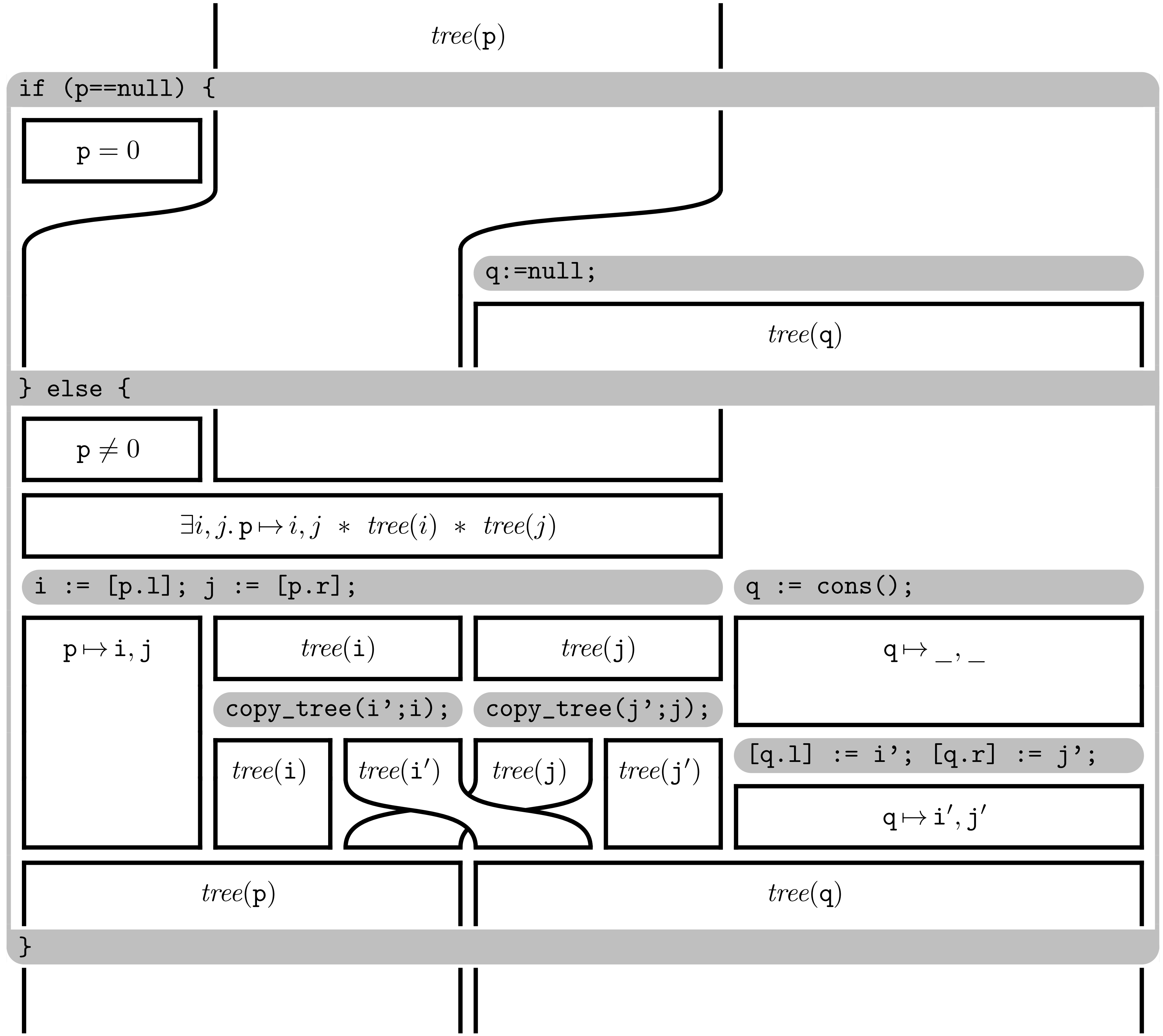
_tree(j);

dispose(p);


A
while (...) {
B
}
C
A
B
C

Example: copy-tree

```
copy_tree(q;p) {
  local i,j,i',j';
  if (p==null) {
    q:=null
  } else {
    i := [p.l];
    j := [p.r];
    copy_tree(i';i);
    copy_tree(j';j);
    q:=cons();
    [q.l] := i';
    [q.r] := j';
  }
}
```



A simple program

$$\{x \mapsto 2 * y \mapsto 3 * z \mapsto 4\}$$

$$[x]{:=}3$$

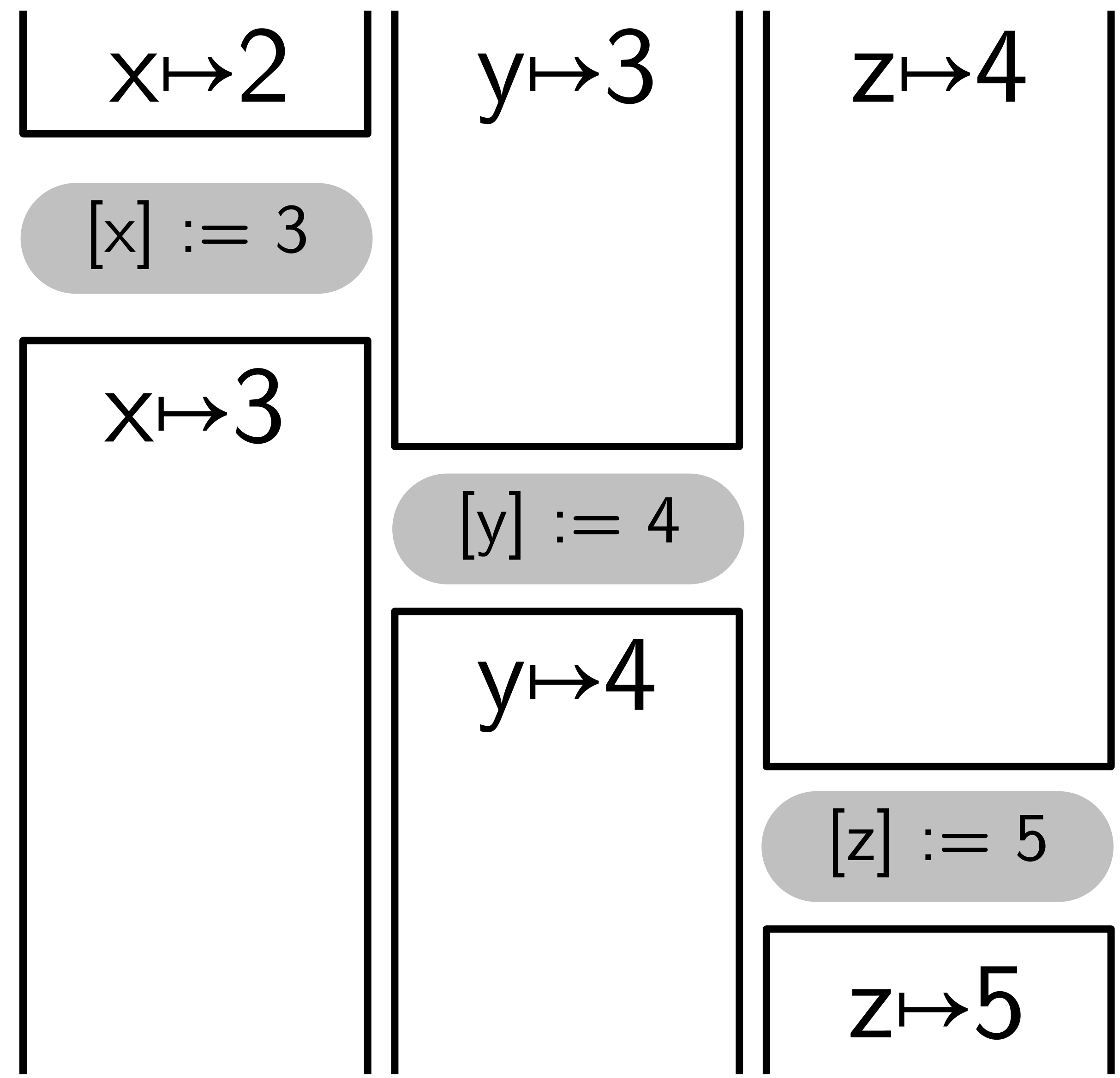
$$\{x \mapsto 3 * y \mapsto 3 * z \mapsto 4\}$$

