1 General Info

$$\text{array}(a, [x_1, x_2, \dots x_n]) \triangleq a \mapsto x_1 * (a+4)a \mapsto x_2 * \dots * (a+4n) \mapsto x_n$$

- For stacks:

$$a \rightsquigarrow nil \triangleq a \mapsto \text{NULL}$$
$$a \rightsquigarrow [x_1 \dots, x_n] \triangleq a \mapsto (x_1, a_1) * \dots a_{n-1} \mapsto (x_n, a_n) * a_n \mapsto \text{NULL}$$

- For queues:

$$(hd, tl) \rightsquigarrow nil \triangleq hd \mapsto \text{NULL } tl \mapsto \text{NULL}$$
$$(hd, tl) \rightsquigarrow [x_n \dots, x_1] \triangleq hd \mapsto (x_1, a_1) * \dots a_{n-1} \mapsto tl = (\text{NULL }, x_n)$$

$l \bigcirc\!\!\!\rightarrow R \triangleq l \underset{0}{\Box\!\!\!\rightarrow} R$ and R is constant (i.e. doesn't depend on its variable)

1.1 Join Spawn rules

$$\frac{\{P\}f\{Q\} \quad l \text{ fresh in } F}{\{F * P\}\text{Spawn f } \{F * l \bigcirc\!\!\!\rightarrow Q\}}\ \text{spwn}$$

$$\frac{}{\{l \bigcirc\!\!\!\rightarrow Q\}\text{Join(l) } \{Q\}}\ \text{join}$$

1.2 Histories

$$Hist \triangleq \mathbb{N} \nrightarrow \text{list } * \text{list}$$
$$t \hookrightarrow_h (l_1, l_2) \triangleq h[t] = (l_1, l_2)$$

- bounded $h \triangleq \exists t. \forall t' > t, t' \notin h$
- last $h \triangleq \min\{t | \forall t' > t, t' \notin h\}$
- listof $h \triangleq \pi_2(h[\text{last } h])$ (i.e. (last h) $\hookrightarrow$ (_, listof h))
- continuous $h \triangleq \forall t. t \in h \wedge (t+1) \in h \rightarrow \exists l. t \hookrightarrow (\_, l) \wedge (t+1) \hookrightarrow (l, \_)$
- gapless $h \triangleq \forall t \in h \rightarrow \forall t' < t, t' \in h$
- init $h \triangleq 0 \hookrightarrow (\epsilon, \_)$
- stacklike $h \triangleq \forall t \in h \rightarrow \exists l, x, t \hookrightarrow (x :: l, l) \vee t \hookrightarrow (l, x :: l)$
- queuelike $h \triangleq \forall t \in h \rightarrow \exists l, x, t \hookrightarrow (l, x :: l) \vee t \hookrightarrow (l :: x, l)$
- stack_history $h \triangleq$ continuous $h \wedge$gapless $h \wedge$bounded $v \wedge$init $h \wedge$stacklike h
- queue_history $h \triangleq$ continuous $h \wedge$gapless $h \wedge$bounded $v \wedge$init $h \wedge$queuelike h

2 Multiple-thread counter.

```
int main() {
  $\{\ \mathsf{emp}\}$
  a = malloc (n);
  $\{\ \mathsf{array}(a, [\_, \_, \ldots, \_]_n\}$
  (l, c) = malloc (LOCK_SIZE);
  $\{\ l \mapsto \_ * c \mapsto \_ * \mathsf{array}(a, [\_, \_, \ldots, \_]_n\}$
  c = 0;
  $\{\ l \mapsto \_ * c \mapsto 0 * \mathsf{array}(a, [\_, \_, \ldots, \_]_n\}$
  MakeLock(l);
  $\{\ l\square\!\xrightarrow[0]{1} R * c \mapsto 0 * \mathsf{Hold}\ l, R, 0 * \mathsf{array}(a, [\_, \_, \ldots, \_]_n\} \quad //R = \lambda v.c \mapsto v$
  Release(l);
  $\{\ l\square\!\xrightarrow[0]{1} R * \mathsf{array}(a, [\_, \_, \ldots, \_]_n\}$
  $\{\ \mathop{\bigstar}_{j=0}^{n-1} l\square\!\xrightarrow[0]{\frac{1}{n}} R * \mathsf{array}(a, [\_, \_, \ldots, \_]_n\}$
  for (i = 0; i < n; i ++) {
  $\{\ \mathop{\bigstar}_{j=0}^{i} l_j\bigcirc\!\!\rightarrow R_j * \mathop{\bigstar}_{j=i+1}^{n-1} l\square\!\xrightarrow[0]{\frac{1}{n}} R * \mathsf{array}(a, [l_1, \ldots, l_i, \_, \ldots \_]_n\}$
    a[i] = Spawn(incr, (l,c));
  $\{\ \mathop{\bigstar}_{j=0}^{i+1} l_j\bigcirc\!\!\rightarrow R_j * \mathop{\bigstar}_{j=i+2}^{n-1} l\square\!\xrightarrow[0]{\frac{1}{n}} R * \mathsf{array}(a, [l_1, \ldots, l_i, l_{i+1}, \_, \ldots \_]_n\}$
  }
  $\{\ \mathop{\bigstar}_{j=0}^{n} l_j\bigcirc\!\!\rightarrow R_j * \mathsf{array}(a, [l_1, \ldots, l_n]\}$
  for (i = 0; i < n; i ++)
  {
  $\{\ \mathop{\bigstar}_{j=0}^{i} R_j * \mathop{\bigstar}_{j=i}^{n} l_j\bigcirc\!\!\rightarrow R_j * \mathsf{array}(a, [l_1, \ldots, l_n]\}$
    Join(a[i]);
  $\{\ \mathop{\bigstar}_{j=0}^{i+1} R_j * \mathop{\bigstar}_{j=i+1}^{n} l_j\bigcirc\!\!\rightarrow R_j * \mathsf{array}(a, [l_1, \ldots, l_n]\}$
  }
  $\{\ \bigstar_{j=0}^{n} R_j * \mathsf{array}(a, [l_1, \ldots, l_n]\} \quad //R_j = l\square\!\xrightarrow[1]{\frac{1}{n}} R$
  $\{\ l\square\!\xrightarrow[n]{} R * \mathsf{array}(a, [l_1, \ldots, l_n]\}$
  free(a);
  $\{\ l\square\!\xrightarrow[n]{} R\}$
  Acquire(l);
  $\{\ l\square\!\xrightarrow[n]{} R * \exists v_o, c \mapsto (n + v_o) * \mathsf{Hold}\ l, R, (n + v_o)\}$
  $\{\ l\square\!\xrightarrow[n]{} R * c \mapsto n * \mathsf{Hold}\ l, R, n\}$
  ret = c;
  $\{\ \mathsf{ret} \mapsto n * l\square\!\xrightarrow[n]{} R * c \mapsto n * \mathsf{Hold}\ l, R, n\}$
  FreeLock (l);
  $\{\ \mathsf{ret} \mapsto n * l \mapsto 0 * c \mapsto n\}$
```

```
    free(l,c);
    $\{ \text{ret} \mapsto n\}$
    return ret }


void incr(l,c) {
    $\{ l\square\xrightarrow[0]{\frac{1}{n}} R \}$
    Acquire(l);
    $\{ \exists v_o, c \mapsto v_o * \text{Hold } l, R, v_o * l\square\xrightarrow[0]{\frac{1}{n}} R\}$
    ( *c)++;
    $\{ \exists v_o, c \mapsto (v_o + 1) * \text{Hold } l, R, v_o * l\square\xrightarrow[0]{\frac{1}{n}} R\}$
    Release(l);
    $\{ l\square\xrightarrow[1]{\frac{1}{n}} R\}$
}
```

