Ribbon Proofs for Separation Logic

John Wickerson

Joint work with
Mike Dodds (University of York) and
Matthew Parkinson (Microsoft Research Cambridge)

Imperial College
18 November 2014

Communications of the ACM October, 1969

An Axiomatic Basis for Computer Programming

C. A. R. Hoare

Line number	Formal proof	Justification
1	$\mathbf{true} \supset x = x + y \times 0$	Lemma 1
2	$x = x + y \times 0 \{r := x\} x = r + y \times 0$	D0
3	$x = r + y \times 0 \ \{q := 0\} \ x = r + y \times q$	D0
4	$\mathbf{true} \ \{r := x\} \ x = r + y \times 0$	D1 (1, 2)
5	$\mathbf{true} \ \{r := x; \ q := 0\} \ x = r + y \times q$	D2 (4, 3)
6	$x = r + y \times q \wedge y \leqslant r \supset x = (r-y) + y \times (1+q)$	Lemma 2
7	$x = (r-y) + y \times (1+q)\{r := r-y\} x = r + y \times (1+q)$	D0
8	$x = r + y \times (1+q)\{q := 1+q\} x = r + y \times q$	D0
9	$x = (r-y) + y \times (1+q)\{r := r-y; \ q := 1+q\} \ x = r + y \times q$	D2 (7, 8)
10	$x = r + y \times q \wedge y \leqslant r \ \{r := r-y; \ q := 1+q\} \ x = r + y \times q$	D1 (6, 9)
11	$x = r + y \times q \ \{\mathbf{while} \ y \leqslant r \ \mathbf{do} \ (r := r-y; \ q := 1+q)\} \ \neg y \leqslant r \wedge x = r + y \times q$	D3 (10)
12	$\mathbf{true} \ \{((r := x; \ q := 0); \ \mathbf{while} \ y \leqslant r \ \mathbf{do} \ (r := r-y; \ q := 1+q))\} \ \neg y \leqslant r \wedge x = r + y \times q$	D2 (5, 11)

```
begin
  comment This program operates on an array A[1:N], and a
    value of f(1 ≤ f ≤ N). Its effect is to rearrange the elements
    of A in such a way that:
      ∀p,q(1≤p≤f≤q≤N⊃A[p]≤A[f]≤A[q]);
  integer m, n; comment
      m ≤ f & ∀p,q(1≤p<m≤q≤N⊃A[p]≤A[q]),
      f ≤ n & ∀p,q(1≤p≤n<q≤N⊃A[p]≤A[q]);
   m := 1; n := N;
  while m < n do
  begin integer r, i, j, w;
    comment
        m ≤ i & ∀p(1≤p<i⊃A[p]≤r),
        j ≤ n & ∀q(j<q≤N⊃r≤A[q]);
      r := A[f]; i := m; j := n;
    while i ≤ j do
    begin while A[i] < r do i := i + 1;
      while r < A[j] do j := j − 1
      comment A[j] ≤ r ≤ A[i];
      if i ≤ j then
      begin w := A[i]; A[i] := A[j]; A[j] := w;
        comment A[i] ≤ r ≤ A[j];
        i := i + 1; j := j − 1;
      end
    end increase i and decrease j;
    if f ≤ j then n := j
  else if i ≤ f then m := i
    else go to L
  end reduce middle part;
L:
end Find
```

Communications of the ACM January, 1971

Proof of a Program: FIND

C. A. R. Hoare
Queen's University, Belfast, Ireland*

Acta Informatica 6, 319—340 (1976)

An Axiomatic Proof Technique for Parallel Programs I*

Susan Owicki and David Gries

$\{x=0\}$

S: **cobegin** $\{x=0\}$

$\{x=0 \lor x=2\}$

$S1$: **await true then** $x := x+1$

$\{Q1: x=1 \lor x=3\}$

//

$\{x=0\}$

$\{x=0 \lor x=1\}$

$S2$: **await true then** $x := x+2$

$\{Q2: x=2 \lor x=3\}$

coend

$\{(x=1 \lor x=3) \land (x=2 \lor x=3)\}$

$\{x=3\}$

Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science (LICS'02)

Separation Logic: A Logic for Shared Mutable Data Structures

John C. Reynolds[*]

$$\{\exists \alpha, \beta.\ (\mathbf{list}\ \alpha\,(\mathsf{i}, \mathbf{nil}) \ast \mathbf{list}\ \beta\,(\mathsf{j}, \mathbf{nil}))$$
$$\quad \wedge\ \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta \wedge \mathsf{i} \neq \mathbf{nil}\}$$
$$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathbf{list}\ \mathsf{a}{\cdot}\alpha\,(\mathsf{i}, \mathbf{nil}) \ast \mathbf{list}\ \beta\,(\mathsf{j}, \mathbf{nil}))$$
$$\quad \wedge\ \alpha_0^\dagger = (\mathsf{a}{\cdot}\alpha)^\dagger{\cdot}\beta\}$$
$$\{\exists \mathsf{a}, \alpha, \beta, \mathsf{k}.\ (\mathsf{i} \mapsto \mathsf{a}, \mathsf{k} \ast \mathbf{list}\ \alpha\,(\mathsf{k}, \mathbf{nil}) \ast \mathbf{list}\ \beta\,(\mathsf{j}, \mathbf{nil}))$$
$$\quad \wedge\ \alpha_0^\dagger = (\mathsf{a}{\cdot}\alpha)^\dagger{\cdot}\beta\}$$
$$\mathsf{k} := [\mathsf{i} + 1]\ ;$$
$$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathsf{i} \mapsto \mathsf{a}, \mathsf{k} \ast \mathbf{list}\ \alpha\,(\mathsf{k}, \mathbf{nil}) \ast \mathbf{list}\ \beta\,(\mathsf{j}, \mathbf{nil}))$$
$$\quad \wedge\ \alpha_0^\dagger = (\mathsf{a}{\cdot}\alpha)^\dagger{\cdot}\beta\}$$
$$[\mathsf{i} + 1] := \mathsf{j}\ ;$$
$$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathsf{i} \mapsto \mathsf{a}, \mathsf{j} \ast \mathbf{list}\ \alpha\,(\mathsf{k}, \mathbf{nil}) \ast \mathbf{list}\ \beta\,(\mathsf{j}, \mathbf{nil}))$$
$$\quad \wedge\ \alpha_0^\dagger = (\mathsf{a}{\cdot}\alpha)^\dagger{\cdot}\beta\}$$
$$\{\exists \mathsf{a}, \alpha, \beta.\ (\mathbf{list}\ \alpha\,(\mathsf{k}, \mathbf{nil}) \ast \mathbf{list}\ \mathsf{a}{\cdot}\beta\,(\mathsf{i}, \mathbf{nil}))$$
$$\quad \wedge\ \alpha_0^\dagger = \alpha^\dagger{\cdot}\mathsf{a}{\cdot}\beta\}$$
$$\{\exists \alpha, \beta.\ (\mathbf{list}\ \alpha\,(\mathsf{k}, \mathbf{nil}) \ast \mathbf{list}\ \beta\,(\mathsf{i}, \mathbf{nil})) \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta\}$$
$$\mathsf{j} := \mathsf{i}\ ;\ \mathsf{i} := \mathsf{k}$$
$$\{\exists \alpha, \beta.\ (\mathbf{list}\ \alpha\,(\mathsf{i}, \mathbf{nil}) \ast \mathbf{list}\ \beta\,(\mathsf{j}, \mathbf{nil})) \wedge \alpha_0^\dagger = \alpha^\dagger{\cdot}\beta\}.$$

Tiny example

```
[x]:=1;

[y]:=1;

[z]:=1;
```

$$\{\mathtt{x} \mapsto 0 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

```
[x]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

```
[y]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 0\}$$

```
[z]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 1\}$$

$$\{\mathtt{x} \mapsto 0 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

```
[x]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

```
[y]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 0\}$$

```
[z]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 1\}$$

$$\{\mathtt{x} \mapsto 0 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

```
[x]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

```
[y]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 0\}$$

```
[z]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 1\}$$

$$\{\mathtt{x} \mapsto 0 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

```
[x]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

```
[y]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 0\}$$