$$[y]{:=}4$$

$$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 4\}$$

$$[z]{:=}5$$

$$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 5\}$$



A simple program

$\{x \mapsto 2 * y \mapsto 3 * z \mapsto 4\}$

$[x] := 3$

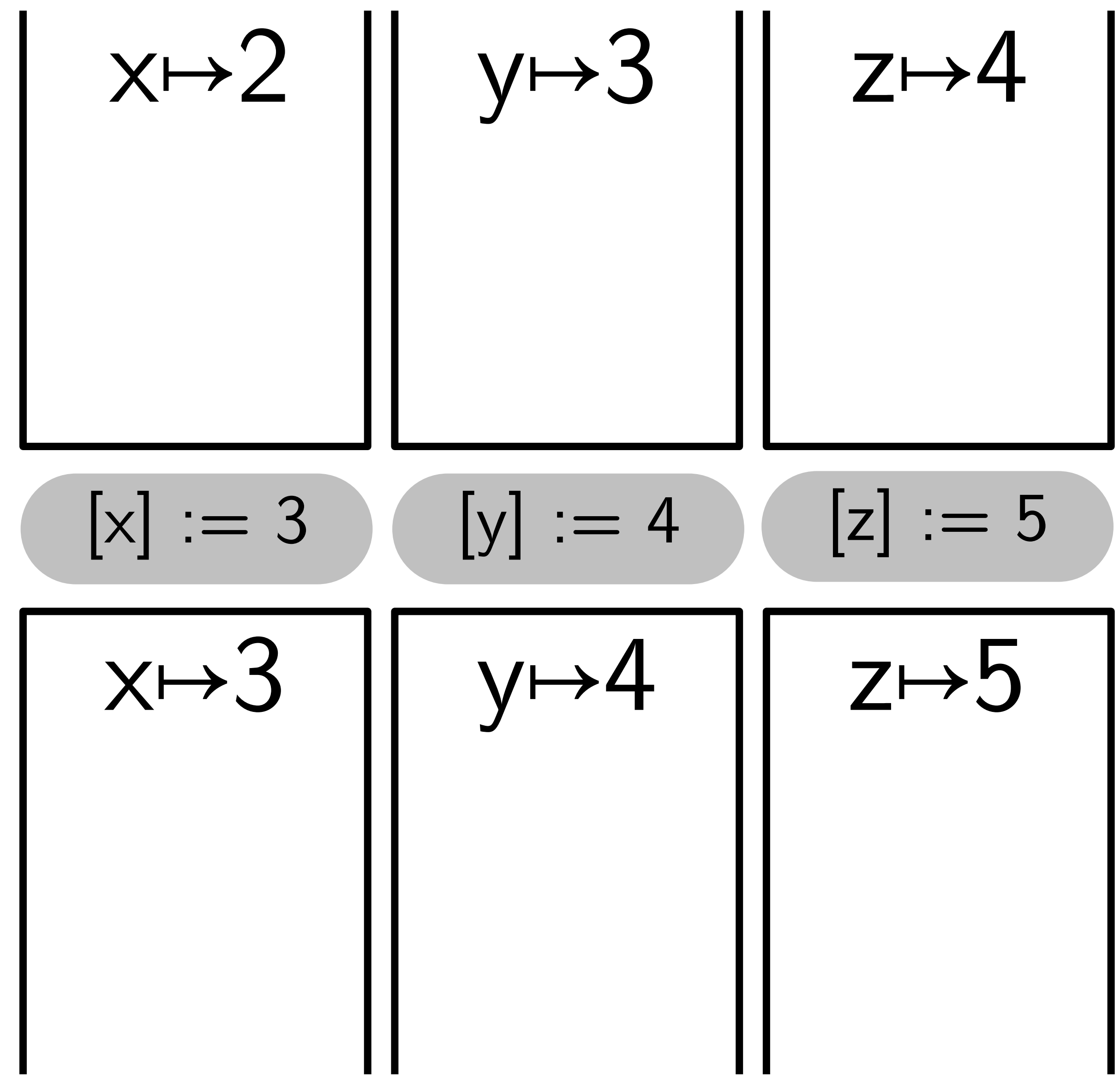
$\{x \mapsto 3 * y \mapsto 3 * z \mapsto 4\}$

$[y] := 4$

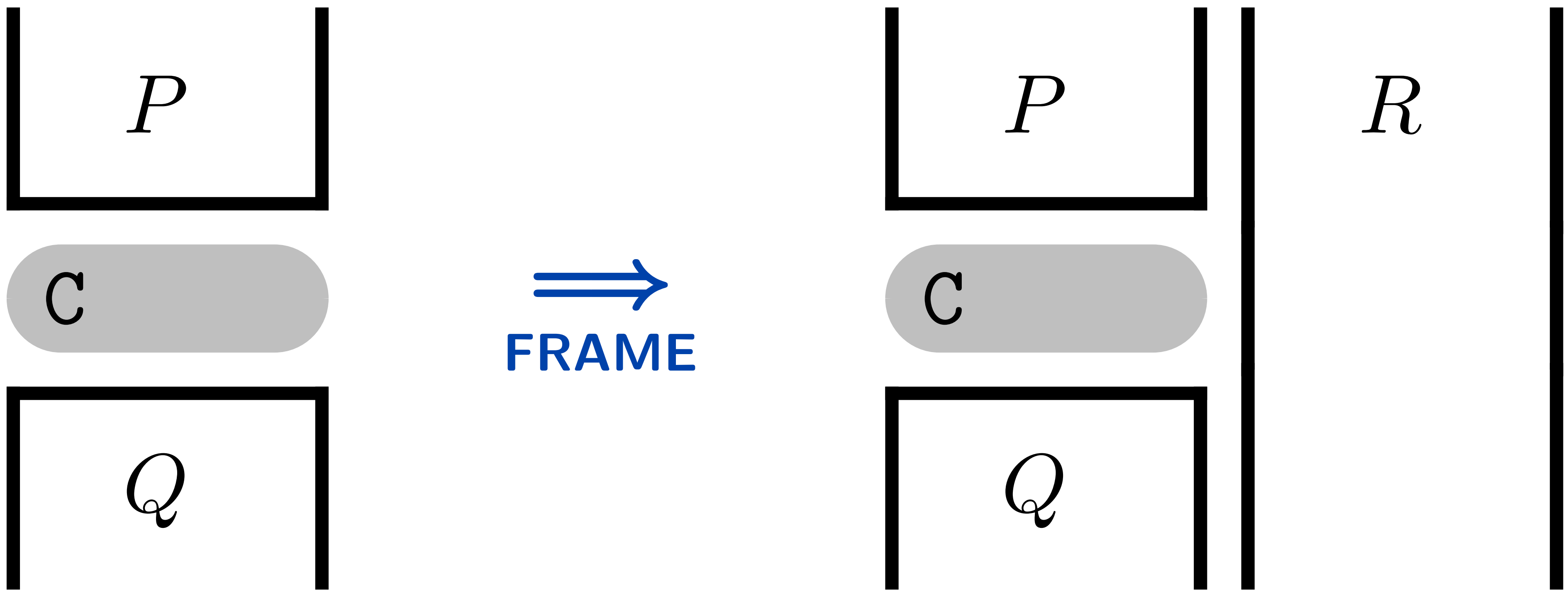
$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 4\}$