3 Single Initialize / concurrent read

```
$\{\ l\Box\xrightarrow[\bot]{\pi} R\ \}$    \\  $R = \lambda v.init \mapsto 0 \wedge v = \bot \vee init \mapsto 1 * d \overset{\top - v}{\mapsto} \text{data} \wedge [\top > v]$
data *first_access(l) {
  $\{\ l\Box\xrightarrow[\bot]{\pi} R\ \}$
  Acquire(l);
  $\{\ \exists v_o, init \mapsto 0 \wedge v_o = \bot \vee init \mapsto 1 * d \overset{s_o}{\mapsto} \text{data} * \text{Hold } l, R, v_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$
 \\ where $s_o = \top - v_o$
  if(init) {
    $\{\ init \mapsto 1 * d \overset{s_o}{\mapsto} \text{data} * \text{Hold } l, R, v_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$
    $\{\ d \overset{\frac{s_o}{2}}{\mapsto} \text{data} * \big(init \mapsto 1 * d \overset{\top - (v_o + \frac{s_o}{2})}{\mapsto} \text{data}\big) * \text{Hold } l, R, v_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$
    Release(l);
    $\{\ d \overset{\frac{s_o}{2}}{\mapsto} \text{data} * l\Box\xrightarrow[\frac{s_o}{2}]{\pi} R\ \}$
    return d;
    $\{\ d \overset{\frac{s_o}{2}}{\mapsto} \text{data} * l\Box\xrightarrow[\frac{s_o}{2}]{\pi} R \wedge ret = d\ \}$
  }
  else {
    $\{\ init \mapsto 0 * \text{Hold } l, R, \bot * l\Box\xrightarrow[\bot]{\pi} R\ \}$
    InitializeData (d);
    $\{\ d \mapsto \text{data} * init \mapsto 0 * \text{Hold } l, R, \bot * l\Box\xrightarrow[\bot]{\pi} R\ \}$
    init = 1;
    $\{\ d \mapsto \text{data} * init \mapsto 1 * \text{Hold } l, R, \bot * l\Box\xrightarrow[\bot]{\pi} R\ \}$
    $\{\ d \overset{\frac{1}{2}}{\mapsto} \text{data} * \big(d \overset{\frac{1}{2}}{\mapsto} \text{data} * init \mapsto 1\big) * \text{Hold } l, R, \bot * l\Box\xrightarrow[\bot]{\pi} R\ \}$
    Release(l)
    $\{\ d \overset{\frac{1}{2}}{\mapsto} \text{data} * l\Box\xrightarrow[\bot]{\pi} R\ \}$
    return d;
    $\{\ d \overset{\frac{1}{2}}{\mapsto} \text{data} * l\Box\xrightarrow[\bot]{\pi} R \wedge ret = d\ \}$
  }
}
$\{\ \exists \pi_s, d \overset{\pi_s}{\mapsto} \text{data} * l\Box\xrightarrow[\bot]{\pi} R \wedge ret = d\ \}$
```