```
[z]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 1\}$$

$$\{\mathtt{x} \mapsto 0 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

`[x]:=1;`

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

$$\{\mathtt{y} \mapsto 0\}$$

`[y]:=1;`

$$\{\mathtt{y} \mapsto 1\}$$

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 0\}$$

`[z]:=1;`

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 1\}$$

frame $\mathtt{x} \mapsto 1 * \mathtt{z} \mapsto 0$ (brackets the lines from $\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$ to $\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 0\}$)

small axiom for heap update (brackets $\{\mathtt{y} \mapsto 0\}$ `[y]:=1;` $\{\mathtt{y} \mapsto 1\}$)

$$\{\mathtt{x} \mapsto 0 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

```
[x]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

```
[y]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 0\}$$

```
[z]:=1;
```

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 1\}$$

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

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

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

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

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

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

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

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

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

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

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

$\{\mathtt{x} \mapsto 0 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$

`[x]:=1;`

$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$

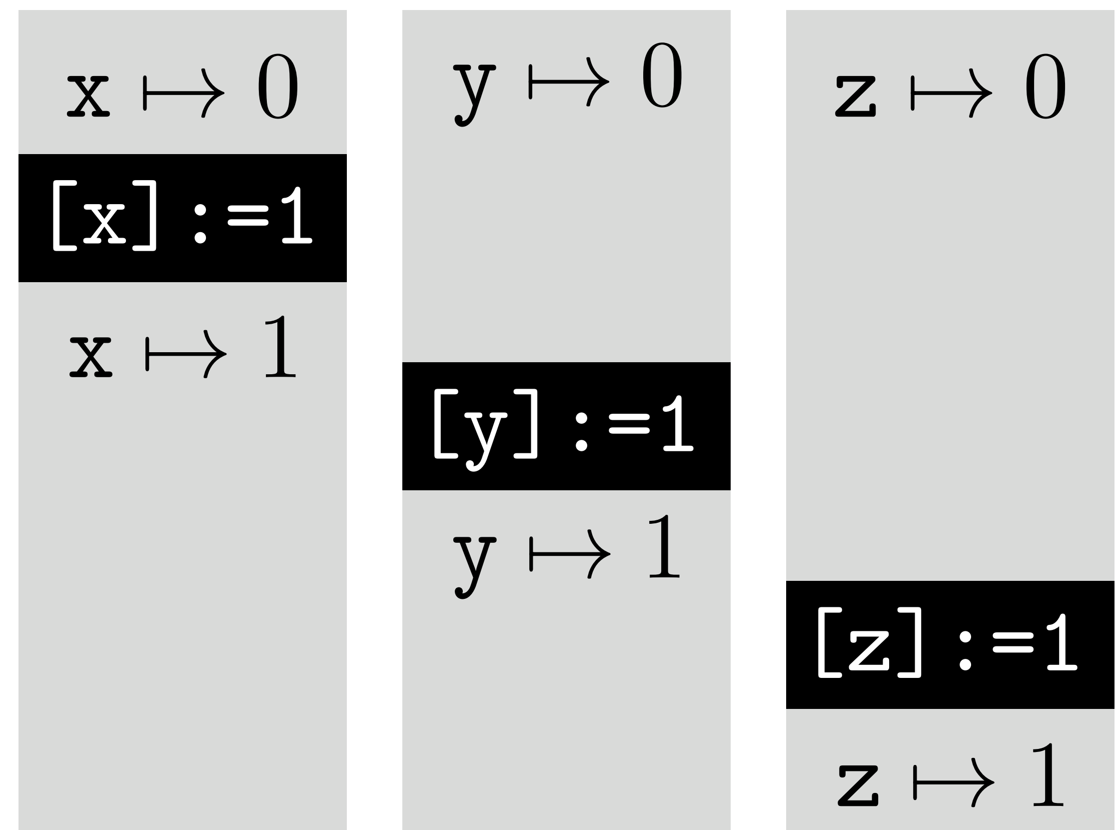
`[y]:=1;`

$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 0\}$

`[z]:=1;`

$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 1\}$

A proof outline



A ribbon proof

$$\{\mathtt{x} \mapsto 0 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

`[x]:=1;`

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

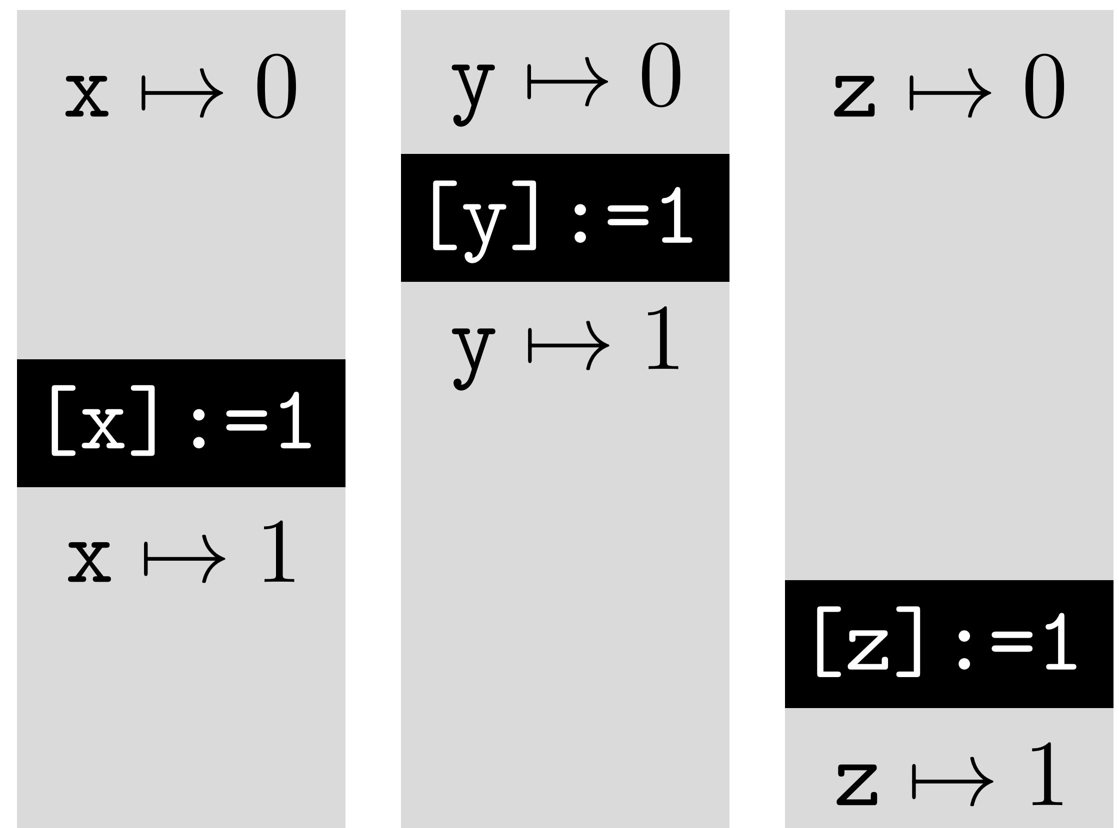
`[y]:=1;`

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 0\}$$

`[z]:=1;`

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 1\}$$

A proof outline



A ribbon proof

$$\{\mathtt{x} \mapsto 0 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

`[x]:=1;`

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 0 * \mathtt{z} \mapsto 0\}$$