$[z] := 5$

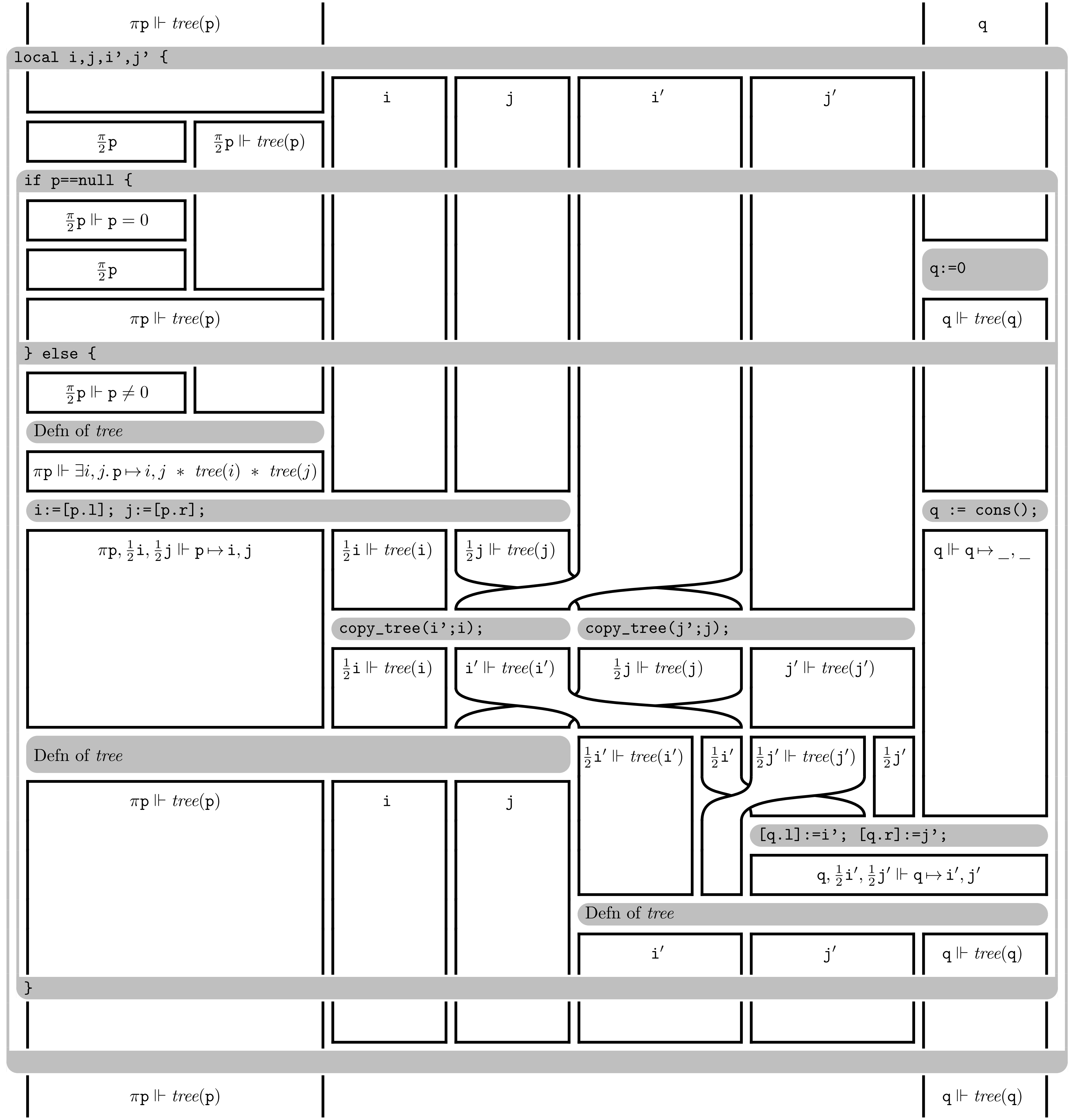
$\{x \mapsto 3 * y \mapsto 4 * z \mapsto 5\}$

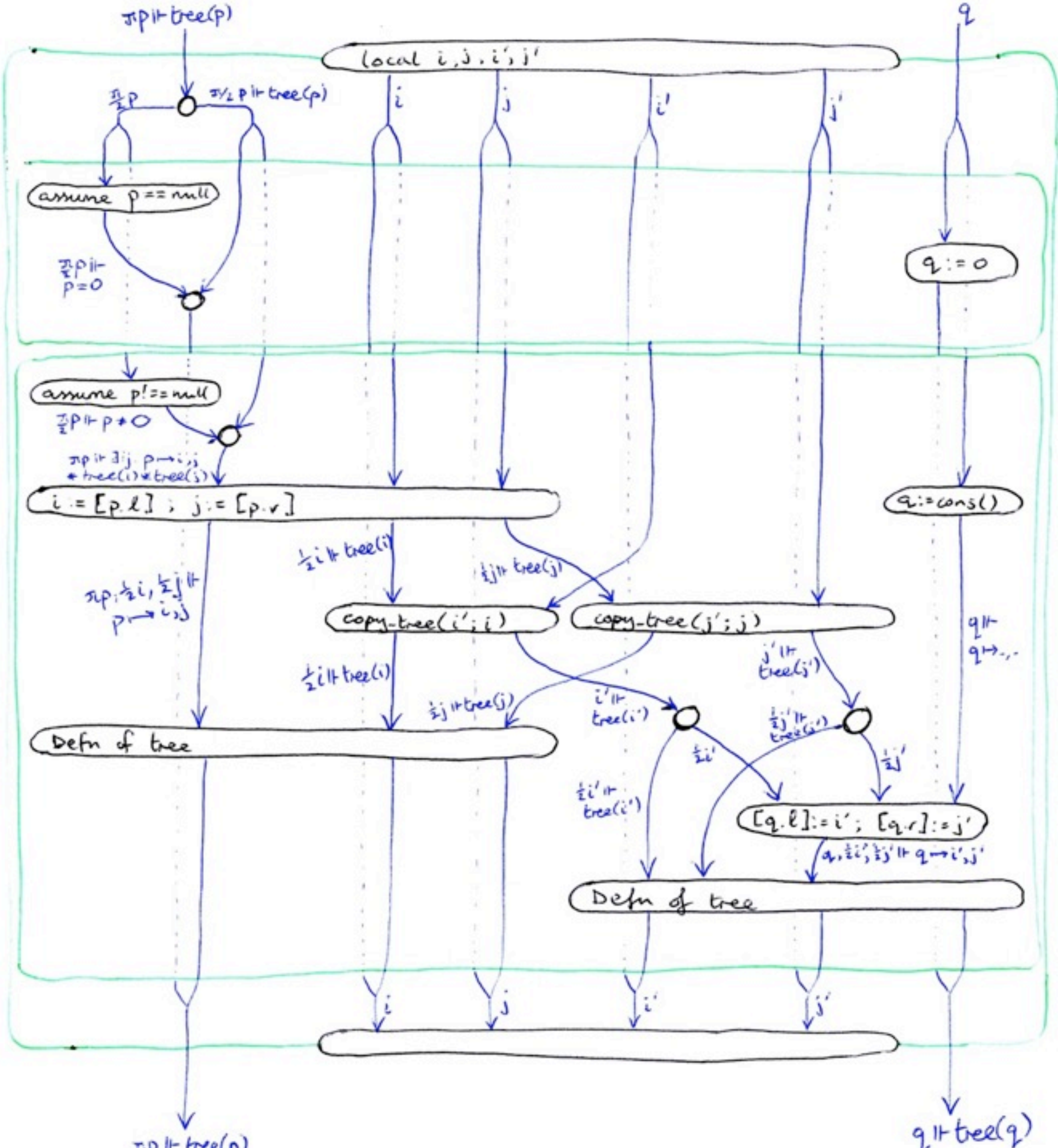


Ribbon proofs for commands



* providing R doesn't mention any variables that C might modify.