4 Stack Producer/consumer

```
$\{\ \mathrm{emp}\ \}$
void create();
$\{\ \mathrm{hd} \rightsquigarrow \epsilon\ \}$


$\{\ \mathrm{hd} \rightsquigarrow \epsilon\ \}$
void delete();
$\{\ \mathrm{emp}\ \}$

$\{\ \mathrm{hd} \rightsquigarrow ls\ \}$
void isemp();
$\{\ \mathrm{hd} \rightsquigarrow ls \wedge$
      $ls = \epsilon \wedge ret = \mathrm{true} \vee$
      $\exists x, l'. l = x :: l \wedge ret = \mathrm{false}\ \}$

$\{\ \mathrm{hd} \rightsquigarrow ls\ \}$
void enq(int x);
$\{\ \mathrm{hd} \rightsquigarrow x :: ls\ \}$

$\{\ \mathrm{hd} \rightsquigarrow x :: ls\ \}$
void deq();
$\{\ \mathrm{hd} \rightsquigarrow ls \wedge ret = x\ \}$

/* Producer */
$\{\ l\Box\xrightarrow[\bot]{\pi} R\ \}$    \\   $R = \lambda h.\ \mathrm{hd} \rightsquigarrow (\mathrm{listof}(h)) \wedge \mathrm{history\_stack}\ h$
void produce(x, l){
  $\{\ l\Box\xrightarrow[\bot]{\pi} R\ \}$
  Acquire(l);
  $\{\ \exists h_o,\ \mathrm{hd} \rightsquigarrow l \wedge \mathrm{history\_stack}\ h * \mathrm{Hold}\ l, R, h_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$    \\   $l = \mathrm{listof}(h_o)$
  enq(x);
  $\{\ \mathrm{hd} \rightsquigarrow x :: l \wedge \mathrm{history\_stack}\ h * \mathrm{Hold}\ l, R, h_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$
  $\{\ \big(\ \mathrm{hd} \rightsquigarrow (\mathrm{listof}(h_o + t \hookrightarrow (l, x :: l))) \wedge \mathrm{history\_stack}\ (h_o + t \hookrightarrow (l, x :: l))\big)$
     $* \mathrm{Hold}\ l, R, h_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$    \\   $t = \mathrm{last}\ h_o + 1$
  Release(l);
  $\{\ l\Box\xrightarrow[t\hookrightarrow(l,x::l)]{\pi} R\ \}$
} $\{\ l\Box\xrightarrow[t\hookrightarrow(l,x::l)]{\pi} R\ \}$

/* Consumer */
$\{\ l\Box\xrightarrow[\bot]{\pi} R\ \}$    \\   $R = \lambda h.\ \mathrm{hd} \rightsquigarrow (\mathrm{listof}(h)) \wedge \mathrm{history\_stack}\ h$
void consume(l){
  $\{\ l\Box\xrightarrow[\bot]{\pi} R\ \}$
  bool cont = true;
```

```
  $\{\ cont = true \wedge l\Box\xrightarrow[\bot]{\pi} R\ \}$
  while (cont) {
    Acquire(l);
    $\{\ cont = true \wedge \exists h_o,\ \text{hd} \rightsquigarrow l \wedge \text{history\_stack}\ h$
    $*\ \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$  $\backslash\backslash l = \text{listof}(h_o)$
    if (isemp() ) {
      Release(l);
      $\{\ cont = true \wedge l\Box\xrightarrow[\bot]{\pi} R\ \}$
    } else {
      $\{\ \exists x, l'.l = x :: l \wedge cont = true \wedge$
    $\text{hd} \rightsquigarrow l \wedge \text{history\_stack}\ h * \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$
      ret = deq();
      $\{\ ret = x \wedge cont = true \wedge$
    $\text{hd} \rightsquigarrow l' \wedge \text{history\_stack}\ h * \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$
      $\{\ ret = x \wedge cont = true \wedge$
    $\big(\ \text{hd} \rightsquigarrow (\text{listof}(h_o + t \hookrightarrow (l, l'))) \wedge \text{history\_stack}\ h\big)$
    $*\ \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$    $\backslash\backslash$  $t = \text{last}\ h_o + 1$
      Release(l);
      $\{\ ret = x \wedge cont = true \wedge l\Box\xrightarrow[t\hookrightarrow(l,l')]{\pi} R\ \}$
      cont = false;
      $\{\ ret = x \wedge cont = false \wedge l\Box\xrightarrow[t\hookrightarrow(l,l')]{\pi} R\ \}$
    }
    $\{\ cont = true \wedge l\Box\xrightarrow[\bot]{\pi} R \vee cont = false \wedge ret = x \wedge l\Box\xrightarrow[t\hookrightarrow(l,l')]{\pi} R\ \}$
  }
  $\{\ cont = false \wedge ret = x \wedge l\Box\xrightarrow[t\hookrightarrow(l,l')]{\pi} R\ \}$
  return ret;
}
$\{\ ret = x \wedge l\Box\xrightarrow[t\hookrightarrow(x::l',l')]{\pi} R\ \}$
```

```
/* Organizer */
$\{\ l\Box\xrightarrow[\bot]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$    \\   $R = \lambda h.\ \mathrm{hd} \rightsquigarrow (\mathrm{listof}(h)) \wedge \mathrm{history\_stack}\ h$
void organize1(l, a, b){
  $\{\ l\Box\xrightarrow[\bot]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
  (t1, v1) = consume(l);
  $\{\ (t1, v1) = x \wedge l\Box\xrightarrow[t1\hookrightarrow(x::l,l)]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
  if (t1) {
    $\{\ t1 \mapsto 1 * (t, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(x::l,l)]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
    a = v1;
    $\{\ t1 \mapsto 1 * (t, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(x::l,l)]{\pi} R * a \mapsto v1 * b \mapsto \_\ \}$
  } else {
    $\{\ t1 \mapsto 0 * (t, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(x::l,l)]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
    b = v1;
    $\{\ t1 \mapsto 0 * (t, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(x::l,l)]{\pi} R * a \mapsto \_ * b \mapsto v1\ \}$
  }
}
$\{\ (t1, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(x::l,l)]{\pi} R *$
    $t1 \mapsto 1 * a \mapsto v1 * b \mapsto \_ \vee$
    $t1 \mapsto 0 * a \mapsto \_ * b \mapsto v1\ \}$

$\{\ l\Box\xrightarrow[\bot]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
void organize2(l,a,b){
  $\{\ l\Box\xrightarrow[\bot]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
  organize1(l,a,b);
  $\{\ (t1, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(x::l,l)]{\pi} R *$
    $t1 \mapsto 1 * a \mapsto v1 * b \mapsto \_ \vee$
    $t1 \mapsto 0 * a \mapsto \_ * b \mapsto v1\ \}$
  organize1(l,a,b);
}
$\{\ (t2, v2) = x' \wedge (t1, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(x::l,l)*t'\hookrightarrow(x'::l',l')]{\pi} R *$
    $t1 \mapsto 1 * t2 \mapsto 1 * a \mapsto v2 * b \mapsto \_ \vee$
    $t1 \mapsto 1 * t2 \mapsto 0 * a \mapsto v1 * b \mapsto v2 \vee$
    $t1 \mapsto 0 * t2 \mapsto 1 * a \mapsto v2 * b \mapsto v1 \vee$
    $t1 \mapsto 0 * t2 \mapsto 0 * a \mapsto \_ * b \mapsto \_\ \}$
```

```
void main(){
  $\{ \ emp \ \}$
  l, x, y,z,a,b, = malloc (LOCK, LOCK, LOCK, LOCK, int, int, );
  hd = create();
  $\{ \ \mathrm{hd} \rightsquigarrow \epsilon * l \mapsto \_ * a \mapsto \_ * b \mapsto \_ \ \}$
  MakeLock(l); \\ $R = \lambda h.\ \mathrm{hd} \rightsquigarrow (\mathrm{listof}(h)) \wedge \text{history\_stack}\ h$
  $\{ \ \mathrm{hd} \rightsquigarrow \epsilon * l\square\xrightarrow[emp]{\top} R * \mathrm{Hold}\ l, R, emp * a \mapsto \_ * b \mapsto \_ \ \}$
  Release(l);
  $\{ \ l\square\xrightarrow[emp]{\top} R * a \mapsto \_ * b \mapsto \_ \ \}$
  x = Spawn(produce, ((0,0), l);
  $\{ \ x\bigcirc\!\!\!\rightarrow R_x * l\square\xrightarrow[emp]{\frac{2}{3}} R * a \mapsto \_ * b \mapsto \_ \ \}$ \\ $R_x = l\square\xrightarrow[t\hookrightarrow(l,(0,0))::l]{\frac{1}{3}} R$
  y = Spawn(produce, ((1,1), l);
  $\{ \ y\bigcirc\!\!\!\rightarrow R_y * x\bigcirc\!\!\!\rightarrow R_x * l\square\xrightarrow[emp]{\frac{1}{3}} R * a \mapsto \_ * b \mapsto \_ \ \}$ \\ $R_y = l\square\xrightarrow[t\hookrightarrow(l,(1,1)::l)]{\frac{1}{3}} R$
  z = Spawn(organize2, (l, a, b));
  $\{ \ z\bigcirc\!\!\!\rightarrow R_z * y\bigcirc\!\!\!\rightarrow R_y * x\bigcirc\!\!\!\rightarrow R_x \ \}$
  Join(x);
  $\{ \ h_1 = t_x \hookrightarrow (l_x, (0,0) :: l_x) \wedge z\bigcirc\!\!\!\rightarrow R_z * y\bigcirc\!\!\!\rightarrow R_y * l\square\xrightarrow[h_1]{\frac{1}{3}} R \ \}$
  Join(y);
  $\{ \ h_2 = t_x \hookrightarrow (l_x, (0,0) :: l_x) * t_y \hookrightarrow (l_y, (1,1) :: l_y) \wedge z\bigcirc\!\!\!\rightarrow R_z * l\square\xrightarrow[h_2]{\frac{2}{3}} R \ \}$
  Join(z);
  $\{ \ h_3 = h_2 * t_z1 \hookrightarrow (x :: l_z1, l_z1) * t_z2 \hookrightarrow (y :: l_z2, l_z2)) \wedge (k1, v1) = x \wedge (k2, v2) = y \wedge$
    $l\square\xrightarrow[h_3]{\top} R *$
    $k1 \mapsto 1 * k2 \mapsto 1 * a \mapsto v2 * b \mapsto \_ \vee$
    $k1 \mapsto 1 * k2 \mapsto 0 * a \mapsto v1 * b \mapsto v2 \vee$
    $k1 \mapsto 0 * k2 \mapsto 1 * a \mapsto v2 * b \mapsto v1 \vee$
    $k1 \mapsto 0 * k2 \mapsto 0 * a \mapsto \_ * b \mapsto \_ \ \}$
  Acquire(l);
  $\{ \ (k1, v1) = x \wedge (k2, v2) = y \wedge$
    $l\square\xrightarrow[h_3]{\top} R * \mathrm{Hold}\ l, R, h_o * \ \mathrm{hd} \rightsquigarrow (\mathrm{listof}(h_3)) \wedge \text{history\_stack}\ (h_3) *$
    $k1 \mapsto 1 * k2 \mapsto 1 * a \mapsto v2 * b \mapsto \_ \vee$
    $k1 \mapsto 1 * k2 \mapsto 0 * a \mapsto v1 * b \mapsto v2 \vee$
    $k1 \mapsto 0 * k2 \mapsto 1 * a \mapsto v2 * b \mapsto v1 \vee$
    $k1 \mapsto 0 * k2 \mapsto 0 * a \mapsto \_ * b \mapsto \_ \ \}$ \\ Case analysis on h3
  $\{ \ l\square\xrightarrow[h_3]{\top} R * \mathrm{Hold}\ l, R, h_o * \ \mathrm{hd} \rightsquigarrow (\epsilon) *$
    $(1,1) = x \wedge (0,0) = y \wedge k1 \mapsto 1 * k2 \mapsto 0 * a \mapsto 1 * b \mapsto 0 \vee$
    $(0,0) = x \wedge (1,1) = y \wedge k1 \mapsto 0 * k2 \mapsto 1 * a \mapsto 1 * b \mapsto 0 \ \}$
  Free(l); free(k1,k2);
  $\{ \ \mathrm{hd} \rightsquigarrow \epsilon * a \mapsto 1 * b \mapsto 0 \ \}$
  delete();
}
$\{ \ a \mapsto 1 * b \mapsto 0 \ \}$
```

