Auxiliary Variables

To assist interpretation, the following auxiliary variables are added to the buffer:

- `t_used, t_valid`: the tail, split into t_{used} (the item exactly before it is allocated to a writer, but not yet ready to pop) and t_{valid} (the items before it are ready for a reader to pop).
- `h_used, h_red, h_blue`: similarly, the head is also split, h_{used} means allocated to a reader, while h_{valid} is further split into h_{red} and h_{blue}, indicating the number of items that are finished reading for each stream, and $h_{\text{red}} + h_{\text{blue}} = h_{\text{valid}}$. h_{valid} is thus implicit.
- `x_red, x_blue`: the x pointers in both input streams are copied into the buffer so that x can be shared by both streams to help interpretation. The original x pointers (private in each stream) are kept but not used in the interpretations.

Added to the streams are:

- `Red.H, Red.T`: booleans, meaning "Red owns the Head pointer/Tail pointer";
- `Blue.H, Blue.T`: likewise.

New Notations

To distinguish items of different colors for buffer interpretation, the following syntax is defined using our former `upto` notation and Python-like list comprehension:

- `B[h redupto t) = [item.v for item in B[h upto t) if item.c = Red]`
- `B[h blueupto t) = [item.v for item in B[h upto t) if item.c = Blue]`

Interpretations

$$
\begin{aligned}
Red.S_O &= Red.O[0, Red.y) & Blue.S_O &= Blue.O[0, Blue.y) \\
Red.S_{BO} &= Red.O[Red.y, h_{\text{red}}) & Blue.S_{BO} &= Blue.O[Blue.y, h_{\text{blue}}) \\
Red.S_B &= B[(h_{\text{red}} + h_{\text{blue}}) \text{ redupto } t_{\text{valid}}) & Blue.S_B &= B[(h_{\text{red}} + h_{\text{blue}}) \text{ blueupto } t_{\text{valid}}) \\
Red.S_{IB} &= B[t_{\text{valid}} \text{ redupto } (x_{\text{red}} + x_{\text{blue}})) & Blue.S_{IB} &= B[t_{\text{valid}} \text{ blueupto } (x_{\text{red}} + x_{\text{blue}})) \\
Red.S_I &= Red.I[x_{\text{red}}, Red.N) & Blue.S_I &= Blue.I[x_{\text{blue}}, Blue.N)
\end{aligned}
$$

Invariant

The invariant can be split into five parts:

$$
\begin{aligned}
S &: \text{Red}.S_O + \text{Red}.S_{BO} + \text{Red}.S_B + \text{Red}.S_{IB} + \text{Red}.S_I = K_{\text{Red}} \wedge \\
&\quad \text{Blue}.S_O + \text{Blue}.S_{BO} + \text{Blue}.S_B + \text{Blue}.S_{IB} + \text{Blue}.S_I = K_{\text{Blue}} \\
B &: h_{\text{Red}} + h_{\text{Blue}} \leq t_{\text{valid}} \wedge t_{\text{valid}} - h_{\text{Red}} - h_{\text{Blue}} \leq N_B \\
I &: \text{Red}.x \leq \text{Red}.N \wedge \text{Blue}.x \leq \text{Blue}.N \\
O &: \text{Red}.y \leq \text{Red}.N \wedge \text{Blue}.y \leq \text{Blue}.N \\
T &: \neg(\text{Red}.H \wedge \text{Blue}.H) \wedge \neg(\text{Red}.T \wedge \text{Blue}.T)
\end{aligned}
$$

S ensures for both streams the overall content is constant. B is the boundary invariant for the buffer. I and O ensures the boundary of input and output pointers for both streams. T is the mutual exclusion invariant so that each pointer can be held by at most one stream.

Local Correctness

For brevity, the annotations are only shown for `Red.push()` and `Red.pop()`. For Blue it is only reversing the color.

All variables are assigned 0 (or `false` for booleans) on initialization.

