1 General Info

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

$a \rightsquigarrow l \triangleq a \mapsto (x_1, a_1) * a_1 \mapsto (x_2, a_2) * \dots a_{n-1} \mapsto (x_n, a_n)$ Where $l = x_1, x_2, \dots x_n$

$l \bigcirc\!\!\rightarrow R \triangleq l \square\!\!\underset{0}{\rightarrow} R$ 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

$$Hist \triangleq \mathbb{N} \rightharpoonup \text{list } * \text{list}$$

$$t \hookrightarrow_h (l_1, l_2) \triangleq h[t] = (l_1, l_2)$$

- 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 $h \triangleq \forall t \in h \rightarrow \forall t' < t, t' \in h$
- 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 $v \wedge$ stacklike 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);

$\{\ l\square\!\xrightarrow[0]{1} R * \mathsf{array}(a, [\_, \_, \ldots, \_]_n\} \quad //R = \lambda v.c \mapsto v$

$\{\ \bigstar_{j=0}^{n-1} l\square\!\xrightarrow[0]{\frac{1}{n}} R * \mathsf{array}(a, [\_, \_, \ldots, \_]_n\}$

for (i = 0; i < n; i ++) {

$\{\ \bigstar_{j=0}^{i} l_j \bigcirc\!\!\!\rightarrow R_j * \bigstar_{j=i+1}^{n-1} l\square\!\xrightarrow[0]{\frac{1}{n}} R * \mathsf{array}(a, [l_1, \ldots, l_i, \_, \ldots \_]_n\}$

a[i] = Spawn(incr, (l,c));

$\{\ \bigstar_{j=0}^{i+1} l_j \bigcirc\!\!\!\rightarrow R_j * \bigstar_{j=i+2}^{n-1} l\square\!\xrightarrow[0]{\frac{1}{n}} R * \mathsf{array}(a, [l_1, \ldots, l_i, l_{i+1}, \_, \ldots \_]_n\}$

}

$\{\ \bigstar_{j=0}^{n} l_j \bigcirc\!\!\!\rightarrow R_j * \mathsf{array}(a, [l_1, \ldots, l_n]\}$

for (i = 0; i < n; i ++)

{

$\{\ \bigstar_{j=0}^{i} R_j * \bigstar_{j=i}^{n} l_j \bigcirc\!\!\!\rightarrow R_j * \mathsf{array}(a, [l_1, \ldots, l_n]\}$

Join(a[i]);

$\{\ \bigstar_{j=0}^{i+1} R_j * \bigstar_{j=i+1}^{n} l_j \bigcirc\!\!\!\rightarrow R_j * \mathsf{array}(a, [l_1, \ldots, l_n]\}$

}

$\{\ \bigstar_{j=0}^{n} R_j * \mathsf{array}(a, [l_1, \ldots, l_n]\} \quad //R_j = l\square\!\xrightarrow[1]{\frac{1}{n}} R$

$\{\ l\square\!\xrightarrow[n]{} R * \mathsf{array}(a, [l_1, \ldots, l_n]\}$

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);

$\{\ \mathsf{ret} \mapsto n\}$

return ret }

```
void incr(l,c) {
```

$\{\ l\square\xrightarrow[0]{\frac{1}{n}} R\ \}$

```
  Acquire(l);
```

$\{\ \exists v_o, c \mapsto v_o * \mathrm{Hold}\ l, R, v_o * l\square\xrightarrow[0]{\frac{1}{n}} R\}$

```
  ( *c)++;
```

$\{\ \exists v_o, c \mapsto (v_o + 1) * \mathrm{Hold}\ l, R, v_o * l\square\xrightarrow[0]{\frac{1}{n}} R\}$

```
  Release(l);
```

$\{\ l\square\xrightarrow[1]{\frac{1}{n}} R\}$

```
}
```

3 Single Initialize / concurrent read

$\{\ l\square\xrightarrow[\bot]{\pi} R\ \}$ \\\\ $R = \lambda v.init \mapsto 0 \wedge v = \bot \vee init \mapsto 1 * d \overset{\top - v}{\mapsto} \text{data} \wedge [\top > v]$

```
data *first_access(l) {
```

$\{\ l\square\xrightarrow[\bot]{\pi} R\ \}$

```
  Acquire(l);
```

$\{\ \exists v_o, init \mapsto 0 \wedge v_o = \bot \vee init \mapsto 1 * d \overset{s_o}{\mapsto} \text{data} * \text{Hold } l, R, v_o * l\square\xrightarrow[\bot]{\pi} R\ \}$

\\\\ where $s_o = \top - v_o$

```
  if(init) {
```

$\{\ init \mapsto 1 * d \overset{s_o}{\mapsto} \text{data} * \text{Hold } l, R, v_o * l\square\xrightarrow[\bot]{\pi} R\ \}$

$\{\ d \overset{\frac{s_o}{2}}{\mapsto} \text{data} * \big(init \mapsto 1 * d \overset{\top - (v_o + \frac{s_o}{2})}{\mapsto} \text{data}\big) * \text{Hold } l, R, v_o * l\square\xrightarrow[\bot]{\pi} R\ \}$

```
    Release(l);
```