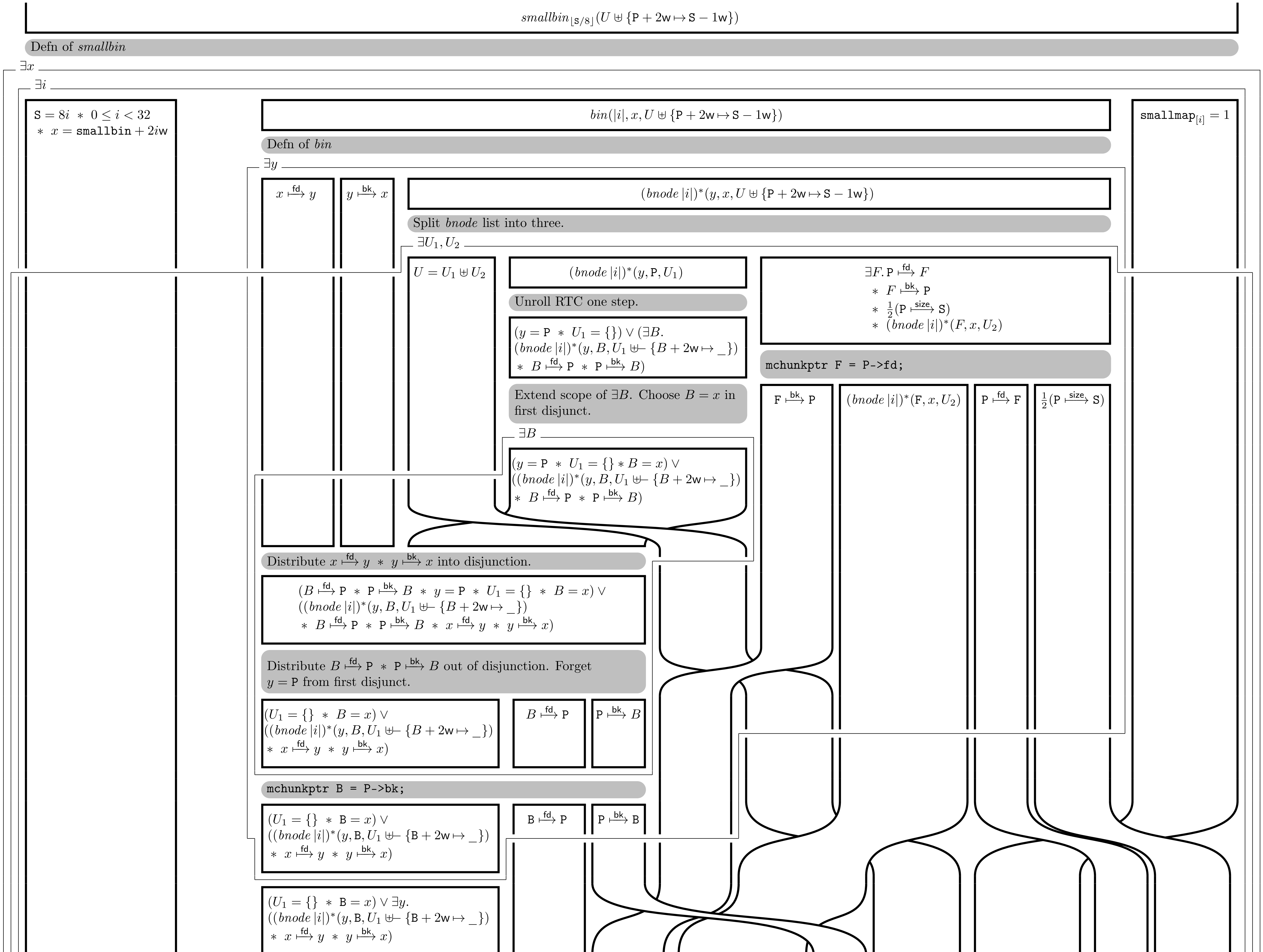

πp ⊩ tree(p)
q
local i,j,i',j' {
i
j
i′
j′
π/2 p
π/2 p ⊩ tree(p)
if p==null {
π/2 p ⊩ p = 0
π/2 p
πp ⊩ tree(p)
q:=0
q ⊩ tree(q)
} else {
π/2 p ⊩ p ≠ 0
Defn of tree
πp ⊩ ∃i, j. p ↦ i, j ∗ tree(i) ∗ tree(j)
i:=[p.l]; j:=[p.r];
q := cons();
πp, 1/2 i, 1/2 j ⊩ p ↦ i, j
1/2 i ⊩ tree(i)
1/2 j ⊩ tree(j)
q ⊩ q ↦ _, _
copy_tree(i';i);
copy_tree(j';j);
1/2 i ⊩ tree(i)
i′ ⊩ tree(i′)
1/2 j ⊩ tree(j)
j′ ⊩ tree(j′)
Defn of tree
1/2 i′ ⊩ tree(i′)
1/2 i′
1/2 j′ ⊩ tree(j′)
1/2 j′
πp ⊩ tree(p)
i
j
[q.l]:=i'; [q.r]:=j';
q, 1/2 i′, 1/2 j′ ⊩ q ↦ i′, j′
Defn of tree
i′
j′
q ⊩ tree(q)
}
πp ⊩ tree(p)
q ⊩ tree(q)


πp ⊩ tree(p)
local i, j, i', j'
q
assume p == null
q := 0
assume p! == null
i := [p.l] ; j := [p.r]
q := cons()
copy-tree(i'; i)
copy-tree(j'; j)
Defn of tree
[q.l] := i'; [q.r] := j'
Defn of tree
πp ⊩ tree(p)
q ⊩ tree(q)

Talk outline

1. The problem
2. Towards a solution
3. A big proof
4. Fun with quantifiers
5. What about concurrency?

Proof of `unlink_first_small_chunk`



$y = \mathtt{P}$ from first disjunct.

$(U_1 = \{\} \;*\; B = x) \vee ((\mathit{bnode}\,|i|)^*(y, B, U_1 \uplus\!\!- \{B + 2\mathsf{w} \mapsto \_\}) \;*\; x \overset{\mathsf{fd}}{\longmapsto} y \;*\; y \overset{\mathsf{bk}}{\longmapsto} x)$

$B \overset{\mathsf{fd}}{\longmapsto} \mathtt{P}$

$\mathtt{P} \overset{\mathsf{bk}}{\longmapsto} B$

```
mchunkptr B = P->bk;
```

$(U_1 = \{\} \;*\; \mathtt{B} = x) \vee ((\mathit{bnode}\,|i|)^*(y, \mathtt{B}, U_1 \uplus\!\!- \{\mathtt{B} + 2\mathsf{w} \mapsto \_\}) \;*\; x \overset{\mathsf{fd}}{\longmapsto} y \;*\; y \overset{\mathsf{bk}}{\longmapsto} x)$

$\mathtt{B} \overset{\mathsf{fd}}{\longmapsto} \mathtt{P}$

$\mathtt{P} \overset{\mathsf{bk}}{\longmapsto} \mathtt{B}$

$(U_1 = \{\} \;*\; \mathtt{B} = x) \vee \exists y.\, ((\mathit{bnode}\,|i|)^*(y, \mathtt{B}, U_1 \uplus\!\!- \{\mathtt{B} + 2\mathsf{w} \mapsto \_\}) \;*\; x \overset{\mathsf{fd}}{\longmapsto} y \;*\; y \overset{\mathsf{bk}}{\longmapsto} x)$

```
bindex_t I = small_index(S);
```

$\mathtt{S} = 8\mathtt{I} \;*\; 0 \leq \mathtt{I} < 32 \;*\; x = \mathtt{smallbin} + 2\mathtt{I}\mathsf{w}$

$(U_1 = \{\} \;*\; \mathtt{B} = x) \vee \exists y.\, ((\mathit{bnode}\,|\mathtt{I}|)^*(y, \mathtt{B}, U_1 \uplus\!\!- \{\mathtt{B} + 2\mathsf{w} \mapsto \_\}) \;*\; x \overset{\mathsf{fd}}{\longmapsto} y \;*\; y \overset{\mathsf{bk}}{\longmapsto} x)$

$(\mathit{bnode}\,|\mathtt{I}|)^*(\mathtt{F}, x, U_2)$

$\mathtt{smallmap}_{[\mathtt{I}]} = 1$

$\mathtt{P} \overset{\mathsf{bk}}{\longmapsto} \_$

$\mathtt{P} \overset{\mathsf{fd}}{\longmapsto} \_$

```
if (F==B) {
```

$\mathtt{B} = \mathtt{F}$

Ribbons 1 and 4 contradict second disjunct of ribbon 2; hence $\mathtt{B} = \mathtt{F} = x$ and $U_1 = \{\}$. Since $\mathtt{F} = x$, ribbon 5 implies $U_2 = \{\}$. (See Lemma 1, above.)

$x \overset{\mathsf{fd}}{\longmapsto} \_$

$x \overset{\mathsf{bk}}{\longmapsto} \_$

$U_1 = \{\} \;*\; U_2 = \{\}$

$U = \{\}$

```
clear_smallmap(M,I);
```

$U = \{\}$

$\mathtt{smallmap}_{[\mathtt{I}]} = (U \neq \{\})$

Defn of *bin*.

$\mathit{bin}(|\mathtt{I}|, x, U)$

```
} else {
```

$\mathtt{B} \neq \mathtt{F}$

$(U_1 = \{\} \;*\; \mathtt{B} = x) \vee \exists y.\, ((\mathit{bnode}\,|\mathtt{I}|)^*(y, \mathtt{B}, U_1 \uplus\!\!- \{\mathtt{B} + 2\mathsf{w} \mapsto \_\}) \;*\; x \overset{\mathsf{fd}}{\longmapsto} y \;*\; y \overset{\mathsf{bk}}{\longmapsto} x)$

$\mathtt{B} \overset{\mathsf{fd}}{\longmapsto} \mathtt{P}$

$\mathtt{F} \overset{\mathsf{bk}}{\longmapsto} \mathtt{P}$

$(\mathit{bnode}\,|\mathtt{I}|)^*(\mathtt{F}, x, U_2)$

$U = U_1 \uplus U_2$

$\mathtt{smallmap}_{[\mathtt{I}]} = 1$

From ribbons 1, 2 and 3, deduce U_1 and U_2 can't both be empty. Extend scope of $\exists y$ in ribbon 2, choosing $y = \mathtt{F}$ in first disjunct.

$\exists y.\, (y = \mathtt{F} \;*\; U_1 = \{\} \;*\; \mathtt{B} = x) \vee ((\mathit{bnode}\,|\mathtt{I}|)^*(y, \mathtt{B}, U_1 \uplus\!\!- \{\mathtt{B} + 2\mathsf{w} \mapsto \_\}) \;*\; x \overset{\mathsf{fd}}{\longmapsto} y \;*\; y \overset{\mathsf{bk}}{\longmapsto} x)$

```
B->fd = F
```

```
F->bk = B
```

$\mathtt{B} \overset{\mathsf{fd}}{\longmapsto} \mathtt{F}$

$\mathtt{F} \overset{\mathsf{bk}}{\longmapsto} \mathtt{B}$

$(\mathit{bnode}\,|\mathtt{I}|)^*(\mathtt{F}, x, U_2)$

$U = U_1 \uplus U_2$

$\mathtt{smallmap}_{[\mathtt{I}]} = (U \neq \{\})$

Distribute $\mathtt{B} \overset{\mathsf{fd}}{\longmapsto} \mathtt{F} \;*\; \mathtt{F} \overset{\mathsf{bk}}{\longmapsto} \mathtt{B}$ into disjunction.