5 Queue Producer/consumer

```
struct elem {
  struct elem *next;
  struct elem *data;
};

struct fifo {
  struct elem *hd;
  struct elem *tl;
};

$\{ \text{emp} \}$
fifo *create(){
  Q = malloc(sizeof(fifo));
  $\{ Q.hd \mapsto \_ * Q.hd \mapsto \_ \}$
  hd, tl = NULL;
  $\{ Q \leadsto \epsilon \}$
  return Q;
}
$\{ \text{Q} \leadsto \epsilon \}$


$\{ \text{Q} \leadsto \epsilon \}$
void delete(Q){
  free(Q);
}
$\{ \text{emp} \}$

$\{ \text{Q} \leadsto ls \}$
void isemp(){
  return (Q.hd == NULL)
  $\{ \text{Q} \leadsto ls \land ret = (Q.hd == NULL) \}$
}
$\{ \text{Q} \leadsto ls \land$
    $ls = \epsilon \land ret = \text{true} \lor$
    $\exists x, l'.l = l :: x \land ret = \text{false} \}$

$\{ \text{Q} \leadsto ls \}$
void enq(fifo Q, type x){
  $\{ \text{Q} \leadsto ls \}$
  if (hd==NULL) {
    $\{ \text{Q} \leadsto \epsilon \land hd = NULL \}$
    Q->hd=(NULL, x);
    Q->tl=(NULL, x);
    $\{ \text{Q} \leadsto x :: \epsilon \}$
  }
    else {
    $\{ \text{Q} \leadsto [x_1, \ldots, x_n] \land hd \neq \text{NULL} \}$
```

```
        tl→next = (NULL, x);
        $\{\ Q.hd \mapsto (x_1, a_1) * \ldots a_{n-1} \mapsto tl = (a_n, x_n) * a_n \mapsto (\text{NULL}\ , x)\ \}$
       Q→tl=(\NULL, x);
        $\{\ Q.hd \mapsto (x_1, a_1) * \ldots a_{n-1} \mapsto (a_n, x_n) * a_n \mapsto tl = (\text{NULL}\ , x)\ \}$
        $\{\ \text{Q} \rightsquigarrow [x, x_n, \ldots, x_1]\ \}$
    }
  }
$\{\ \text{Q} \rightsquigarrow x :: ls\ \}$

$\{\ \text{Q} \rightsquigarrow ls :: x\ \}$
void deq(fifo Q){
   h=Q→hd→data;
   $\{\ h = x \wedge \text{Q} \rightsquigarrow ls :: x\ \}$
   n=Q→hd→next;
   $\{\ h = x \wedge n = a_1 \wedge Q.hd \mapsto (x, a_1) * a_1 \mapsto (x_2, a_2) * \ldots a_{n-1} \mapsto tl = (\text{NULL}\ , x_n)\ \}$
   Q→head=n;
   $\{\ h = x \wedge n = a_1 \wedge Q.hd \mapsto (x_2, a_2) * \ldots a_{n-1} \mapsto tl = (\text{NULL}\ , x_n)\ \}$
   return h;
}$\{\ \text{Q} \rightsquigarrow ls \wedge ret = x\ \}$

/* Producer */
$\{\ l\Box\xrightarrow[\bot]{\pi} R\ \}$    \\   $R = \lambda h.\ \text{Q} \rightsquigarrow (\text{listof}(h)) \wedge \text{history\_queue}\ h$
void produce(fifo Q, type x, lock l){
   $\{\ l\Box\xrightarrow[\bot]{\pi} R\ \}$
   Acquire(l);
   $\{\ \exists h_o,\ \text{Q} \rightsquigarrow l \wedge \text{history\_queue}\ h * \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$    \\   $l = \text{listof}(h_o)$
   enq(x);
   $\{\ \ \text{Q} \rightsquigarrow x :: l \wedge \text{history\_queue}\ h_o * \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$
   $\{\ (\ \text{Q} \rightsquigarrow (\text{listof}(h_o \oplus t \hookrightarrow (l, x :: l))) \wedge \text{history\_queue}\ (h_o \oplus t \hookrightarrow (l, x :: l)))$
      $* \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$    \\   $t = \text{last}\ h_o + 1$
   Release(l);
   $\{\ l\Box\xrightarrow[t \hookrightarrow (l, x :: l)]{\pi} R\ \}$
} $\{\ l\Box\xrightarrow[t \hookrightarrow (l, x :: l)]{\pi} R\ \}$

/* Consumer */
$\{\ l\Box\xrightarrow[\epsilon]{\pi} R\ \}$    \\   $R = \lambda h.\ \text{Q} \rightsquigarrow (\text{listof}(h_o)) \wedge \text{history\_queue}\ h$
void consume(l){
   $\{\ l\Box\xrightarrow[\epsilon]{\pi} R\ \}$
   bool cont = true;
   $\{\ cont = true \wedge l\Box\xrightarrow[\epsilon]{\pi} R\ \}$
   while (cont) {
      Acquire(l);
```

```
    $\{\ cont = true \wedge \exists h_o,\ \mathrm{Q} \rightsquigarrow l \wedge \text{history\_queue}\ h_o$
    $*\ \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\epsilon]{\pi} R\ \}$  \\$l = \text{listof}(h_o)$
    if (isemp() ) {
       Release(l);
       $\{\ cont = true \wedge l\Box\xrightarrow[\epsilon]{\pi} R\ \}$
    } else {
       $\{\ \exists x, l'.l = l :: x \wedge cont = true \wedge$
     $\mathrm{Q} \rightsquigarrow l \wedge \text{history\_queue}\ h_o * \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\epsilon]{\pi} R\ \}$
       ret = deq();
       $\{\ ret = x \wedge cont = true \wedge$
     $\mathrm{Q} \rightsquigarrow l' \wedge \text{history\_queue}\ h_o * \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\epsilon]{\pi} R\ \}$
       $\{\ ret = x \wedge cont = true \wedge$
    $(\ \mathrm{Q} \rightsquigarrow (\text{listof}(h_o \oplus t \hookrightarrow (l, l'))) \wedge \text{history\_queue}\ h_o)$
    $*\ \text{Hold}\ l, R, h_o * l\Box\xrightarrow[\epsilon]{\pi} R\ \}$     \\   $t = \text{last}\ h_o + 1$
       Release(l);
       $\{\ ret = x \wedge cont = true \wedge l\Box\xrightarrow[t\hookrightarrow(l,l')]{\pi} R\ \}$
       cont = false;
       $\{\ ret = x \wedge cont = false \wedge l\Box\xrightarrow[t\hookrightarrow(l,l')]{\pi} R\ \}$
    }
    $\{\ cont = true \wedge l\Box\xrightarrow[\epsilon]{\pi} R \vee cont = false \wedge ret = x \wedge l\Box\xrightarrow[t\hookrightarrow(l,l')]{\pi} R\ \}$
  }
  $\{\ cont = false \wedge ret = x \wedge l\Box\xrightarrow[t\hookrightarrow(l,l')]{\pi} R\ \}$
  return ret;
}
$\{\ ret = x \wedge l\Box\xrightarrow[t\hookrightarrow(l'::x,l')]{\pi} R\ \}$
```

```
/* Organizer */
$\{\ l\Box\xrightarrow[\epsilon]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$    \\   $R = \lambda h.\ \mathrm{Q} \rightsquigarrow (\mathrm{listof}(h)) \wedge \mathrm{history\_queue}\ h_o$
void organize1(l, a, b){
  $\{\ l\Box\xrightarrow[\epsilon]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
  (t1, v1) = consume(l);
  $\{\ (t1, v1) = x \wedge l\Box\xrightarrow[t1\hookrightarrow(l::x,l)]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
  if (t1) {
    $\{\ t1 \mapsto 1 * (t, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(l::x,l)]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
    a = v1;
    $\{\ t1 \mapsto 1 * (t, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(l::x,l)]{\pi} R * a \mapsto v1 * b \mapsto \_\ \}$
  } else {
    $\{\ t1 \mapsto 0 * (t, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(l::x,l)]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
    b = v1;
    $\{\ t1 \mapsto 0 * (t, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(l::x,l)]{\pi} R * a \mapsto \_ * b \mapsto v1\ \}$
  }
}
$\{\ (t1, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(l::x,l)]{\pi} R *$
    $t1 \mapsto 1 * a \mapsto v1 * b \mapsto \_ \vee$
    $t1 \mapsto 0 * a \mapsto \_ * b \mapsto v1\ \}$

$\{\ l\Box\xrightarrow[\epsilon]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
void organize2(l,a,b){
  $\{\ l\Box\xrightarrow[\epsilon]{\pi} R * a \mapsto \_ * b \mapsto \_\ \}$
  organize1(l,a,b);
  $\{\ (t1, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(l::x,l)]{\pi} R *$
    $t1 \mapsto 1 * a \mapsto v1 * b \mapsto \_ \vee$
    $t1 \mapsto 0 * a \mapsto \_ * b \mapsto v1\ \}$
  organize1(l,a,b);
}
 $\{\ R_z = (t2, v2) = x' \wedge (t1, v1) = x \wedge l\Box\xrightarrow[t\hookrightarrow(l::x,l)\oplus t'\hookrightarrow(l'::x',l')]{\pi} R *$
    $t1 \mapsto 1 * t2 \mapsto 1 * a \mapsto v2 * b \mapsto \_ \vee$
    $t1 \mapsto 1 * t2 \mapsto 0 * a \mapsto v1 * b \mapsto v2 \vee$
    $t1 \mapsto 0 * t2 \mapsto 1 * a \mapsto v2 * b \mapsto v1 \vee$
    $t1 \mapsto 0 * t2 \mapsto 0 * a \mapsto \_ * b \mapsto \_\ \}$
```

```
void main(){
  ${\ emp\ }$
  l, x, y,z,a,b, = malloc (LOCK, LOCK, LOCK, LOCK, int, int, );
  hd = create();
   ${\ \ \mathrm{Q} \leadsto \epsilon * l \mapsto \_ * a \mapsto \_ * b \mapsto \_\ }$
  MakeLock(l);  \\   $R = \lambda h.\ \mathrm{Q} \leadsto (\mathrm{listof}(h)) \wedge \mathrm{history\_queue}\ h$
  ${\ \ \mathrm{Q} \leadsto \epsilon * l\Box\xrightarrow[emp]{\top} R * \mathrm{Hold}\ l, R, emp * a \mapsto \_ * b \mapsto \_\ }$
  Release(l);
  ${\ l\Box\xrightarrow[emp]{\top} R * a \mapsto \_ * b \mapsto \_\ }$
  x = Spawn(produce, ((0,0), l);
  ${\ x\bigcirc\!\!\!\rightarrow R_x * l\Box\xrightarrow[emp]{\frac{2}{3}} R * a \mapsto \_ * b \mapsto \_\ }$   \\  $R_x = l\Box\xrightarrow[t\hookrightarrow(l,(0,0))::l]{\frac{1}{3}} R$
  y = Spawn(produce, ((1,1), l);
  ${\ y\bigcirc\!\!\!\rightarrow R_y * x\bigcirc\!\!\!\rightarrow R_x * l\Box\xrightarrow[emp]{\frac{1}{3}} R * a \mapsto \_ * b \mapsto \_\ }$   \\  $R_y = l\Box\xrightarrow[t\hookrightarrow(l,(1,1)::l)]{\frac{1}{3}} R$
  z = Spawn(organize2, (l, a, b));
  ${\ z\bigcirc\!\!\!\rightarrow R_z * y\bigcirc\!\!\!\rightarrow R_y * x\bigcirc\!\!\!\rightarrow R_x\ }$
  Join(x);
  ${\ h_1 = t_x \hookrightarrow (l_x, (0,0) :: l_x) \wedge z\bigcirc\!\!\!\rightarrow R_z * y\bigcirc\!\!\!\rightarrow R_y * l\Box\xrightarrow[h_1]{\frac{1}{3}} R\ }$
  Join(y);
  ${\ h_2 = t_x \hookrightarrow (l_x, (0,0) :: l_x) \oplus t_y \hookrightarrow (l_y, (1,1) :: l_y) \wedge z\bigcirc\!\!\!\rightarrow R_z * l\Box\xrightarrow[h_2]{\frac{2}{3}} R\ }$
  Join(z);
  ${\ h_3 = h_2 \oplus t_z1 \hookrightarrow (l_z1 :: x, l_z1) \oplus t_z2 \hookrightarrow (l_z2 :: x', l_z2)) \wedge (k1, v1) = x \wedge (k2, v2) = x' \wedge$
    $l\Box\xrightarrow[h_3]{\top} R *$
    $k1 \mapsto 1 * k2 \mapsto 1 * a \mapsto v2 * b \mapsto \_ \vee$
    $k1 \mapsto 1 * k2 \mapsto 0 * a \mapsto v1 * b \mapsto v2 \vee$
    $k1 \mapsto 0 * k2 \mapsto 1 * a \mapsto v2 * b \mapsto v1 \vee$
    $k1 \mapsto 0 * k2 \mapsto 0 * a \mapsto \_ * b \mapsto \_\ }$
  Acquire(l);
  ${\ (k1, v1) = x \wedge (k2, v2) = y \wedge$
    $l\Box\xrightarrow[h_3]{\top} R * \mathrm{Hold}\ l, R, h_o *\ \ \mathrm{Q} \leadsto (\mathrm{listof}(h_3)) \wedge \mathrm{history\_queue}\ (h_3) *$
    $k1 \mapsto 1 * k2 \mapsto 1 * a \mapsto v2 * b \mapsto \_ \vee$
    $k1 \mapsto 1 * k2 \mapsto 0 * a \mapsto v1 * b \mapsto v2 \vee$
    $k1 \mapsto 0 * k2 \mapsto 1 * a \mapsto v2 * b \mapsto v1 \vee$
    $k1 \mapsto 0 * k2 \mapsto 0 * a \mapsto \_ * b \mapsto \_\ }$      \\  Case analysis on h3
  ${\ l\Box\xrightarrow[h_3]{\top} R * \mathrm{Hold}\ l, R, h_o *\ \ \mathrm{Q} \leadsto (\epsilon) *$
    $(1,1) = x \wedge (0,0) = y \wedge k1 \mapsto 1 * k2 \mapsto 0 * a \mapsto 1 * b \mapsto 0 \vee$
    $(0,0) = x \wedge (1,1) = y \wedge k1 \mapsto 0 * k2 \mapsto 1 * a \mapsto 1 * b \mapsto 0\ }$
  Free(l);  free(k1,k2);
  ${\ \ \mathrm{Q} \leadsto \epsilon * a \mapsto 1 * b \mapsto 0\ }$
  delete();
}
${\ a \mapsto 1 * b \mapsto 0\ }$
```