For `push()`, $\text{Red}.T \to t_{\text{valid}} = x_{\text{red}} + x_{\text{blue}}$ is true on initialization, and it is proved as below that it is preserved by `push()`, thus by induction it is a precondition and a postcondition. The same reasoning applies for the precondition and postcondition `Red.y = h_red` in `pop()`.

```
Red.push()
[Precondition] Red.T->(t_valid=x_red+x_blue), Red.x<Red.N, not Red.T
[Postcondition] Red.T->(t_valid=x_red+x_blue), Red.x<=Red.N, not Red.T

> S, B, I, O, T holds
> Red.x < Red.N and not Red.T
> Red.T -> (t_valid = x_red+x_blue)
do (not Red.T)
  > S, B, I, O, T, Red.x < Red.N, not Red.T, Red.T->(t_valid=x_red+x_blue)
  t' := t_used
  > S, B, I, O, T, Red.x < Red.N, not Red.T
  > Red.T -> (t_valid = x_red + x_blue)
  if t' = t_valid then
    > S, B, I, O, T, Red.x < Red.N, not Red.T, Red.T->(t_valid=x_red+x_blue)
    > t' = t_used -> t_valid = t_used
    if t - h_red - h_blue < N_B then
      > S, B, I, O, T, Red.x < Red.N, not Red.T, Red.T->(t_valid=x_red+x_blue)
      > t' = t_used -> (t_valid = t_used and t_valid - h_red - h_blue < N_B)
      CAS(Red.T, t_used, t'+1, t')
      > S, B, I, O, T, Red.x < Red.N
      > Red.T -> (t_valid = x_red + x_blue)
      > Red.T -> (t_valid + 1 = t_used and t_valid - h_red - h_blue < N_B)
      > Red.T -> not Blue.T
    fi
```

```
  fi
  > S, B, I, O, T, Red.x < Red.N
  > Red.T -> (t_valid = x_red + x_blue)
  > Red.T -> (t_valid + 1 = t_used and t_valid - h_red - h_blue < N_B)
  > Red.T -> not Blue.T
od
> S, B, I, O, T, Red.x < Red.N, Red.T, not Blue.T
> t_valid = x_red + x_blue
> t_valid + 1 = t_used, t_valid - h_red - h_blue < N_B
> Red.S_O + Red.S_BO + B[(h_red+h_blue) redupto t_valid) + [] +
  (Red.I[x_red] + Red.I[x_red + 1, Red.N)) = K_Red
B[t_valid mod N_B].v := Red.I[Red.x]
> S, B, I, O, T, Red.x < Red.N, Red.T, not Blue.T
> t_valid = x_red + x_blue
> t_valid + 1 = t_used, t_valid - h_red - h_blue < N_B
> Red.S_O + Red.S_BO + B[(h_red+h_blue) redupto t_valid) + [] +
  (B[t_valid mod N_B].v + Red.I[x_red + 1, Red.N)) = K_Red
B[t_valid mod N_B].c := Red
> S, B, I, O, T, Red.x < Red.N, Red.T, not Blue.T
> t_valid = x_red + x_blue
> t_valid + 1 = t_used, t_valid - h_red - h_blue < N_B
> Red.S_O + Red.S_BO + B[(h_red+h_blue) redupto t_valid) + [] +
  (B[t_valid mod N_B].v + Red.I[x_red + 1, Red.N)) = K_Red
x_red := x_red + 1
> S, B, I, O, T, Red.x < Red.N, Red.T, not Blue.T
> t_valid + 1 = x_red + x_blue
> t_valid + 1 = t_used, t_valid - h_red - h_blue < N_B
> Red.S_O + Red.S_BO + B[(h_red+h_blue) redupto t_valid) +
  B[t_valid mod N_B].v + Red.I[x_red, Red.N) = K_Red
Red.x := Red.x + 1
> S, B, I, O, T, Red.x <= Red.N, Red.T, not Blue.T
> t_valid + 1 = x_red + x_blue
> B[(t_valid + 1) redupto (x_red + x_blue)] = []
> t_valid + 1 = t_used, t_valid - h_red - h_blue < N_B
> Red.S_O + Red.S_BO + B[(h_red+h_blue) redupto t_valid) +
  B[t_valid mod N_B].v + Red.I[x_red, Red.N) = K_Red
Red.T := false