$\exists y$

$x \overset{\mathsf{fd}}{\longmapsto} \_$ | $x \overset{\mathsf{bk}}{\longmapsto} \_$ | $U_1 = \{\} \ * \ U_2 = \{\}$

$U = \{\}$

`clear_smallmap(M,I);`

$U = \{\}$

$\mathtt{smallmap}_{[\mathtt{I}]} = (U \neq \{\})$

Defn of *bin*.

$bin(|\mathtt{I}|, x, U)$

`} else {`

$\mathtt{B} \neq \mathtt{F}$

$(U_1 = \{\} \ * \ \mathtt{B} = x) \vee \exists y.$
$((bnode\,|\mathtt{I}|)^*(y, \mathtt{B}, U_1 \uplus\!\!- \{\mathtt{B} + 2\mathsf{w} \mapsto \_\})$
$* \ x \overset{\mathsf{fd}}{\longmapsto} y \ * \ y \overset{\mathsf{bk}}{\longmapsto} x)$

$\mathtt{B} \overset{\mathsf{fd}}{\longmapsto} \mathtt{P}$ | $\mathtt{F} \overset{\mathsf{bk}}{\longmapsto} \mathtt{P}$ | $(bnode\,|\mathtt{I}|)^*(\mathtt{F}, x, U_2)$ | $U = U_1 \uplus U_2$ | $\mathtt{smallmap}_{[\mathtt{I}]} = 1$

From ribbons 1, 2 and 3, deduce U_1 and U_2 can't both be empty. Extend scope of $\exists y$ in ribbon 2, choosing $y = \mathtt{F}$ in first disjunct.

$\exists y. (y = \mathtt{F} \ * \ U_1 = \{\} \ * \ \mathtt{B} = x) \vee$
$((bnode\,|\mathtt{I}|)^*(y, \mathtt{B}, U_1 \uplus\!\!- \{\mathtt{B} + 2\mathsf{w} \mapsto \_\})$
$* \ x \overset{\mathsf{fd}}{\longmapsto} y \ * \ y \overset{\mathsf{bk}}{\longmapsto} x)$

`B->fd = F` | `F->bk = B`

$\mathtt{B} \overset{\mathsf{fd}}{\longmapsto} \mathtt{F}$ | $\mathtt{F} \overset{\mathsf{bk}}{\longmapsto} \mathtt{B}$

$(bnode\,|\mathtt{I}|)^*(\mathtt{F}, x, U_2)$ | $U = U_1 \uplus U_2$ | $\mathtt{smallmap}_{[\mathtt{I}]} = (U \neq \{\})$

Distribute $\mathtt{B} \overset{\mathsf{fd}}{\longmapsto} \mathtt{F} \ * \ \mathtt{F} \overset{\mathsf{bk}}{\longmapsto} \mathtt{B}$ into disjunction.

$\exists y$

$(y = \mathtt{F} \ * \ U_1 = \{\} \ * \ \mathtt{B} = x \ * \ \mathtt{B} \overset{\mathsf{fd}}{\longmapsto} \mathtt{F} \ * \ \mathtt{F} \overset{\mathsf{bk}}{\longmapsto} \mathtt{B}) \vee$
$((bnode\,|\mathtt{I}|)^*(y, \mathtt{B}, U_1 \uplus\!\!- \{\mathtt{B} + 2\mathsf{w} \mapsto \_\})$
$* \ x \overset{\mathsf{fd}}{\longmapsto} y \ * \ y \overset{\mathsf{bk}}{\longmapsto} x \ * \ \mathtt{B} \overset{\mathsf{fd}}{\longmapsto} \mathtt{F} \ * \ \mathtt{F} \overset{\mathsf{bk}}{\longmapsto} \mathtt{B})$

Distribute $x \overset{\mathsf{fd}}{\longmapsto} y \ * \ y \overset{\mathsf{bk}}{\longmapsto} x$ out of disjunction.

$x \overset{\mathsf{fd}}{\longmapsto} y$ | $y \overset{\mathsf{bk}}{\longmapsto} x$

$(y = \mathtt{F} \ * \ U_1 = \{\}) \vee$
$((bnode\,|\mathtt{I}|)^*(y, \mathtt{B}, U_1 \uplus\!\!- \{\mathtt{B} + 2\mathsf{w} \mapsto \_\})$
$* \ \mathtt{B} \overset{\mathsf{fd}}{\longmapsto} \mathtt{F} \ * \ \mathtt{F} \overset{\mathsf{bk}}{\longmapsto} \mathtt{B})$

Roll RTC.

$(bnode\,|\mathtt{I}|)^*(y, \mathtt{F}, U_1)$

Merge two *bnode* lists.

$(bnode\,|\mathtt{I}|)^*(y, x, U)$

Defn of *bin*.

$bin(|\mathtt{I}|, x, U)$

`}`

Defn of *smallbin*.

$smallbin_{\lfloor \mathtt{S}/8 \rfloor}(U)$

$smallbin_{\lfloor \mathtt{S}/8 \rfloor}(U)$ | $\mathtt{P} \overset{\mathsf{bk}}{\longmapsto} \_$ | $\mathtt{P} \overset{\mathsf{fd}}{\longmapsto} \_$ | $\frac{1}{2}(\mathtt{P} \overset{\mathsf{size}}{\longmapsto} \mathtt{S})$

Talk outline

1. The problem
2. Towards a solution
3. A big proof
4. Fun with quantifiers
5. What about concurrency?

Example: `list-rev`

$\{\text{list } \delta\ x\}$

y := 0;

while $\{\exists \alpha, \beta.\ \text{list } \alpha\ x * \text{list } \beta\ y * \delta \doteq {-\beta} \cdot \alpha\}$ (x≠0) do {

$\{\exists i, \alpha, \beta, Z.\ x \mapsto i, Z * \text{list } \alpha\ Z * \text{list } \beta\ y * \delta \doteq {-\beta} \cdot i \cdot \alpha\}$

z := [x+1];

$\{\exists i, \alpha, \beta.\ x \mapsto i, z * \text{list } \alpha\ z * \text{list } \beta\ y * \delta \doteq {-\beta} \cdot i \cdot \alpha\}$

[x+1] := y;

$\{\exists i, \alpha, \beta.\ x \mapsto i, y * \text{list } \alpha\ z * \text{list } \beta\ y * \delta \doteq {-\beta} \cdot i \cdot \alpha\}$