6 Tree add

```
struct node
{
  int k;          //key_value
  struct node *l; //left subtree
  struct node *r; //right subtree
};

void AddTree(struct node * t, int *res){
  $\{\ t \rightarrowtail tree * res \mapsto \_\ \}$
  if (empty(t)){
    $\{\ t \rightarrowtail \epsilon * res \mapsto \_\ \}$
    res = 0;
    $\{\ t \rightarrowtail \epsilon * res \mapsto 0\ \}$
  } else {
    $\{\ t \rightarrowtail (k, ltree, rtree) * res \mapsto \_\ \}$
    int *lres, rres;
    thread lth, rth;
    $\{\ t \rightarrowtail (k, ltree, rtree) * (res, lres, rres, lth, rth) \mapsto \_\ \}$
    $\{\ t \mapsto (k, l, r) * l \rightarrowtail ltree * r \rightarrowtail rtree * (res, lres, rres, lth, rth) \mapsto \_\ \}$
    lthread = spawn (AddTree, (left, t→l, lies));
    $\{\ lth \bigcirc\!\!\!\rightarrow R_l * t \mapsto (k, l, r) * r \rightarrowtail rtree * (res, rres, rth) \mapsto \_\ \}$
    rthread = spawn (AddTree, (right, t→r));
    $\{\ rth \bigcirc\!\!\!\rightarrow R_r * lth \bigcirc\!\!\!\rightarrow R_l * t \mapsto (k, l, r) * res \mapsto \_\ \}$
    join (lth);
    $\{\ (\mathrm{add\_tree}(ltree) = k_l) \wedge l \rightarrowtail ltree * lres \mapsto k_l) *$
    $rth \bigcirc\!\!\!\rightarrow R_r * t \mapsto (k, l, r) * res \mapsto \_\ \}$
    join (rth);
    $\{\ (\mathrm{add\_tree}(ltree) = k_l) \wedge l \rightarrowtail ltree * lres \mapsto k_l) *$
    $(\mathrm{add\_tree}(rtree) = k_r) \wedge r \rightarrowtail rtree * rres \mapsto k_r) *$
    $t \mapsto (k, l, r) * (res) \mapsto \_\ \}$
    res = lres + rres + t.k;
    $\{\ (\mathrm{add\_tree}(ltree) = k_l) \wedge lres \mapsto k_l) *$
    $(\mathrm{add\_tree}(rtree) = k_r) \wedge rres \mapsto k_r) *$
    $t \rightarrowtail (k, ltree, rtree) * (res) \mapsto (k_l + k_r + k)\ \}$
  }
  $\{\ (\mathrm{add\_tree}(tree)) = k' \wedge t \rightarrowtail tree * (res) \mapsto (k')\ \}$
}
```