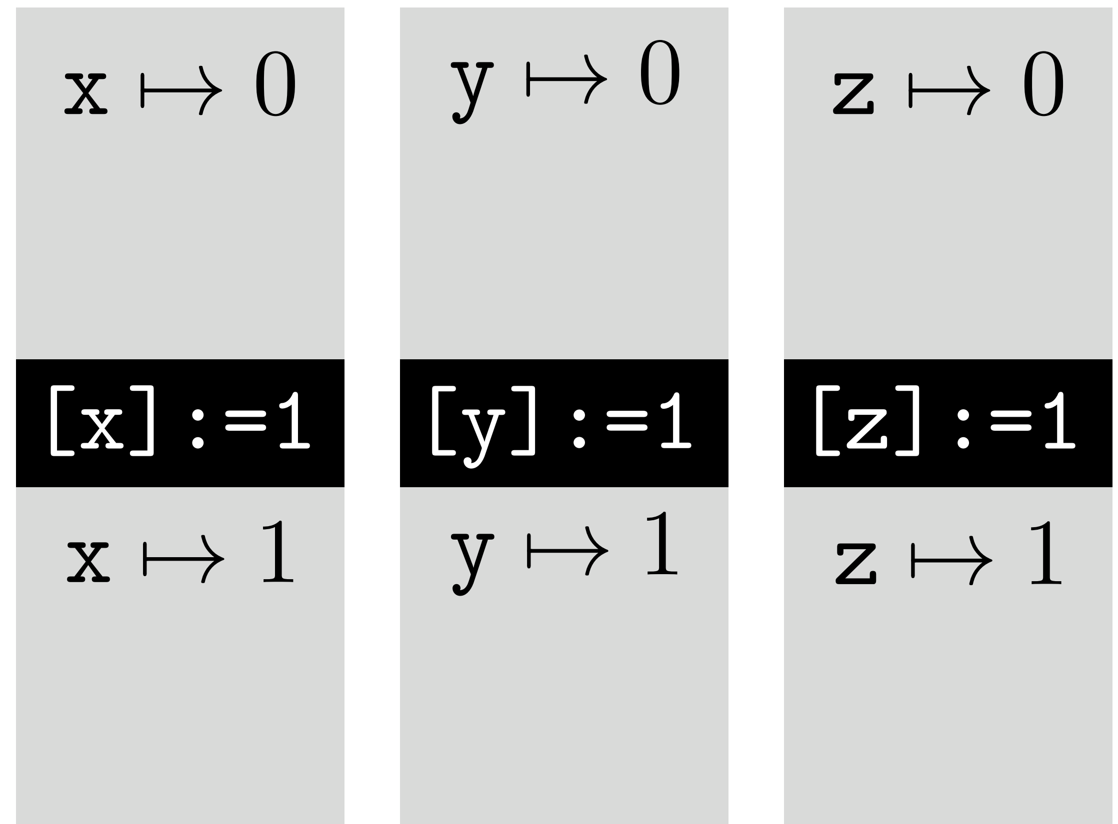
`[y]:=1;`

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 0\}$$

`[z]:=1;`

$$\{\mathtt{x} \mapsto 1 * \mathtt{y} \mapsto 1 * \mathtt{z} \mapsto 1\}$$

A proof outline



A ribbon proof

Example: in-place list reversal

list x

$$list\ x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * list\ x')$$

list x

```
y := nil;
while (x != nil) {
   z := [x+1];
   [x+1] := y;
   y := x;
   x := z;
}
```

list y

list y

$$\mathit{list}\ x \stackrel{\text{def}}{=} (x \doteq \mathtt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * \mathit{list}\ x')$$

list x

```
y := nil;
while (x != nil) {
   z := [x+1];
   [x+1] := y;
   y := x;
   x := z;
}
```

list y

list x

```
y:=nil
```

list y

list y

$$\mathit{list}\, x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * \mathit{list}\, x')$$

list x

```
y := nil;
while (x != nil) {
   z := [x+1];
   [x+1] := y;
   y := x;
   x := z;
}
```

list y

list x

`y:=nil`

list y

```
while (x!=nil) {
}
```

list y

$$
\mathit{list}\,x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * \mathit{list}\,x')
$$

list x

```
y := nil;
while (x != nil) {
    z := [x+1];
    [x+1] := y;
    y := x;
    x := z;
}
```

list y

list x

```
y:=nil
```

list y

```
while (x!=nil) {
```

$\mathtt{x} \mathrel{\dot{\neq}} \mathtt{nil}$ | *list* x | *list* y

```
}
```

list y

$$list\ x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * list\ x')$$

list x

```
y := nil;
while (x != nil) {
    z := [x+1];
    [x+1] := y;
    y := x;
    x := z;
}
```

list y

list x

`y:=nil`

list y

`while (x!=nil) {`

x $\dot{\neq}$ nil | *list* x | *list* y

list x | *list* y

`}`

list y

$$\mathit{list}\ x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * \mathit{list}\ x')$$

list x

```
y := nil;
while (x != nil) {
   z := [x+1];
   [x+1] := y;
   y := x;
   x := z;
}
```

list y

list x

`y:=nil`

list y

`while (x!=nil) {`

$\texttt{x} \mathrel{\dot{\neq}} \texttt{nil}$ | *list* x | *list* y

list x | *list* y

`}`

$\texttt{x} \doteq \texttt{nil}$ | *list* x | *list* y

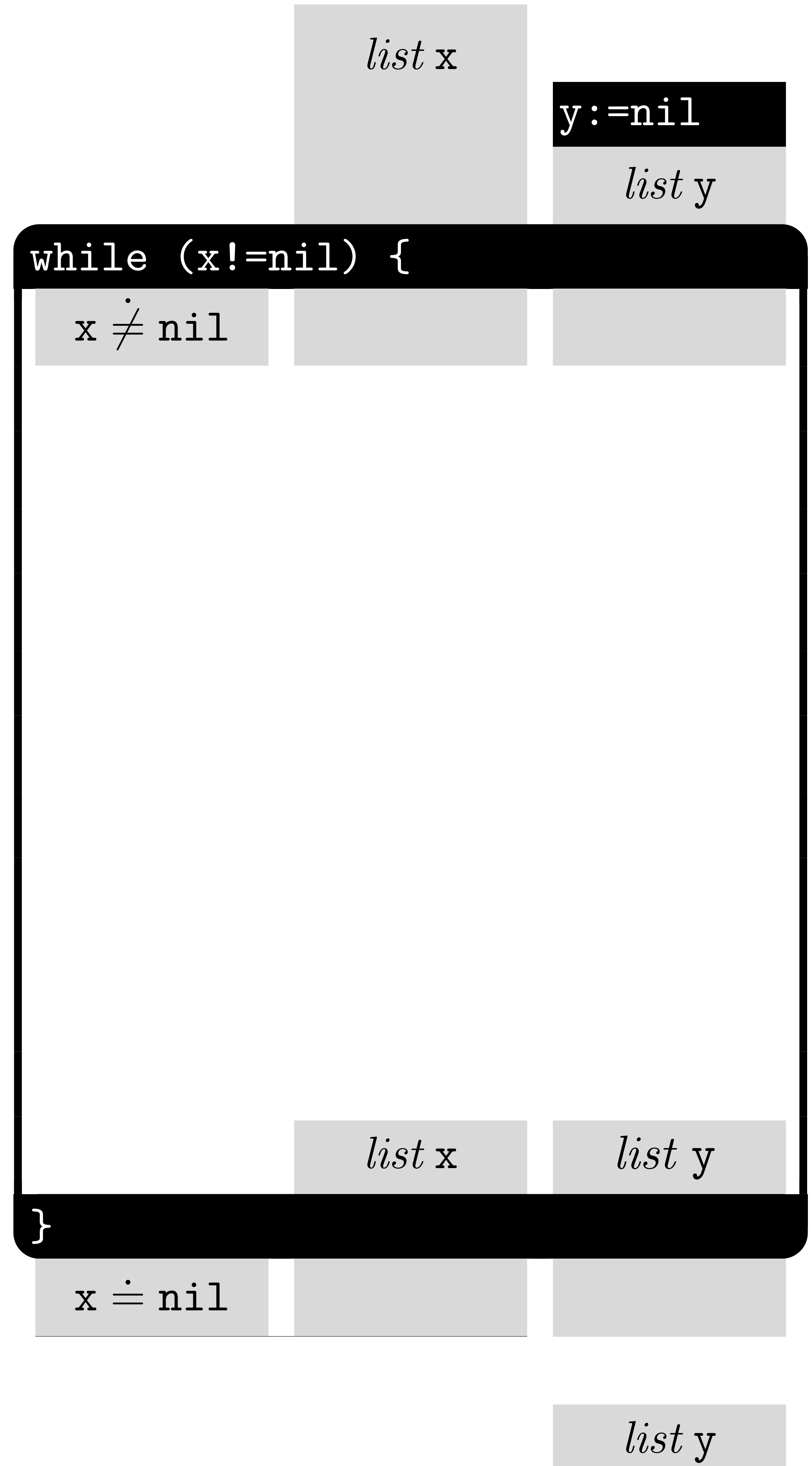
list y

$$\mathit{list}\,x \stackrel{\text{def}}{=} (x \doteq \mathtt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * \mathit{list}\,x')$$