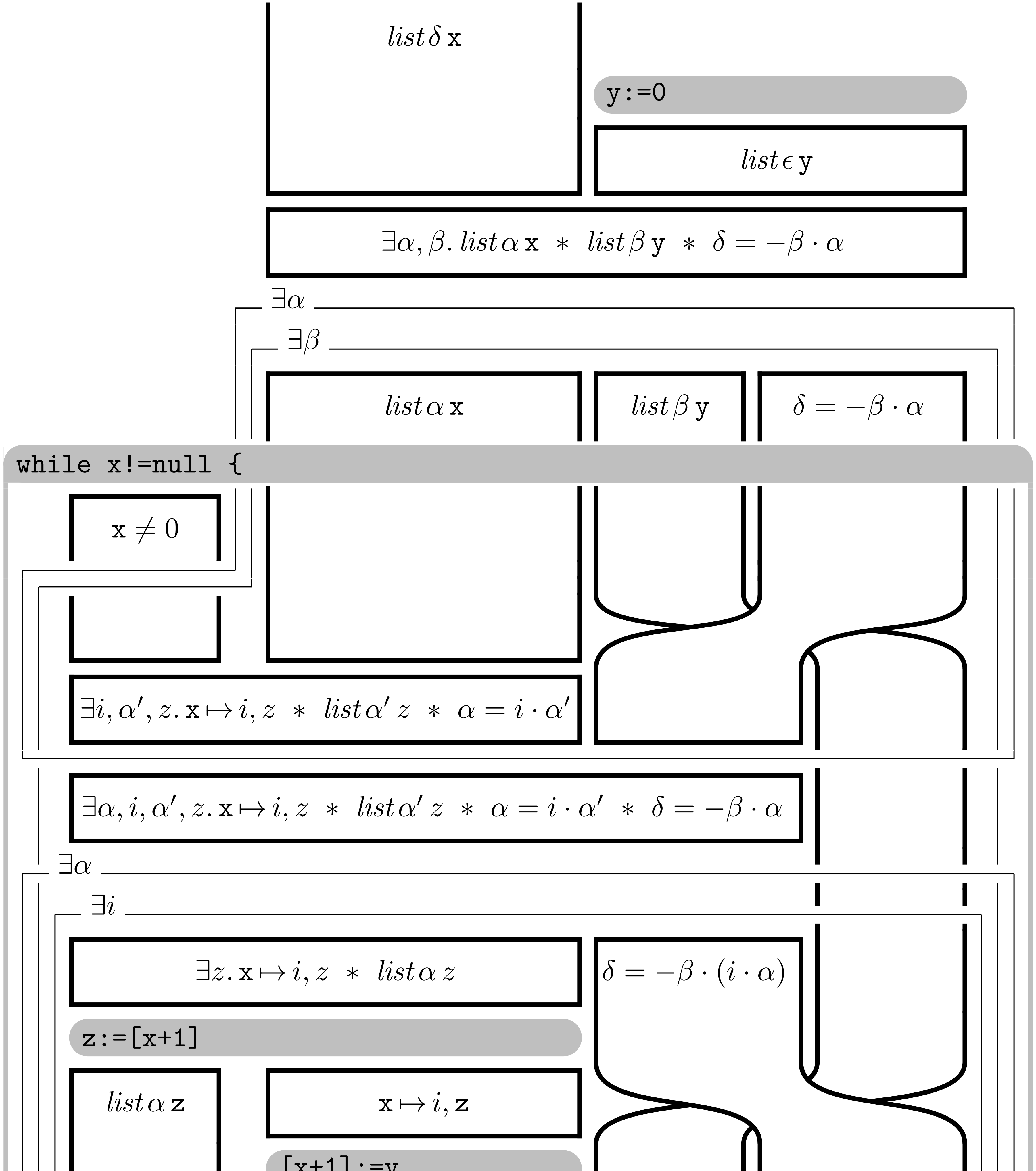
$\{\exists \alpha, \beta.\ \text{list } \alpha\ z * \text{list } \beta\ x * \delta \doteq {-\beta} \cdot \alpha\}$

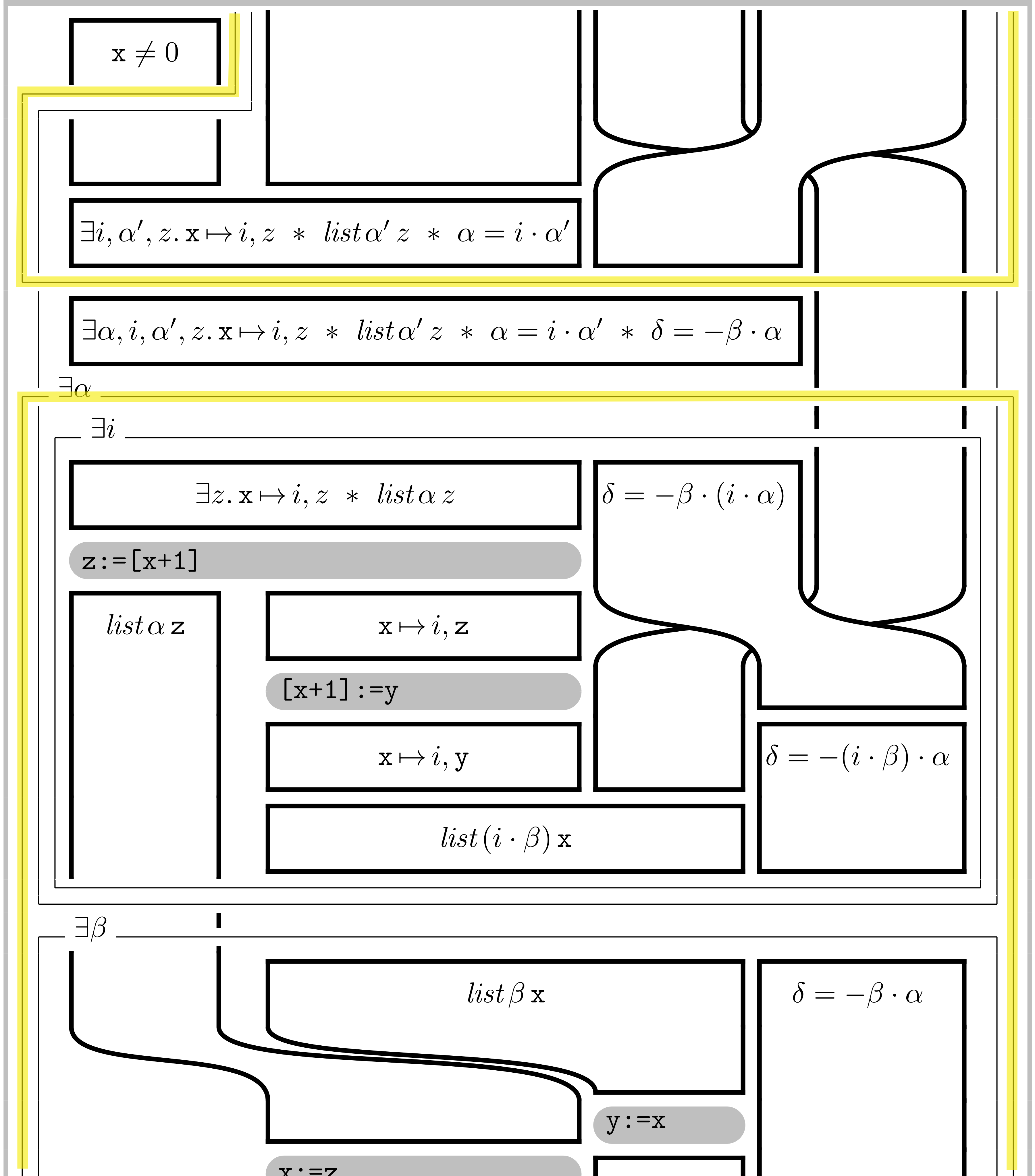
y := x; x := z;

$\{\exists \alpha, \beta.\ \text{list } \alpha\ x * \text{list } \beta\ y * \delta \doteq {-\beta} \cdot \alpha\}$