7 Tree add with reporting

```
struct node
{
  lock l;
  int *k;          //sum_value
  int k;           //key_value
  struct node *l; //left subtree
  struct node *r; //right subtree
};


$\{\ node.l\square\!\xrightarrow[\bot]{\pi} R\ \}$    //  $R = \lambda v.STUFF$
void AddTreeRep(struct node * t, int *RL){
  $\{\ t \rightarrowtail tree * RL\square\!\xrightarrow[0]{\pi} R\ \}$
  if (empty(t)){
    \\This branch is useless in practice.
    $\{\ \mathrm{add\_tree}(\epsilon) = 0 \wedge t \rightarrowtail \epsilon * RL\square\!\xrightarrow[0]{\pi} R\ \}$
  } else {
    $\{\ t \rightarrowtail (k, ltree, rtree) * RL\square\!\xrightarrow[0]{\pi} R\ \}$
    $\{\ t \mapsto (k, l, r) * l \rightarrowtail ltree * r \rightarrowtail rtree * RL\square\!\xrightarrow[0]{\pi} R\ \}$
    lthread = spawn (AddTreeRep, (left , t->l, lies));
    $\{\ lth\bigcirc\!\!\rightarrow R_l * t \mapsto (k, l, r) * r \rightarrowtail rtree * RL\square\!\xrightarrow[0]{\frac{2\pi}{3}} R\ \}$
    rthread = spawn (AddTreeRep, (right , t->r));
    $\{\ rth\bigcirc\!\!\rightarrow R_r * lth\bigcirc\!\!\rightarrow R_l * t \mapsto (k, l, r) * RL\square\!\xrightarrow[0]{\frac{\pi}{3}} R\ \}$
    Acquire(RL);
    $\{\ \exists v_o.result \mapsto (v_o + 0) * \mathrm{Hold}\ RL, R, v_o *$
    $rth\bigcirc\!\!\rightarrow R_r * lth\bigcirc\!\!\rightarrow R_l * t \mapsto (k, l, r) * RL\square\!\xrightarrow[0]{\frac{\pi}{3}} R\ \}$
    result = result + (t->k);
    $\{\ \exists v_o.result \mapsto (v_o + k) * \mathrm{Hold}\ RL, R, v_o *$
    $rth\bigcirc\!\!\rightarrow R_r * lth\bigcirc\!\!\rightarrow R_l * t \mapsto (k, l, r) * RL\square\!\xrightarrow[0]{\frac{\pi}{3}} R\ \}$
    Release(RL);
    $\{\ rth\bigcirc\!\!\rightarrow R_r * lth\bigcirc\!\!\rightarrow R_l * t \mapsto (k, l, r) * RL\square\!\xrightarrow[k]{\frac{\pi}{3}} R\ \}$
    join (lth);
    $\{\ (\mathrm{add\_tree}(ltree) = k_l \wedge l \rightarrowtail ltree * RL\square\!\xrightarrow[k_l]{\frac{\pi}{3}} R) *$
    $rth\bigcirc\!\!\rightarrow R_r * t \mapsto (k, l, r) * RL\square\!\xrightarrow[k]{\frac{\pi}{3}} R\ \}$
    join (rth);
```