$\{\ d \overset{\frac{s_o}{2}}{\mapsto} \text{data} * l\square\xrightarrow[\frac{s_o}{2}]{\pi} R\ \}$

```
    return d;
```

$\{\ d \overset{\frac{s_o}{2}}{\mapsto} \text{data} * l\square\xrightarrow[\frac{s_o}{2}]{\pi} R \wedge ret = d\ \}$

```
  }
  else {
```

$\{\ init \mapsto 0 * \text{Hold } l, R, \bot * l\square\xrightarrow[\bot]{\pi} R\ \}$

```
    InitializeData (d);
```

$\{\ d \mapsto \text{data} * init \mapsto 0 * \text{Hold } l, R, \bot * l\square\xrightarrow[\bot]{\pi} R\ \}$

```
    init = 1;
```

$\{\ d \mapsto \text{data} * init \mapsto 1 * \text{Hold } l, R, \bot * l\square\xrightarrow[\bot]{\pi} R\ \}$

$\{\ d \overset{\frac{1}{2}}{\mapsto} \text{data} * \big(d \overset{\frac{1}{2}}{\mapsto} \text{data} * init \mapsto 1\big) * \text{Hold } l, R, \bot * l\square\xrightarrow[\bot]{\pi} R\ \}$

```
    Release(l)
```

$\{\ d \overset{\frac{1}{2}}{\mapsto} \text{data} * l\square\xrightarrow[\bot]{\pi} R\ \}$

```
    return d;
```

$\{\ d \overset{\frac{1}{2}}{\mapsto} \text{data} * l\square\xrightarrow[\bot]{\pi} R \wedge ret = d\ \}$

```
  }
}
```

$\{\ \exists \pi_s, d \overset{\pi_s}{\mapsto} \text{data} * l\square\xrightarrow[\bot]{\pi} R \wedge ret = d\ \}$

4 Stack Producer/consumer

$\{ \text{emp} \}$
`void create();`
$\{ \text{list } \epsilon \text{ hd} \}$

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

$\{ \text{list } ls \text{ hd} \}$
`void enq(int x);`
$\{ \text{list } x :: ls \text{ hd} \}$

$\{ \text{list } ls \text{ hd} \}$
`void deq();`
$\{ ls = \epsilon \wedge \text{list } ls \text{ hd} \wedge ret = null \vee$
$ls = x :: ls' \wedge \text{list } ls\text{hd} \wedge ret = x \}$

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

`/* Consumer */`
$\{ l\square\xrightarrow[\bot]{\pi} R \}$ \\\\ $R = \lambda h.\text{list (listof}(h)) \text{ hd} \wedge \text{history\_stack } h$
`void consumer(l){`
  $\{ l\square\xrightarrow[\bot]{\pi} R \}$
  `bool cont = true;`
  $\{ cont = true \wedge l\square\xrightarrow[\bot]{\pi} R \}$
  `while (cont) {`
    `Acquire(l);`

$\{ \; \mathit{cont} = \mathit{true} \wedge \exists h_o, \text{list } l \text{ hd} \wedge \text{history\_stack } h$
$* \text{ Hold } l, R, h_o * l \square\!\xrightarrow[\bot]{\pi} R \; \}$ $\backslash\backslash l = \text{listof}(h_o)$

```
if (isemp() ) {
   Release(l);
```

$\{ \; \mathit{cont} = \mathit{true} \wedge l \square\!\xrightarrow[\bot]{\pi} R \; \}$

```
} else {
```

$\{ \; \exists x, l'. l = x :: l \wedge \mathit{cont} = \mathit{true} \wedge$
$\text{list } l \text{ hd} \wedge \text{history\_stack } h * \text{Hold } l, R, h_o * l \square\!\xrightarrow[\bot]{\pi} R \; \}$

```
   ret = deq();
```

$\{ \; \mathit{ret} = x \wedge \mathit{cont} = \mathit{true} \wedge$
$\text{list } l' \text{ hd} \wedge \text{history\_stack } h * \text{Hold } l, R, h_o * l \square\!\xrightarrow[\bot]{\pi} R \; \}$

$\{ \; \mathit{ret} = x \wedge \mathit{cont} = \mathit{true} \wedge$
$\big(\text{list } (\text{listof}(h_o + t \hookrightarrow (l, l'))) \text{ hd} \wedge \text{history\_stack } h\big)$
$* \text{ Hold } l, R, h_o * l \square\!\xrightarrow[\bot]{\pi} R \; \}$ $\backslash\backslash$ $t = \text{last } h_o + 1$

```
   Release(l);
```

$\{ \; \mathit{ret} = x \wedge \mathit{cont} = \mathit{true} \wedge l \square\!\xrightarrow[t \hookrightarrow (l,l')]{\pi} R \; \}$

```
   cont = false;
```

$\{ \; \mathit{ret} = x \wedge \mathit{cont} = \mathit{false} \wedge l \square\!\xrightarrow[t \hookrightarrow (l,l')]{\pi} R \; \}$

```
}
```