list x

```
y := nil;
while (x != nil) {
    z := [x+1];
    [x+1] := y;
    y := x;
    x := z;
}
```

list y



$$\mathit{list}\ x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * \mathit{list}\ x')$$

list x

```
y := nil;
while (x != nil) {
    z := [x+1];
    [x+1] := y;
    y := x;
    x := z;
}
```

list y

$$list\ x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * list\ x')$$

list x

```
y := nil;
while (x != nil) {
   z := [x+1];
   [x+1] := y;
   y := x;
   x := z;
}
```

list y

list x

`y:=nil`

list y

`while (x!=nil) {`

$\mathtt{x} \not\doteq \mathtt{nil}$

Unfold *list* def

$\exists Z.\, \mathtt{x} \mapsto \_, Z * list\ Z$

`z:=[x+1]`

list z | $\mathtt{x} \mapsto \_, \mathtt{z}$

list x | *list* y

`}`

$\mathtt{x} \doteq \mathtt{nil}$

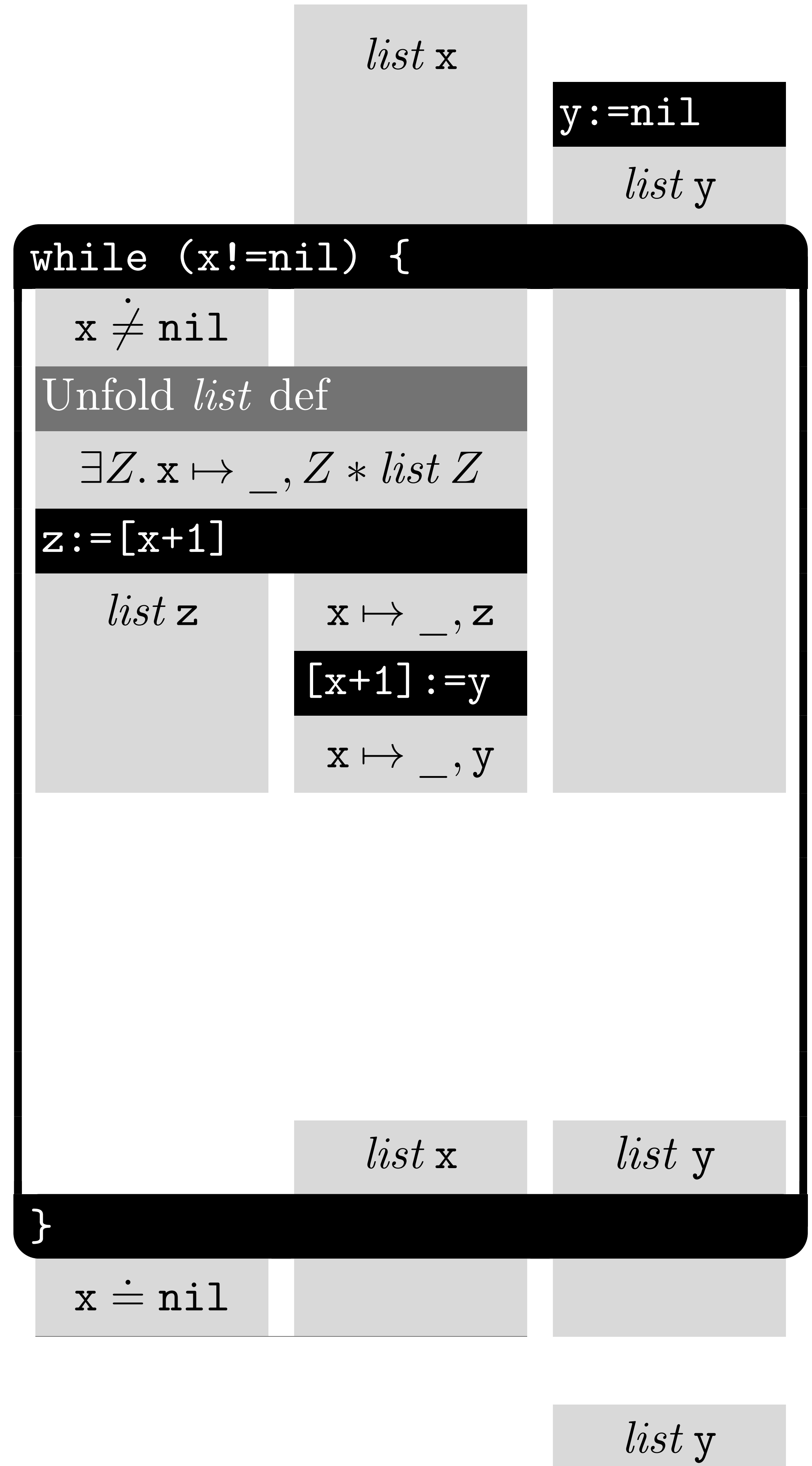
list y

$$\mathit{list}\ x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\ x \mapsto \_, x' * \mathit{list}\ x')$$

list x

```
y := nil;
while (x != nil) {
   z := [x+1];
   [x+1] := y;
   y := x;
   x := z;
}
```

list y

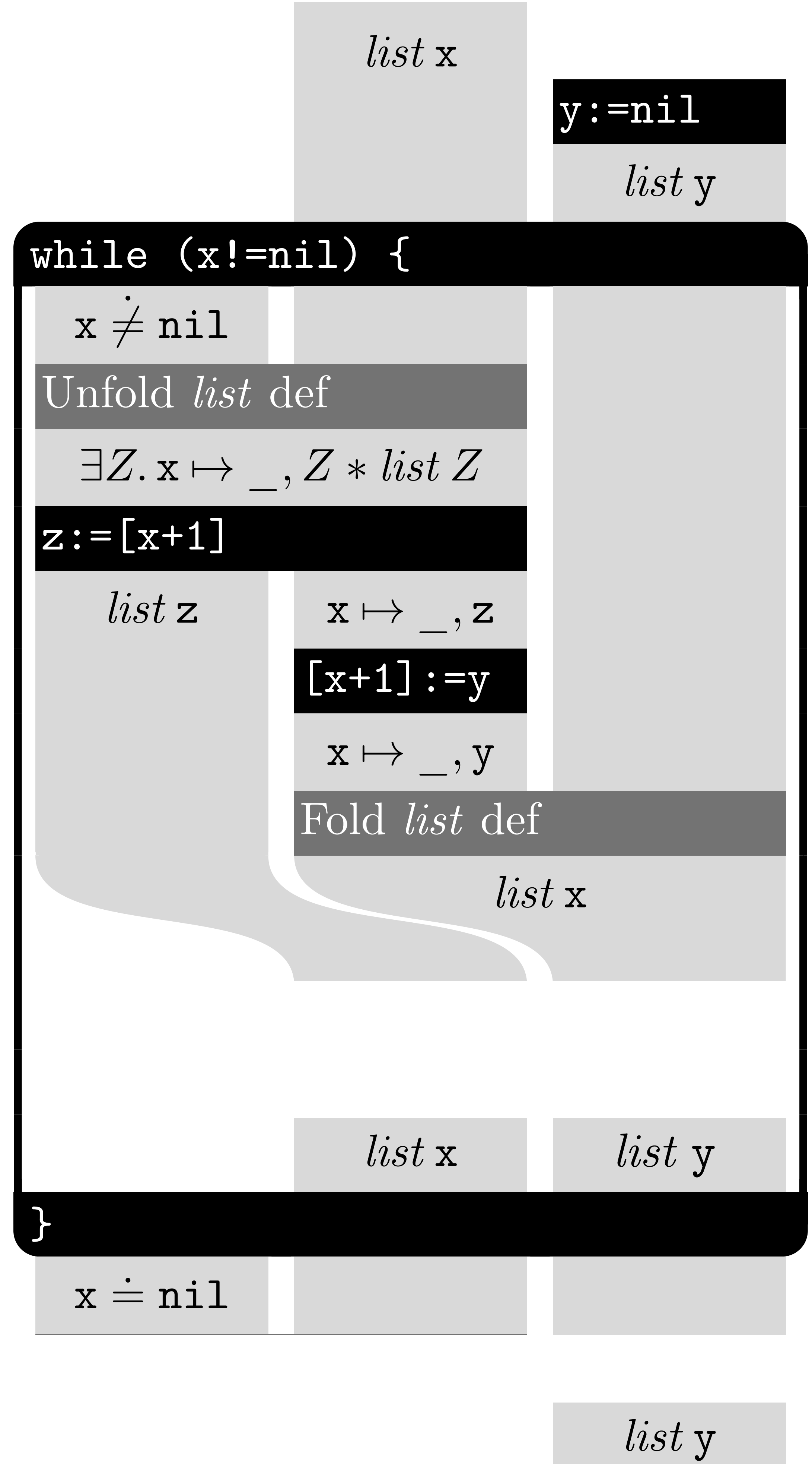


$$\mathit{list}\,x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * \mathit{list}\,x')$$