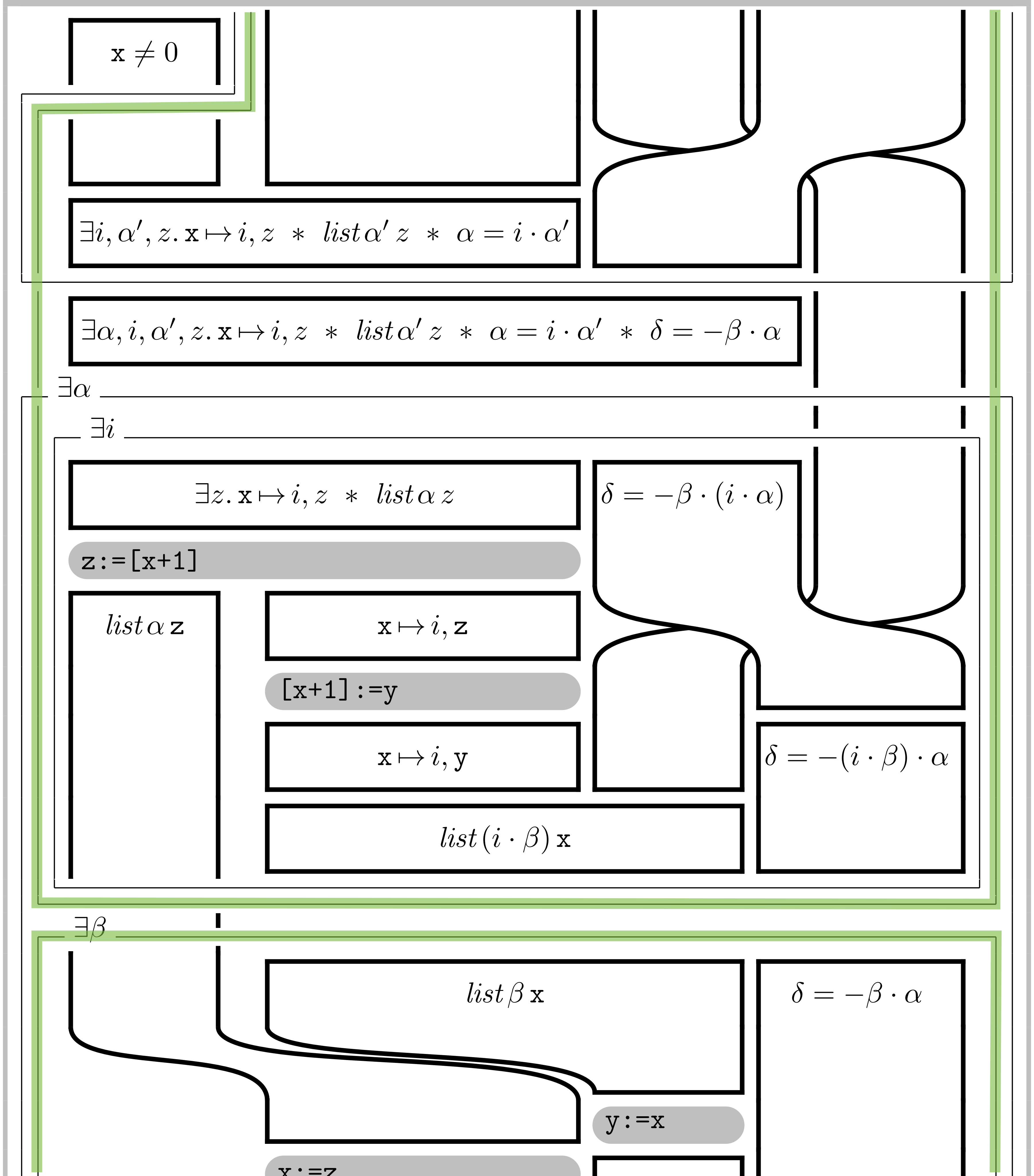
}

$\{\text{list } {-\delta}\ y\}$

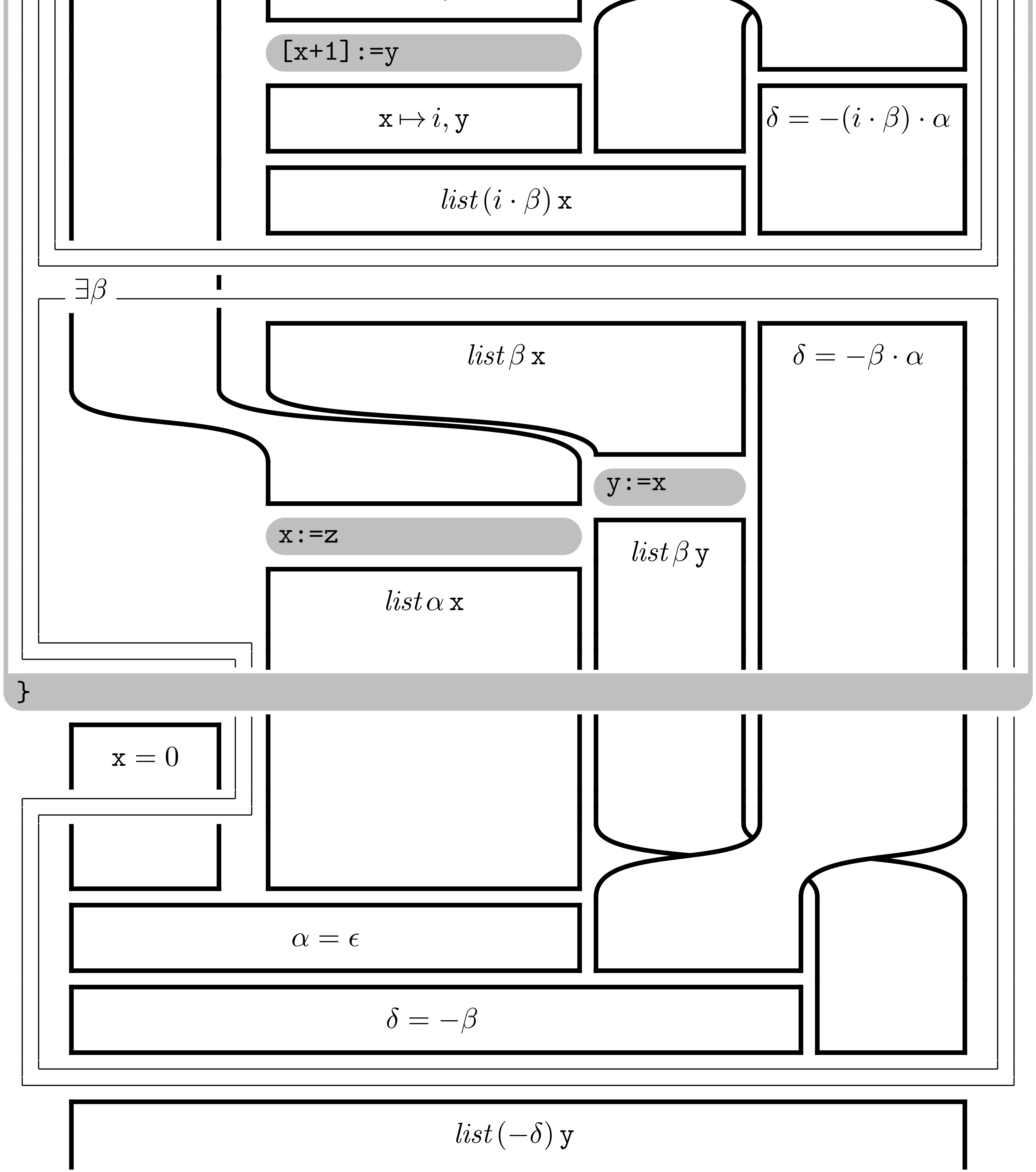

$\mathit{list}\,\delta\,\mathtt{x}$
y:=0
$\mathit{list}\,\epsilon\,\mathtt{y}$
$\exists\alpha,\beta.\,\mathit{list}\,\alpha\,\mathtt{x} \;*\; \mathit{list}\,\beta\,\mathtt{y} \;*\; \delta = -\beta\cdot\alpha$
$\exists\alpha$
$\exists\beta$
$\mathit{list}\,\alpha\,\mathtt{x}$
$\mathit{list}\,\beta\,\mathtt{y}$
$\delta = -\beta\cdot\alpha$
while x!=null {
$\mathtt{x} \neq 0$
$\exists i,\alpha',z.\,\mathtt{x}\mapsto i,z \;*\; \mathit{list}\,\alpha'\,z \;*\; \alpha = i\cdot\alpha'$
$\exists\alpha,i,\alpha',z.\,\mathtt{x}\mapsto i,z \;*\; \mathit{list}\,\alpha'\,z \;*\; \alpha = i\cdot\alpha' \;*\; \delta = -\beta\cdot\alpha$
$\exists\alpha$
$\exists i$
$\exists z.\,\mathtt{x}\mapsto i,z \;*\; \mathit{list}\,\alpha\,z$
$\delta = -\beta\cdot(i\cdot\alpha)$
z:=[x+1]
$\mathit{list}\,\alpha\,\mathtt{z}$
$\mathtt{x}\mapsto i,\mathtt{z}$



$\mathtt{x} \neq 0$
$\exists i, \alpha', z.\, \mathtt{x} \mapsto i, z \;*\; \mathit{list}\, \alpha'\, z \;*\; \alpha = i \cdot \alpha'$
$\exists \alpha, i, \alpha', z.\, \mathtt{x} \mapsto i, z \;*\; \mathit{list}\, \alpha'\, z \;*\; \alpha = i \cdot \alpha' \;*\; \delta = -\beta \cdot \alpha$
$\exists \alpha$
$\exists i$
$\exists z.\, \mathtt{x} \mapsto i, z \;*\; \mathit{list}\, \alpha\, z$
$\delta = -\beta \cdot (i \cdot \alpha)$
z:=[x+1]
$\mathit{list}\, \alpha\, \mathtt{z}$
$\mathtt{x} \mapsto i, \mathtt{z}$
[x+1]:=y
$\mathtt{x} \mapsto i, \mathtt{y}$
$\delta = -(i \cdot \beta) \cdot \alpha$
$\mathit{list}\, (i \cdot \beta)\, \mathtt{x}$
$\exists \beta$
$\mathit{list}\, \beta\, \mathtt{x}$
$\delta = -\beta \cdot \alpha$
y:=x



x ≠ 0
∃i, α′, z. x ↦ i, z ∗ list α′ z ∗ α = i · α′
∃α, i, α′, z. x ↦ i, z ∗ list α′ z ∗ α = i · α′ ∗ δ = −β · α
∃α
∃i
∃z. x ↦ i, z ∗ list α z
δ = −β · (i · α)
z:=[x+1]
list α z
x ↦ i, z
[x+1]:=y
x ↦ i, y
δ = −(i · β) · α
list (i · β) x
∃β
list β x
δ = −β · α
y:=x


[x+1]:=y
x ↦ i, y
δ = −(i · β) · α
list (i · β) x
∃β
list β x
δ = −β · α
y:=x
x:=z
list β y
list α x
}
x = 0
α = ϵ
δ = −β
list (−δ) y