$\{ \; \mathit{cont} = \mathit{true} \wedge l \square\!\xrightarrow[\bot]{\pi} R \vee \mathit{cont} = \mathit{false} \wedge \mathit{ret} = x \wedge l \square\!\xrightarrow[t \hookrightarrow (l,l')]{\pi} R \; \}$

```
}
```

$\{ \; \mathit{cont} = \mathit{false} \wedge \mathit{ret} = x \wedge l \square\!\xrightarrow[t \hookrightarrow (l,l')]{\pi} R \; \}$

```
return ret;
}
```

$\{ \; \mathit{ret} = x \wedge l \square\!\xrightarrow[t \hookrightarrow (x::l',l')]{\pi} R \; \}$

5 Queue Producer/consumer

```
struct node
{
    int info;
    struct node *ptr;
}*hd,*tl;
```

$\{$ emp $\}$

```
void create();
```

$\{$ hd $\mapsto$ _ $*$ tl $\mapsto$ _ $\}$

$\{$ list $(ls($ tl$, lst) * lst \mapsto (z, null) *$ hd $\mapsto lst$ $\}$

```
void enq(int x);
```

$\{$ list $ls :: x($tl$, lst') * lst' \mapsto (z, null) *$ hd $\mapsto lst'$ $\}$

$\{$ list $ls($tl$, lst) * lst \mapsto (z, null) *$ hd $\mapsto lst$ $\}$

```
void deq();
```

$\{$ $ls = \epsilon /$ list $ls($tl$, lst) * lst \mapsto (z, null) *$ hd $\mapsto lst \wedge ret = x$ $\}$

```
/* Create an empty queue */
void create()
{
    front = rear = NULL;
}

/* Enqueing the queue */
```

$\{$ front $\mapsto$ _ $*$ rear $\mapsto$ _ $\}$

```
void enq(int data)
{
    if (rear == NULL)
    {
        rear = (struct node *)malloc(1*sizeof(struct node));
        rear >ptr = NULL;
        rear >info = data;
        front = rear;
    }
    else
    {
        temp=(struct node *)malloc(1*sizeof(struct node));
        rear >ptr = temp;
        temp >info = data;
        temp >ptr = NULL;

        rear = temp;
```

```
    }
    count++;
}

/* Displaying the queue elements */
void display()
{
    front1 = front;

    if ((front1 == NULL) && (rear == NULL))
    {
        printf("Queue␣is␣empty");
        return;
    }
    while (front1 != rear)
    {
        printf("%d␣", front1 >info);
        front1 = front1 >ptr;
    }
    if (front1 == rear)
        printf("%d", front1 >info);
}

/* Dequeing the queue */
void deq()
{
    front1 = front;

    if (front1 == NULL)
    {
        printf("\n␣Error:␣Trying␣to␣display␣elements␣from␣empty␣queue");
        return;
    }
    else
        if (front1 >ptr != NULL)
        {
            front1 = front1 >ptr;
            printf("\n␣Dequed␣value␣:␣%d", front >info);
            free(front);
            front = front1;
        }
        else
        {
            printf("\n␣Dequed␣value␣:␣%d", front >info);
            free(front);
            front = NULL;
```

rear = NULL;
}
count ;
}

$\{\ l\Box\xrightarrow[\bot]{\pi} R\ \}$ \\ $R = \lambda v.init \mapsto 0 \wedge v = \bot \vee init \mapsto 1 * d \overset{\top - v}{\mapsto} \text{data} \wedge [\top > v]$

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) {

$\{\ init \mapsto 1 * d \overset{s_o}{\mapsto} \text{data} * \text{Hold } l, R, v_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$

$\{\ d \overset{\frac{s_o}{2}}{\mapsto} \text{data} * \left(init \mapsto 1 * d \overset{\top - (v_o + \frac{s_o}{2})}{\mapsto} \text{data}\right) * \text{Hold } l, R, v_o * l\Box\xrightarrow[\bot]{\pi} R\ \}$

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;

$\{\ d \mapsto \text{data} * init \mapsto 1 * \text{Hold } l, R, \bot * l\Box\xrightarrow[\bot]{\pi} R\ \}$

$\{\ d \overset{\frac{1}{2}}{\mapsto} \text{data} * \left(d \overset{\frac{1}{2}}{\mapsto} \text{data} * init \mapsto 1\right) * \text{Hold } l, R, \bot * l\Box\xrightarrow[\bot]{\pi} R\ \}$

Release(l)

$\{\ d \overset{\frac{1}{2}}{\mapsto} \text{data} * l\Box\xrightarrow[\bot]{\pi} R\ \}$

return d;

$\{\ d \overset{\frac{1}{2}}{\mapsto} \text{data} * l\Box\xrightarrow[\bot]{\pi} R \wedge ret = d\ \}$

    }
}

$\{\ \exists \pi_s, d \overset{\pi_s}{\mapsto} \mathrm{data} * l\Box\xrightarrow[\perp]{\pi} R \wedge ret = d\ \}$