list x

```
y := nil;
while (x != nil) {
   z := [x+1];
   [x+1] := y;
   y := x;
   x := z;
}
```

list y

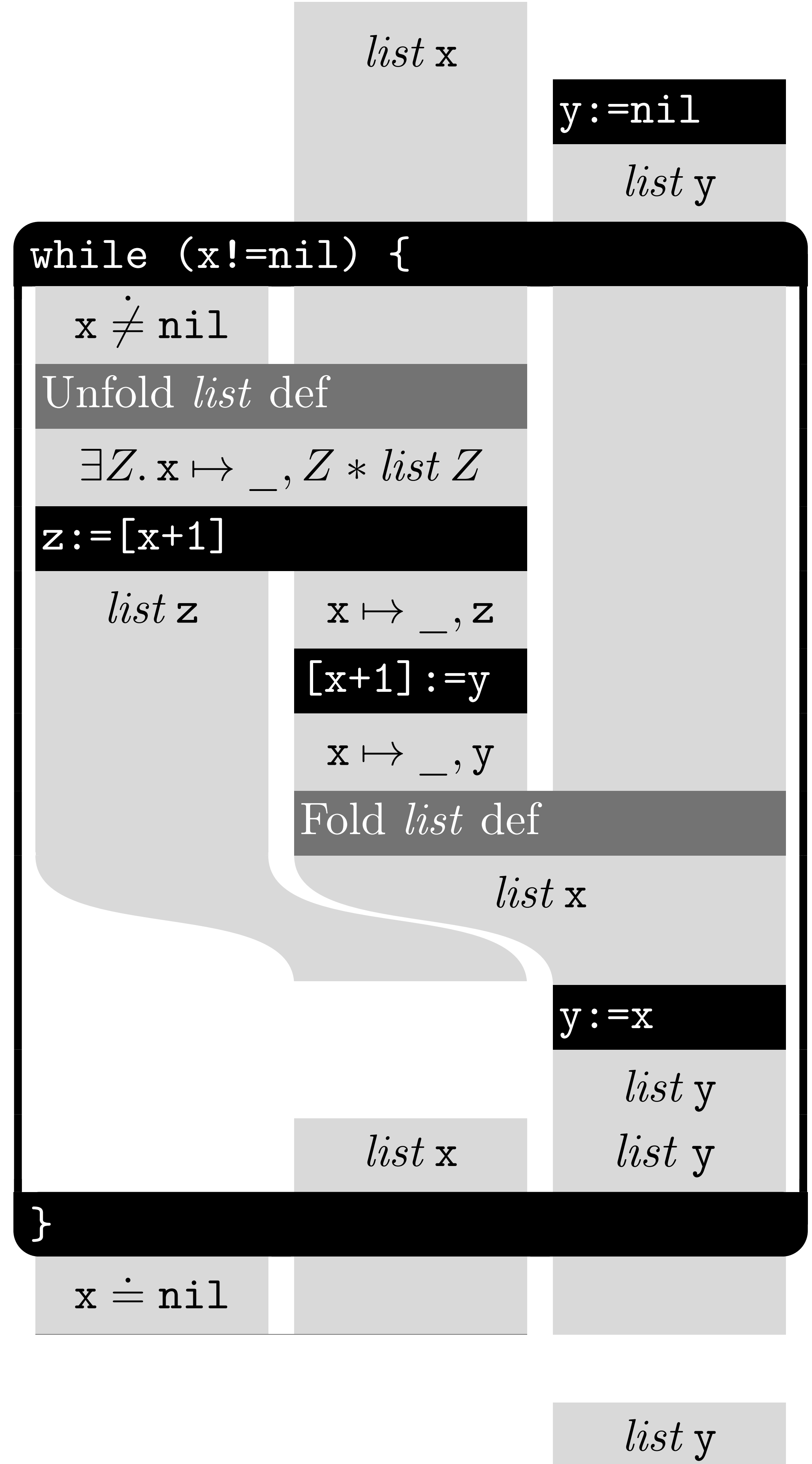


$$\mathit{list}\ x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\ x \mapsto \_, x' * \mathit{list}\ x')$$

list x

```
y := nil;
while (x != nil) {
   z := [x+1];
   [x+1] := y;
   y := x;
   x := z;
}
```

list y

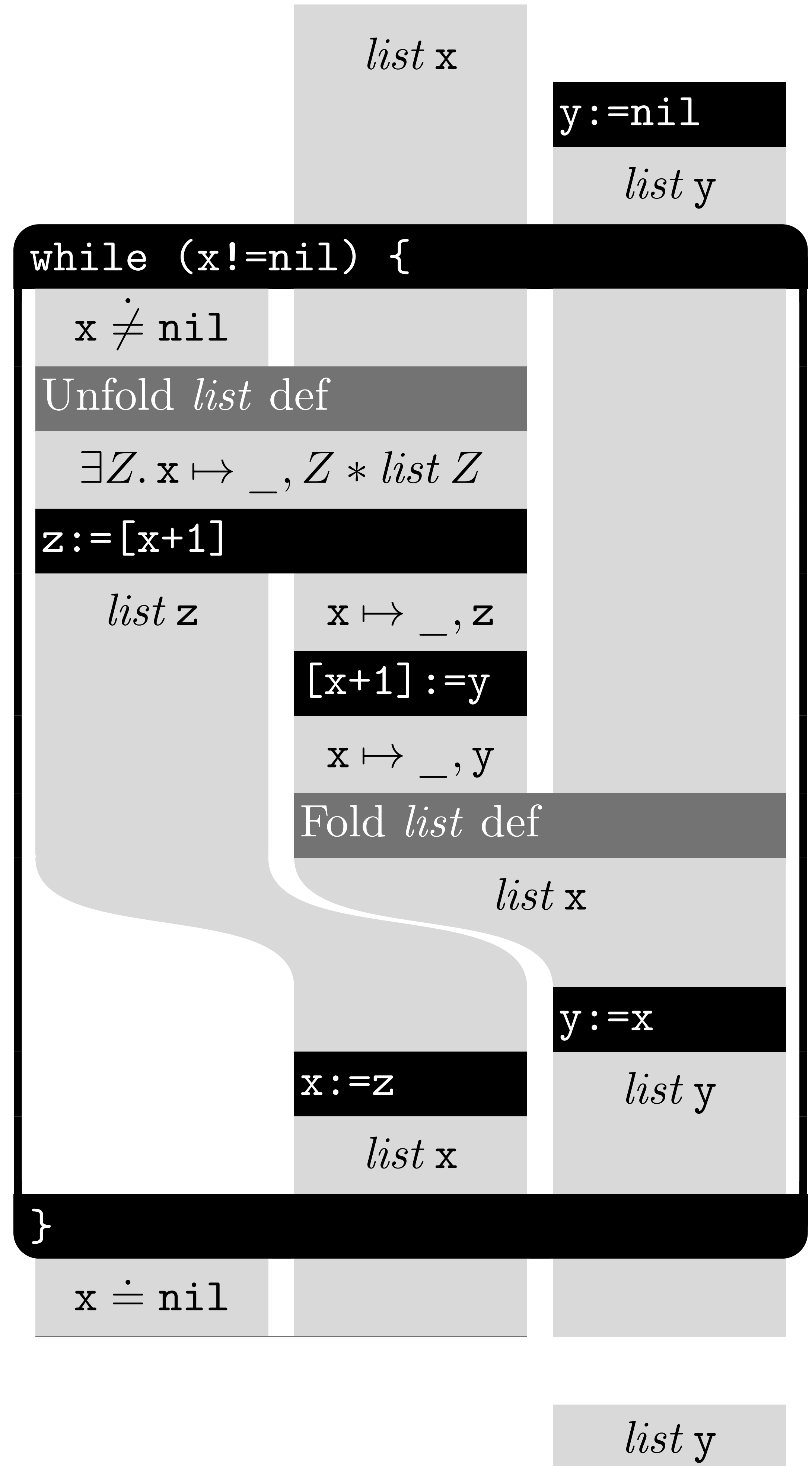


$$\mathit{list}\ x \stackrel{\text{def}}{=} (x \doteq \mathtt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * \mathit{list}\ x')$$