Talk outline

1. The problem
2. Towards a solution
3. A big proof
4. Fun with quantifiers
5. What about concurrency?

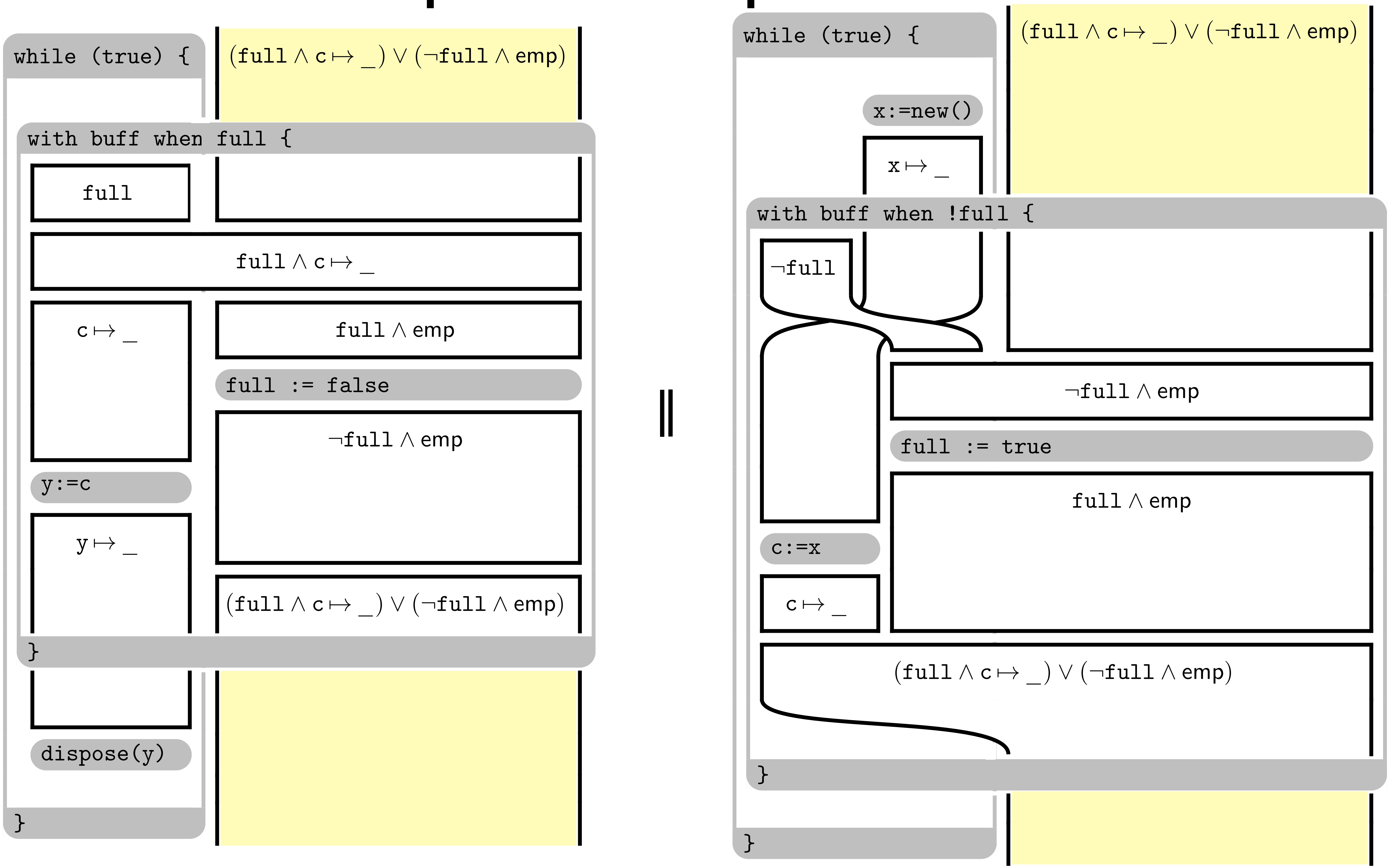
Example: one-place buffer

```
while (true) {
  with buff when full {
    full := false;
    y := c;
  }
  dispose(y);
}
```

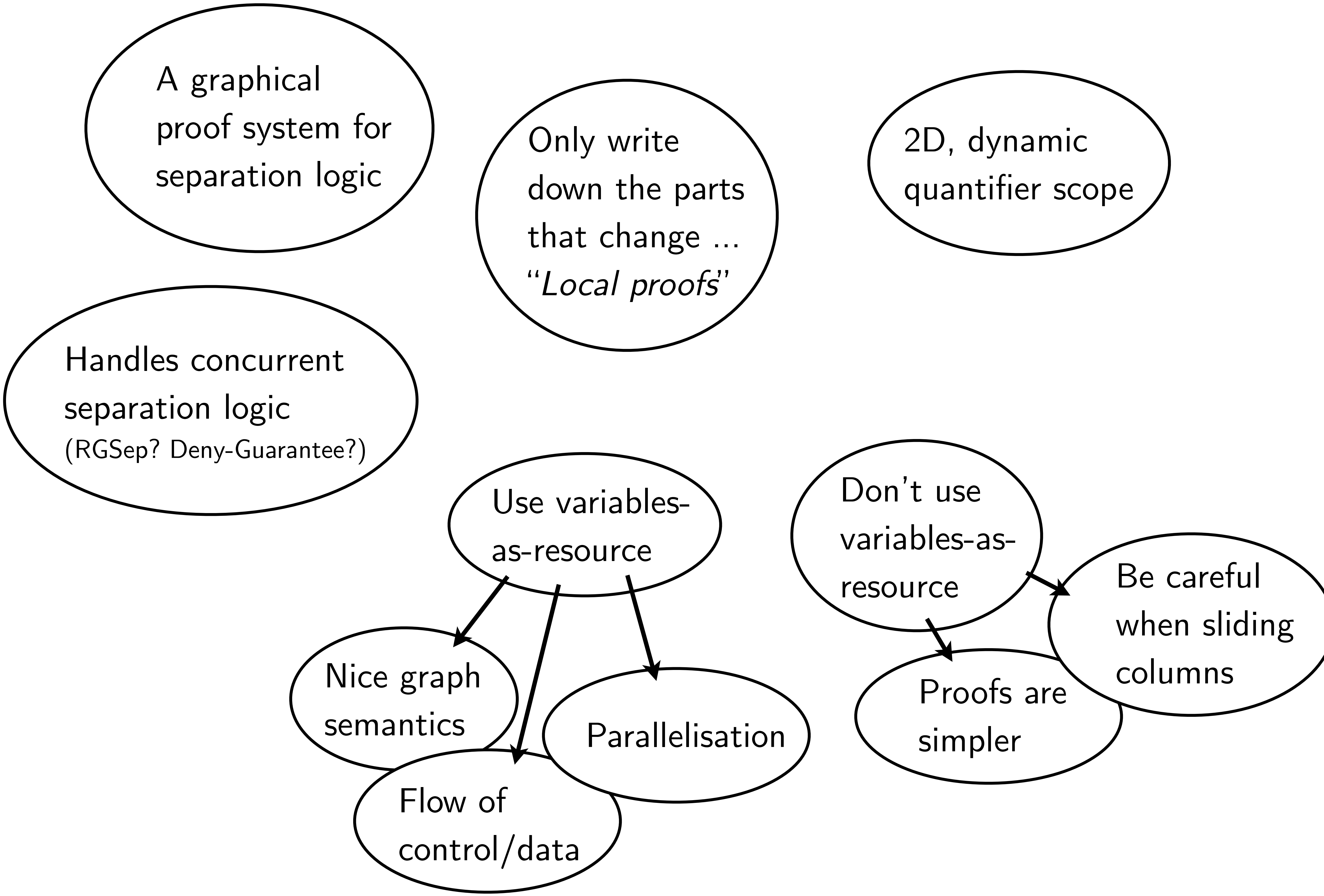
‖

```
while (true) {
  x := new();
  with buff when ¬full {
    full := true;
    c := x;
  }
}
```

Example: one-place buffer



In conclusion



Thanks for listening

References & Further reading

Jules Bean. *Ribbon Proofs - A Proof System for the Logic of Bunched Implications.* PhD thesis, available from http://www.dcs.qmul.ac.uk/tech_reports/RR-06-01.pdf

Richard Bornat, Matthew Parkinson, Cristiano Calcagno. *Variables as Resource in Hoare Logics.* In LICS 2006.

Josh Berdine, Cristiano Calcagno, Peter O'Hearn. *Symbolic Execution with Separation Logic.* In APLAS 2005.

John C. Reynolds. *Separation Logic: A Logic for Shared Mutable Data Structures.* In LICS 2002.

Peter O'Hearn. *Resources, Concurrency and Local Reasoning.* In CONCUR 2004.