$\{\ (\text{add\_tree}(rtree) = k_r \wedge r \rightarrowtail rtree * RL\Box\xrightarrow[k_r]{\frac{\pi}{3}} R) *$

$(\text{add\_tree}(ltree) = k_l \wedge l \rightarrowtail ltree * RL\Box\xrightarrow[k_l]{\frac{\pi}{3}} R) *$

$t \mapsto (k, l, r) * RL\Box\xrightarrow[k]{\frac{\pi}{3}} R\ \}$

$\{\ \text{add\_tree}(rtree) = k_r \wedge \text{add\_tree}(ltree) = k_l \wedge$

$r \rightarrowtail rtree * l \rightarrowtail ltree * t \mapsto (k, l, r) * RL\Box\xrightarrow[k+k_l+k_r]{\pi} R\ \}$

}

$\{\ \text{add\_tree}(tree) = k' \wedge t \rightarrowtail tree * RL\Box\xrightarrow[k']{\pi} R\ \}$

}

$\{\ \text{add\_tree}(tree) = k' \wedge t \rightarrowtail tree * RL\Box\xrightarrow[k']{\pi} R\ \}$

8 Adding a Directed Acyclic Graph with repetitions

$$
\begin{aligned}
\text{dag} :=& \epsilon \\
& |\forall sum, k, (l, r : \text{dag})(\pi_l, \pi_r : \text{shares})(sum, k, \pi_l, l, \pi_r, r)
\end{aligned}
$$

$$
\begin{array}{lcl}
g \overset{\pi}{\underset{\perp}{\dagger}} \epsilon & \triangleq & g = \text{NULL} \\
g \overset{\pi}{\underset{0}{\dagger}} \epsilon & \triangleq & g = \text{NULL} \\
g \overset{\pi}{\underset{sum}{\dagger}} (sum, k, \pi_l, d_l, \pi_r, d_r) & \triangleq & g.lock \square \xrightarrow[(sum,k,\pi_l,l,\pi_r,r)]{\pi} R \\
 & \text{WHERE} & \\
 & & \\
R & \triangleq & \lambda(sum, k, \pi_l, d_l, \pi_r, d_r).\exists l, r, k, sum_l, sum_r \\
 & & g.k \rightarrow k* \\
 & & g.l \rightarrow l* \\
 & & g.r \rightarrow r* \\
 & & \text{if } g.sum = \text{NULL} \ \text{ then} \\
 & & \quad sum = sum_l = sum_r = \perp \\
 & & \quad g.sum = \text{NULL} * \\
 & & \quad l \overset{\pi_l}{\underset{\perp}{\dagger}} d_l * r \overset{\pi_r}{\underset{\perp}{\dagger}} d_r \\
 & & \text{else} \\
 & & \quad sum = k + sum_l + sum_r \wedge \\
 & & \quad g.sum \rightarrow sum* \\
 & & \quad l \overset{\pi_l}{\underset{sum_l}{\dagger}} d_l * r \overset{\pi_r}{\underset{sum_r}{\dagger}} d_r
\end{array}
$$