list x

```
y := nil;
while (x != nil) {
    z := [x+1];
    [x+1] := y;
    y := x;
    x := z;
}
```

list y

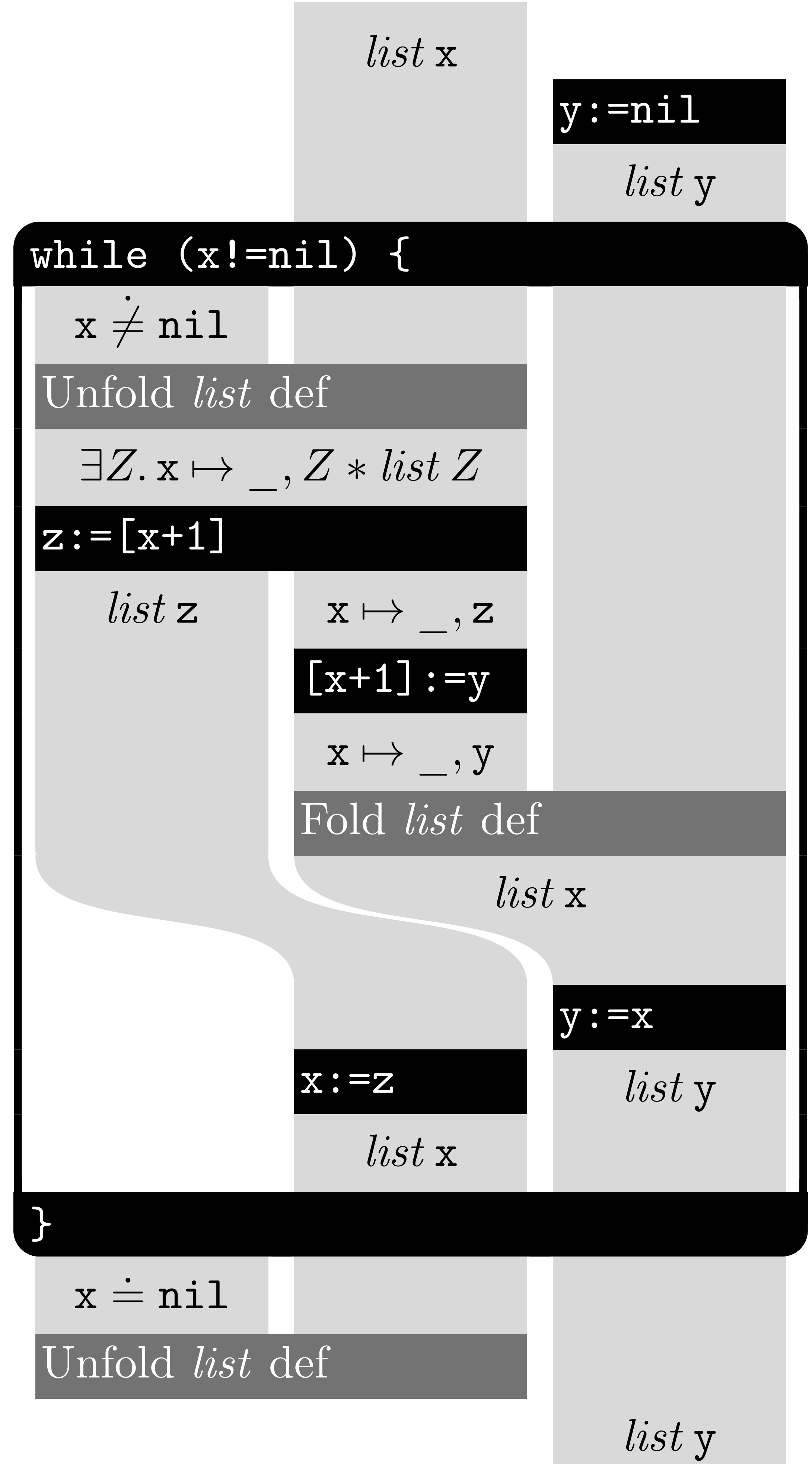


$$list\ x \stackrel{\text{def}}{=} (x \doteq \texttt{nil}) \vee (\exists x'.\, x \mapsto \_, x' * list\ x')$$