> S, B, I, O, T, Red.x <= Red.N, not Red.T, not Blue.T
> B[(t_valid + 1) redupto (x_red + x_blue)] = []
> t_valid + 1 = t_used, t_valid - h_red - h_blue < N_B
> Red.S_O + Red.S_BO + B[(h_red+h_blue) redupto t_valid) +
  B[t_valid mod N_B].v + Red.I[x_red, Red.N) = K_Red
t_valid := t_valid + 1
> S, B, I, O, T, Red.x <= Red.N, not Red.T
> Red.T -> (t_valid = x_red + x_blue)
> B[t_valid redupto (x_red + x_blue)] = []
> Red.S_O + Red.S_BO + B[(h_red+h_blue) redupto t_valid) + [] +
  Red.I[x_red, Red.N) = K_Red
```

```
Red.pop()
[Precondition] Red.y = h_red and Red.y < Red.N and not Red.H
[Postcondition] Red.y = h_red and Red.y <= Red.N and not Red.H

> S, B, I, O, T holds
> Red.y = h_red, Red.y < Red.N, not Red.H
do (not Red.H)
  > S, B, I, O, T, Red.y = h_red, Red.y < Red.N, not Red.H
  h' := h_used
  > S, B, I, O, T, Red.y = h_red, Red.y < Red.N, not Red.H
  if h' = h_red + h_blue then
    > S, B, I, O, T, Red.y = h_red, Red.y < Red.N, not Red.H
    > h' = h_used -> h_used = h_red + h_blue
    if h' < t_valid and B[h' mod N_B].c = Red then
      > S, B, I, O, T, Red.y = h_red, Red.y < Red.N, not Red.H
      > h' = h_used -> h_used = h_red + h_blue
      > h' = h_used -> h_used < t_valid and B[h_used mod N_B].c = Red
      CAS(Red.H, h_used, h'+1, h')
      > S, B, I, O, T, Red.y = h_red, Red.y < Red.N
      > Red.H -> h_used = h_red + h_blue + 1
      > Red.H -> h_red + h_blue < t_valid
      > Red.H -> B[(h_red + h_blue) mod N_B].c = Red
      > Red.H -> not Blue.H
    fi
  fi
  > S, B, I, O, T, Red.y = h_red, Red.y < Red.N
  > Red.H -> h_used = h_red + h_blue + 1
  > Red.H -> h_red + h_blue < t_valid
  > Red.H -> B[(h_red + h_blue) mod N_B].c = Red
  > Red.H -> not Blue.H
od
> S, B, I, O, T, Red.y = h_red, Red.y < Red.N, Red.H, not Blue.H
> h_used = h_red + h_blue + 1
> h_red + h_blue < t_valid
> B[(h_red + h_blue) mod N_B].c = Red
> Red.O[0, Red.y) + [] + (B[(h_red + h_blue) mod N_B].v +
  B[(h_red + h_blue + 1) redupto t_valid)) + Red.S_IB + Red.S_I = K_Red
Red.O[Red.y] := B[(h_red + h_blue) mod N_B].v
> S, B, I, O, T, Red.y = h_red, Red.y < Red.N, Red.H, not Blue.H
> h_used = h_red + h_blue + 1
> h_red + h_blue < t_valid
> B[(h_red + h_blue) mod N_B].c = Red
> Red.O[0, Red.y) + [] + (Red.O[Red.y] +
  B[(h_red + h_blue + 1) redupto t_valid)) + Red.S_IB + Red.S_I = K_Red
Red.H := false
> S, B, I, O, T, Red.y = h_red, Red.y < Red.N, not Red.H, not Blue.H
> h_used = h_red + h_blue + 1
> h_red + h_blue < t_valid
```

```
> B[(h_red + h_blue) mod N_B].c = Red
> Red.O[0, Red.y) + [] + (Red.O[Red.y] +
  B[(h_red + h_blue + 1) redupto t_valid)) + Red.S_IB + Red.S_I = K_Red
h_red := h_red + 1
> S, B, I, O, T, Red.y + 1 = h_red, Red.y < Red.N, not Red.H
> Red.O[0, Red.y) + Red.O[Red.y] + B[(h_red + h_blue) redupto t_valid) +
  Red.S_IB + Red.S_I = K_Red
Red.y := Red.y + 1
> S, B, I, O, T, Red.y = h_red, Red.y <= Red.N, not Red.H
> Red.O[0, Red.y) + [] + B[(h_red + h_blue) redupto t_valid) +
  Red.S_IB + Red.S_I = K_Red
```