$$
\begin{gathered}
\epsilon \oplus \epsilon \triangleq \epsilon \\
(sum_1, k, \pi_l, l_1, \pi_r, r_1) \oplus (sum_2, k, \pi_l, l_2, \pi_r, r_2) \triangleq (sum_1 \oplus sum_2, k, \pi_l, l_1 \oplus l_2, \pi_r, r_1 \oplus r_2)
\end{gathered}
$$

```
struct node
{
  lock l;           //lock
  int * sum;          //Partial sum
  int k;            //key_value
  struct node *l; //left subtree
  struct node *r; //right subtree
};
```

\

```
$\{\ g \overset{\pi}{\underset{\perp}{\dagger}} d\ \}$
void AddDag(struct node * g, int *ret){
  if (g = NULL){
    $\{\ g \overset{\pi}{\underset{\perp}{\dagger}} \epsilon * ret \mapsto \_\ \}$
    ret = 0;
    $\{\ g \overset{\pi}{\underset{0}{\dagger}} \epsilon * ret \mapsto 0\ \}$
    return;
  } else {
    $\{\ g \overset{\pi}{\underset{\perp}{\dagger}} (\perp, k, \pi_l, l_s, \pi_r, r_s) * ret \mapsto \_\ \}$
    Acquire(g);
    $\{\ \exists d_o, R(\perp, k, \pi_l, d_l, \pi_r, d_r) \oplus d_o * \text{Hold } g, R, d_o * g \overset{\pi}{\underset{\perp}{\dagger}} (\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto\ \}$
    $\{\ \exists v_o, d_l, d_r, R(\perp \oplus v_o, k, \pi_l, l_s \oplus d_l, \pi_r, r_s \oplus d_r) *$
    $\text{Hold } g, R, d_o * g \overset{\pi}{\underset{\perp}{\dagger}} (\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto \_\ \}$
    if (g.sum != NULL) {
      $\{\ R(v_o, k, \pi_l, d_l, \pi_r, d_r) * \text{Hold } g, R, d_o * g \overset{\pi}{\underset{\perp}{\dagger}} (\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto \_\ \}$
      ret = g.sum;
      $\{\ R(v_o, k, \pi_l, d_l, \pi_r, d_r) * \text{Hold } g, R, d_o * g \overset{\pi}{\underset{\perp}{\dagger}} (\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto g.sum\ \}$
      Release (g)
      $\{\ g \overset{\pi}{\underset{v_o}{\dagger}} (v_o, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto g.sum\ \}$
    } else {
      $\{\ R(\perp, k, \pi_l, d_l, \pi_r, d_r) * \text{Hold } g, R, d_o * g \overset{\pi}{\underset{\perp}{\dagger}} (\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto \_\ \}$
      $\{\ \exists l, r, k, sum_l, sum_r, g.k \mapsto k * g.l \mapsto l * g.r \mapsto r * g.sum = \text{NULL}\ *$
        $sum = sum_l = sum_r = \perp *$
        $l \overset{\pi_l}{\underset{\perp}{\dagger}} d_l * r \overset{\pi_r}{\underset{\perp}{\dagger}} d_r *$
        $\text{Hold } g, R, d_o * g \overset{\pi}{\underset{\perp}{\dagger}} (\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto \_\ \}$
      int *lret , rret;
      thr = Spawn (AddDag, (g.r, rret));
      $\{\ thr \bigcirc\!\!\!\rightarrow R_r * g.k \mapsto k * g.l \mapsto l * g.r \mapsto r * g.sum = \text{NULL}\ *$
        $sum = sum_l = \perp * l \overset{\pi_l}{\underset{\perp}{\dagger}} d_l *$
        $\text{Hold } g, R, d_o * g \overset{\pi}{\underset{\perp}{\dagger}} (\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto \_\ \}$
      AddDag (g.l, lret);
      $\{\ thr \bigcirc\!\!\!\rightarrow R_r * g.k \mapsto k * g.l \mapsto l * g.r \mapsto r * g.sum = \text{NULL}\ *$
        $l \overset{\pi_l}{\underset{sum_l}{\dagger}} d_l * lret \mapsto sum_l *$
```

```
        $\text{Hold } g, R, d_o * g\overset{\pi}{\underset{\perp}{\dagger}}(\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto \_ \}$
      Join(the);
      $\{ \ g.k \mapsto k * g.l \mapsto l * g.r \mapsto r * g.sum = \text{NULL} \ *$
        $l \overset{\pi_l}{\underset{sum_l}{\dagger}} d_l * lret \mapsto sum_l *$
        $l \overset{\pi_r}{\underset{sum_r}{\dagger}} d_r * rret \mapsto sum_r *$
        $\text{Hold } g, R, d_o * g\overset{\pi}{\underset{\perp}{\dagger}}(\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto \_ \}$
      ret = (g.sum = k + sum_l + sum_r);
      $\{ \ g.k \mapsto k * g.l \mapsto l * g.r \mapsto r *$
        $g.sum \mapsto k + sum_l + sum_r *$
        $l \overset{\pi_l}{\underset{sum_l}{\dagger}} d_l * lret \mapsto sum_l *$
        $l \overset{\pi_r}{\underset{sum_r}{\dagger}} d_r * rret \mapsto sum_r *$
        $\text{Hold } g, R, d_o * g\overset{\pi}{\underset{\perp}{\dagger}}(\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto k + sum_l + sum_r \ \}$
      $\{ \ R(k + sum_l + sum_r, k, \pi_l, d_l, \pi_r, d_r) *$
        $\text{Hold } g, R, d_o * g\overset{\pi}{\underset{\perp}{\dagger}}(\perp, k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto sum' \ \}$    \\ $_{sum'=k+sum_l+sum_r}$
      Release (g);
      $\{ \ g \overset{\pi}{\underset{sum'}{\dagger}} (sum', k, \pi_l, d_l, \pi_r, d_r) * ret \mapsto sum' \ \}$
    }
  }
}
$\{ \ \exists sum', d, g \overset{\pi}{\underset{sum'}{\dagger}} d * ret \mapsto sum' \ \}$
```