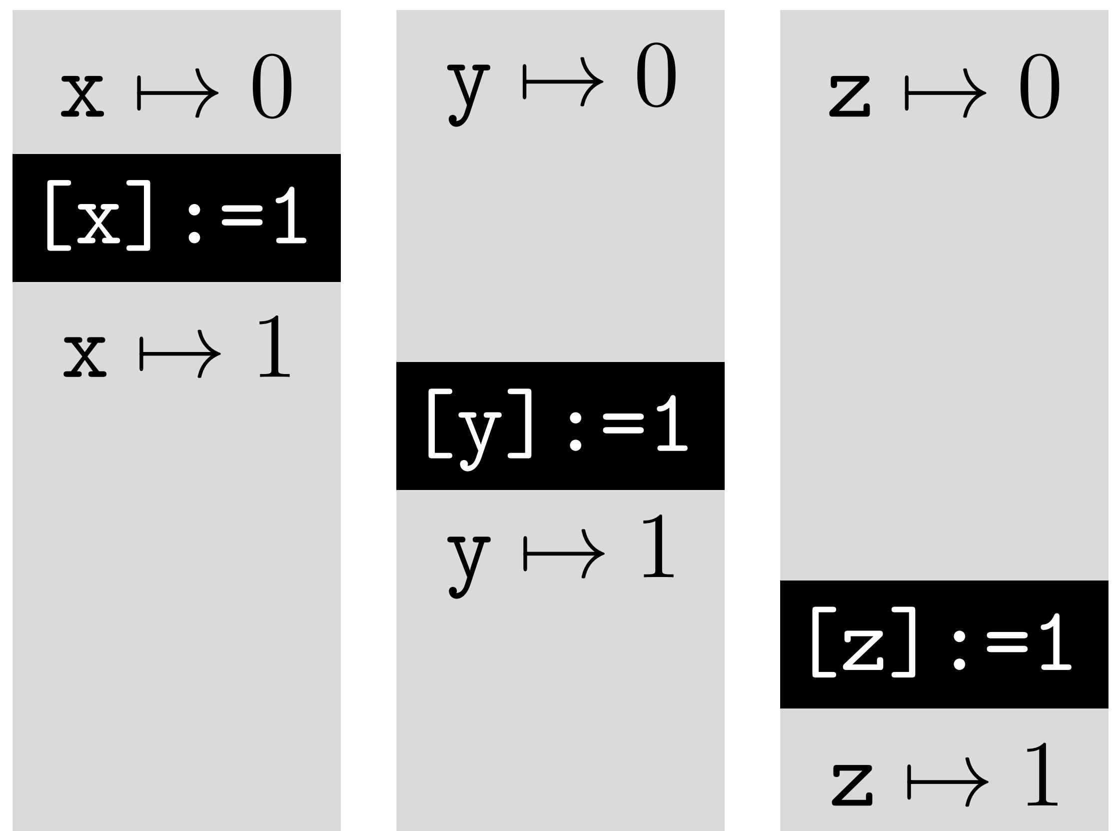
list x

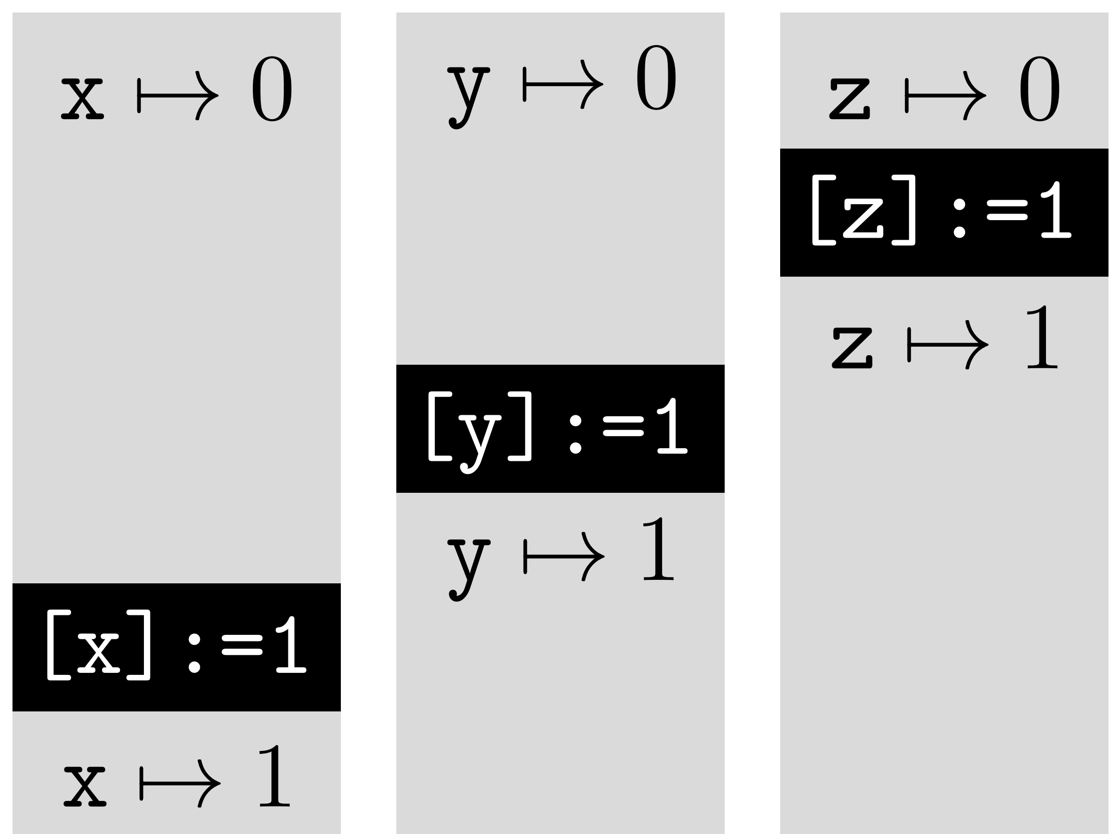
```
y := nil;
while (x != nil) {
    z := [x+1];
    [x+1] := y;
    y := x;
    x := z;
}
```