Noninterference Proof

To prove noninterference between colours and between operations, we find that all assertions are a conjunction of predicates of 5 forms:

- The invariants S, B, I, O, T, which are proved to hold everywhere as long as other conjuncts hold.
- `{MyColor}.{x|y} {<|<=} {MyColor}.N`, `[not] {MyColor}.{T|H}`, which only involve local variables and constants, so interference is impossible.
- `{MyColor}.y = h_{myColor}`, `{MyColor}.y + 1 = h_{myColor}`, which involve shared variables h_{myColor}, but they are only ever updated by the concerned process itself, so it cannot be violated by interference.
- `{Red|Blue}.{H|T} -> not {Blue|Red}.{H|T}`, which are also parts of the invariant, so already covered.

This leaves only the following predicates that might suffer interference:

1. `{MyColor}.T -> (t_valid = x_red + x_blue)` only appears in `{MyColor}.push()`, and the only non-local statements that might could modify these variables are in `{OtherColor}.push()`.

   The first such statement is $\mathbf{x}_{\{\text{otherColor}\}} := x_{\{\text{otherColor}\}} + 1$, which has $\neg\{\text{MyColor}\}.T$ as precondition and postcondition, so `{MyColor}.T -> (t_valid = x_red + x_blue)` always holds.

   The second such statement is $\mathbf{t}_{\text{valid}} := t_{\text{valid}} + 1$, where the pre- and post- conditions are also compatible:

   $\triangleright\ \neg\{\text{MyColor}\}.T \wedge \neg\{\text{OtherColor}\}.T$
   $\triangleright\ \{\text{MyColor}\}.T \to (t_{\text{valid}} = x_{\text{red}} + x_{\text{blue}})$ (precondition holds)
   $\mathbf{t}_{\text{valid}} := t_{\text{valid}} + 1$
   $\triangleright\ \neg\{\text{OtherColor}\}.T$
   $\triangleright\ \{\text{OtherColor}\}.T \to (t_{\text{valid}} = x_{\text{red}} + x_{\text{blue}})$
   $\triangleright\ \{\text{MyColor}\}.T \to (t_{\text{valid}} = x_{\text{red}} + x_{\text{blue}})$ (no conflict)

2. Similarly,

$$\{\text{MyColor}\}.T \wedge (t_{\text{valid}} = x_{\text{red}} + x_{\text{blue}})$$

and

$$\{\text{MyColor}\}.T \wedge (t_{\text{valid}} + 1 = x_{\text{red}} + x_{\text{blue}})$$

also only appear in `{MyColor}.push()`, and are only possibly affected by the same two statements as above. But this is actually impossible by mutual exclusion, because they both have $\neg\{\text{MyColor}\}.T$ as precondition, which is a conflict.

3. `{MyColor}.T -> (t_valid + 1 = t_used)` only appears in `{MyColor}.push()`, and the only statements that might could modify these variables are in `{OtherColor}.push()`.

The first such statement is `CAS({OtherColor}.T, t_used, t'+1, t')`, whose non-interference can be proved by annotations:

$\triangleright \{\text{MyColor}\}.T \rightarrow t_{\text{valid}} + 1 = t_{\text{used}}$
$\triangleright t' = t_{\text{used}} \rightarrow t_{\text{valid}} = t_{\text{used}}$ (precondition no conflict)
$\text{CAS}(\{\text{OtherColor}\}.T, t_{\text{used}}, t' + 1, t')$
$\triangleright \{\text{OtherColor}\}.T \rightarrow \neg\{\text{MyColor}\}.T$
$\triangleright \{\text{OtherColor}\}.T \rightarrow t_{\text{valid}} + 1 = t_{\text{used}}$
$\triangleright \{\text{MyColor}\}.T \rightarrow t_{\text{valid}} + 1 = t_{\text{used}}$ (postcondition no conflict)

The second such statement is $\mathbf{t}_{\text{valid}} := t_{\text{valid}} + 1$, where the pre- and post- conditions are also compatible:

$\triangleright \{\text{MyColor}\}.T \rightarrow t_{\text{valid}} + 1 = t_{\text{used}}$
$\triangleright t_{\text{valid}} + 1 = t_{\text{used}}$
$\triangleright \neg\{\text{MyColor}\}.T \wedge \neg\{\text{OtherColor}\}.T$ (precondition no conflict)
$\mathbf{t}_{\text{valid}} := t_{\text{valid}} + 1$
$\triangleright \{\text{MyColor}\}.T \rightarrow t_{\text{valid}} + 1 = t_{\text{used}}$
$\triangleright \neg\{\text{OtherColor}\}.T$ (postcondition no conflict)

4. Similarly, $\neg\{\text{OtherColor}\}.T \wedge t_{\text{valid}} + 1 = t_{\text{used}}$ also only appears in `{MyColor}.push()`, and are only possibly affected by the same two statements as above.

For `CAS({OtherColor}.T, t_used, t'+1, t')`, the noninterference can be proved by annotations:

$\triangleright \neg\{\text{OtherColor}\}.T \wedge t_{\text{valid}} + 1 = t_{\text{used}}$
$\triangleright t' = t_{\text{used}} \rightarrow t_{\text{valid}} = t_{\text{used}}$ (so, $t' \neq t_{\text{used}}$)
$\text{CAS}(\{\text{OtherColor}\}.T, t_{\text{used}}, t' + 1, t')$
$\triangleright \neg\{\text{OtherColor}\}.T \wedge t_{\text{valid}} + 1 = t_{\text{used}}$ (postcondition no conflict)

For $\mathbf{t}_{\text{valid}} := t_{\text{valid}} + 1$:

$\triangleright \neg\{\text{OtherColor}\}.T \wedge t_{\text{valid}} + 1 = t_{\text{used}}$
$\triangleright \neg\{\text{MyColor}\}.T \wedge \neg\{\text{OtherColor}\}.T$ (precondition no conflict)
$\mathbf{t}_{\text{valid}} := t_{\text{valid}} + 1$
$\triangleright \neg\{\text{OtherColor}\}.T$ (postcondition does not care about t_{used}, so no conflict)

5. Following the same reasoning in 3 and 4,

$$\{\text{MyColor}\}.T \to (t_{\text{valid}} - h_{\text{red}} - h_{\text{blue}} < N_B)$$

and

$$\neg\{\text{OtherColor}\}.T \wedge (t_{\text{valid}} - h_{\text{red}} - h_{\text{blue}} < N_B)$$

in {`MyColor`}`.push()` cannot be interfered by {`OtherColor`}`.push()`. In addition, h_{red} and h_{blue} can be updated by $h_{\text{color}} := h_{\text{color}} + 1$ in `pop()` of both colors, but it only makes the lefthand side of the inequality smaller, so they still holds.