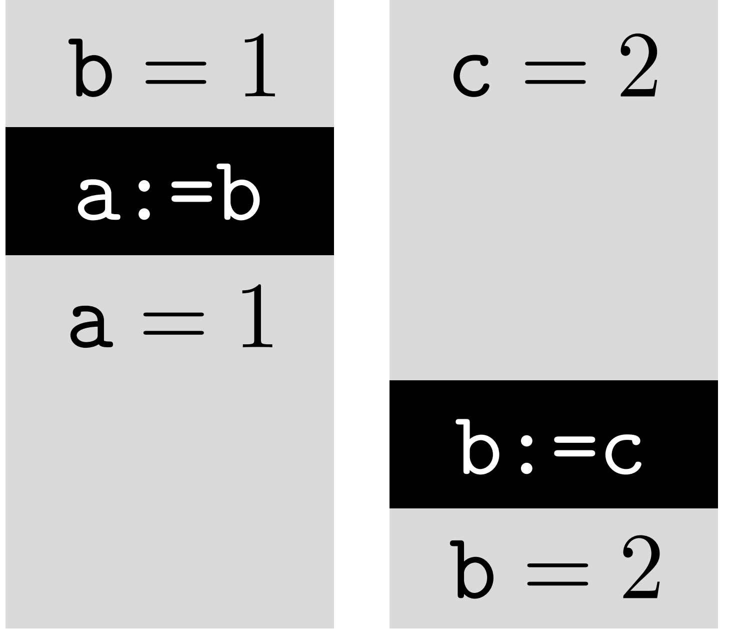
list y

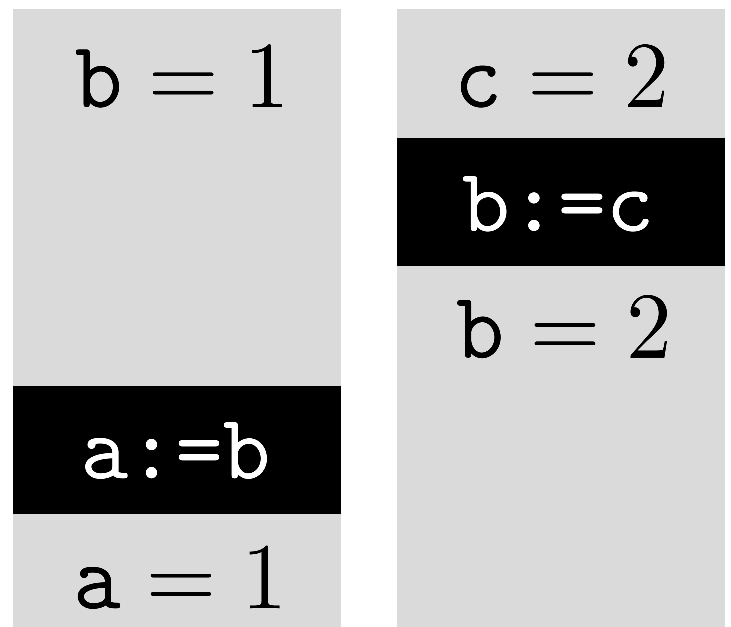


Dealing with program variables


x ↦ 0
[x]:=1
x ↦ 1
y ↦ 0
[y]:=1
y ↦ 1
z ↦ 0
[z]:=1
z ↦ 1

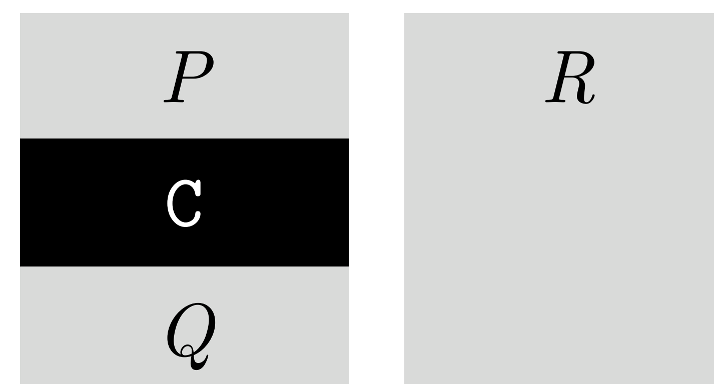

x ↦ 0
[x]:=1
x ↦ 1
y ↦ 0
[y]:=1
y ↦ 1
z ↦ 0
[z]:=1
z ↦ 1

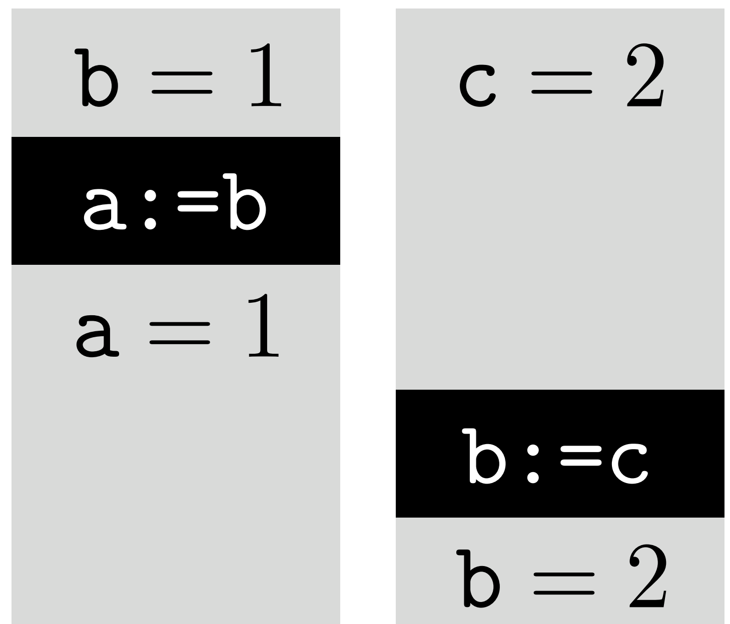

b = 1
a:=b
a = 1
c = 2
b:=c
b = 2

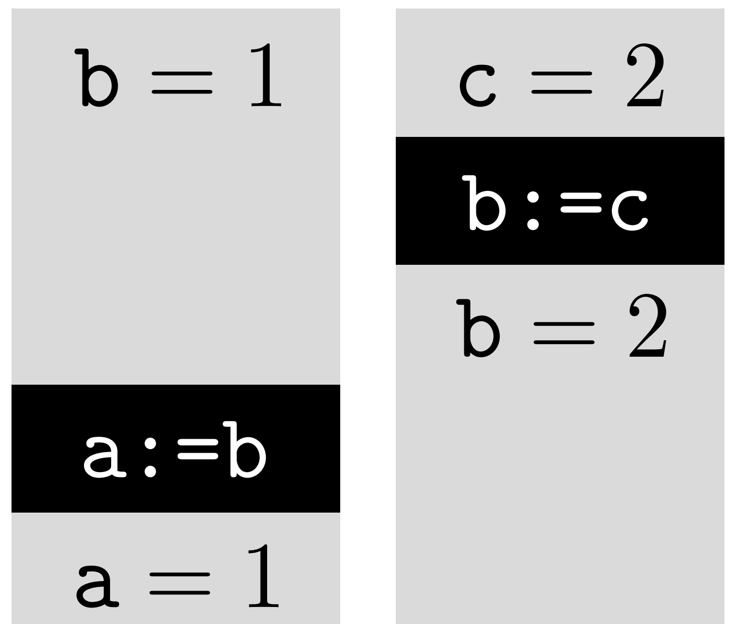

b = 1
a:=b
a = 1
c = 2
b:=c
b = 2

$$\frac{\{P\}\ \mathtt{C}\ \{Q\}}{\{P * R\}\ \mathtt{C}\ \{Q * R\}}$$

providing $fv(R) \cap \mathit{modified}(\mathtt{C}) = \{\}$

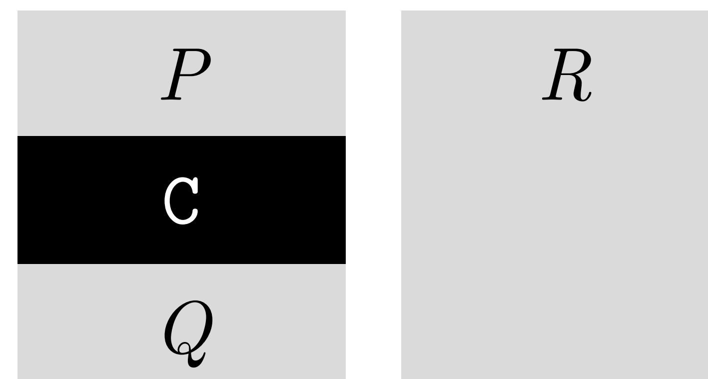



b = 1
a:=b
a = 1
c = 2
b:=c
b = 2


b = 1
a:=b
a = 1
c = 2
b:=c
b = 2

$$\frac{\{P\}\ \mathtt{C}\ \{Q\}}{\{P * R\}\ \mathtt{C}\ \{Q * R\}}$$

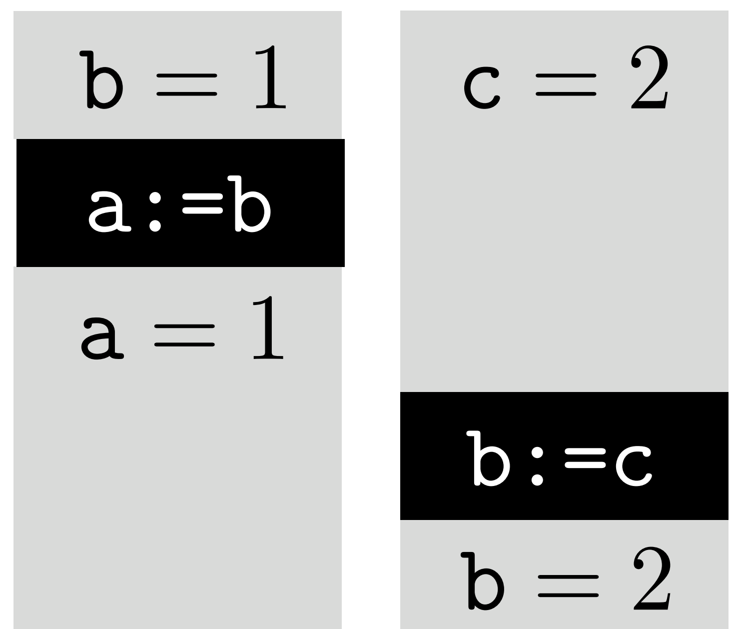
~~providing $fv(R) \cap modified(\mathtt{C}) = \{\}$~~

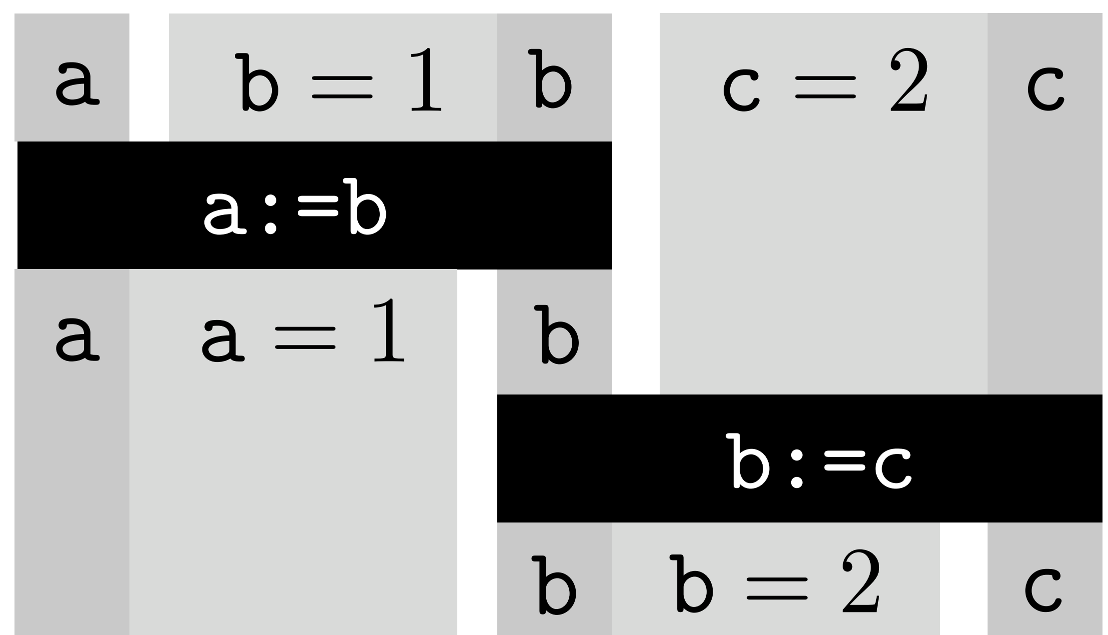


Electronic Notes in Theoretical Computer Science 155 (2006)

Variables as Resource in Separation Logic

Richard Bornat Cristiano Calcagno Hongseok Yang


b = 1
a:=b
a = 1
c = 2
b:=c
b = 2


a
b = 1
b
c = 2
c
a:=b
a
a = 1
b
b:=c
b
b = 2
c

Conclusion

- Scalable and readable separation logic proofs
- Possible application to parallelisation, providing side-conditions on the Frame rule are dealt with (e.g. by variables-as-resource)