6. $t' = t_{\text{used}} \to t_{\text{valid}} = t_{\text{used}}$ and $t' = t_{\text{used}} \to t_{\text{valid}} - h_{\text{red}} - h_{\text{blue}} < N_B$ only appears in `push()`, where t' is a local variable. Wherever interference might happen, it is possible to set $t' \neq t_{\text{used}}$, so it still holds.

7.
$$B[(t_{\text{valid}} + 1)\ \{\text{myColor}\}\text{upto}\ (x_{\text{red}} + x_{\text{blue}})] = []$$

and

$$B[t_{\text{valid}}\ \{\text{myColor}\}\text{upto}\ (x_{\text{red}} + x_{\text{blue}})] = []$$

only appears in {`MyColor`}`.push()`, and the only statements that might invalidate it is in {`OtherColor`}`.push()`.

First, $x_{\{\text{otherColor}\}} := x_{\{\text{otherColor}\}} + 1$ makes the range larger by 1, but it does not add or remove items of `myColor`, so they still holds.

Second, $t_{\text{valid}} := t_{\text{valid}} + 1$ only makes the range smaller, so they can only stay empty.

8. Following the same reasoning as 6, in `pop()`, $h' = h_{\text{used}} \to h_{\text{used}} = h_{\text{red}} + h_{\text{blue}}$ and $h' = h_{\text{used}} \to h_{\text{used}} < t_{\text{valid}} \wedge B[h_{\text{used}} \bmod N_B].c = \text{Red}$ also hold where interference is possible.

9. Following the same reasoning as in 3 and 4,

$\{\text{MyColor}\}.H \to$
$h_{\text{used}} = h_{\text{red}} + h_{\text{blue}} + 1 \wedge h_{\text{red}} + h_{\text{blue}} < t_{\text{valid}} \wedge B[(h_{\text{red}} + h_{\text{blue}}) \bmod N_B].c = \text{Red}$

and

$\neg\{\text{OtherColor}\}.H \wedge$
$h_{\text{used}} = h_{\text{red}} + h_{\text{blue}} + 1 \wedge h_{\text{red}} + h_{\text{blue}} < t_{\text{valid}} \wedge B[(h_{\text{red}} + h_{\text{blue}}) \bmod N_B].c = \text{Red}$

in {`MyColor`}`.pop()` cannot be interfered by {`OtherColor`}`.pop()`. In addition, t_{valid} can be updated by $t_{\text{valid}} := t_{\text{valid}} + 1$ in `push()` of both colors, but it only makes t_{valid} larger, so the inequality still holds.

Partial Correctness

From initialization $x_{\{\text{color}\}}, h_{\{\text{color}\}}, h_{\text{used}}, t_{\text{used}}, t_{\text{valid}}, \{\text{Color}\}.x, \{\text{Color}\}.y := 0, 0, 0, 0, 0, 0, 0$ and the extended invariant, we know that $[\ ] + [\ ] + [\ ] + [\ ] + I[0, N) = K_{\{\text{Color}\}}$, so $K_{\{\text{Color}\}} = I[0, N)$.

The extended invariant plus the termination condition ($Red.y = Red.N$) for **R** (the receiver) implies that

$$\{\text{Color}\}.O[0,N) + \{\text{Color}\}.S_{BO} + \{\text{Color}\}.S_B + \{\text{Color}\}.S_{IB} + \{\text{Color}\}.S_I = K_{\{\text{Color}\}},$$

which implies

$$N + |\{\text{Color}\}.S_{BO}| + |\{\text{Color}\}.S_B| + |\{\text{Color}\}.S_{IB}| + |\{\text{Color}\}.S_I| = |\{\text{Color}\}.I[0,N)| = N,$$

therefore

$$|\{\text{Color}\}.S_{BO}| = |\{\text{Color}\}.S_B| = |\{\text{Color}\}.S_{IB}| = |\{\text{Color}\}.S_I| = 0,$$

thus

$$\{\text{Color}\}.O[0,N) = \{\text{Color}\}.I[0,N).$$