Dequeue-Side Semantics

Disclaimer: we assume familiarity with citeOG, adopt its notational conventions, and steal its definitions! Further, we work with a specific subset of *Rio*, denoted **Rio**, namely

$$\frac{c \in \textbf{Class}}{\textbf{edf}[c], \textbf{fifo}[c] \in \textbf{Rio}}\ \text{set2stream} \qquad \frac{n \in \mathbb{N} \qquad rs \in \textbf{Rio}^n}{\textbf{strict}[rs], \textbf{rr}[rs] \in \textbf{Rio}}\ \text{stream2stream}$$

where **Class** is an opaque collection of *classes*.

1 Structure and Semantics of Rio Trees

Definition 1.1. For topology $t \in \textbf{Topo}$, the set $\textbf{RioTree}(t)$ of *Rio trees* over t is defined by

$$\frac{p \in \textbf{PIFO}(\textbf{Pkt}) \qquad c \in \textbf{Class}}{\text{Leaf}(p, c) \in \textbf{RioTree}(*)} \qquad \frac{ts \in \textbf{Topo}^n \qquad \forall 1 \leq i \leq n.\ qs[i] \in \textbf{RioTree}(ts[i])}{\text{Internal}(qs) \in \textbf{RioTree}(\text{Node}(ts))}$$

These are trees with leaves decorated by both classes and PIFOs.

Definition 1.2. For topology $t \in \textbf{Topo}$, the set $\textbf{OrdTree}(t)$ of *ordered trees* over t is defined by

$$\frac{}{\text{Leaf} \in \textbf{OrdTree}(*)} \qquad \frac{ts \in \textbf{Topo}^n \qquad rs \in \textbf{Rk}^n \qquad \forall 1 \leq i \leq n.\ os[i] \in \textbf{OrdTree}(ts[i])}{\text{Internal}(rs, os) \in \textbf{OrdTree}(\text{Node}(ts))}$$

These are trees with each internal node's child given a rank, thereby inducing a total ordering of children.

Let $\textbf{flow} : \textbf{Pkt} \to \textbf{Class}$ be an opaque mapping from packets to the class they belong to (flow inference).

Definition 1.3. For $t \in \textbf{Topo}$, define $\text{push} : \textbf{RioTree}(t) \times \textbf{Pkt} \times \textbf{Rk} \rightharpoonup \textbf{RioTree}(t)$ such that

$$\frac{\textbf{flow}(\text{pkt}) = c \qquad \text{push}(p, \text{pkt}, r) = p'}{\text{push}(\text{Leaf}(p, c), \text{pkt}, r) = \text{Leaf}(p', c)} \qquad \frac{\forall 1 \leq i \leq |qs|.\ \text{push}(qs[i], \text{pkt}, r) = qs'[i]}{\text{push}(\text{Internal}(qs), \text{pkt}, r) = \text{Internal}(qs')} \qquad \frac{\textbf{flow}(\text{pkt}) \neq c}{\text{push}(\text{Leaf}(p, c), \text{pkt}, r) = \text{Leaf}(p, c)}$$

Informally, we recursively push to all subtrees but only the PIFOs on leaves with the packet's flow are updated.

Definition 1.4. For $t \in \textbf{Topo}$, define $\text{pop} : \textbf{RioTree}(t) \times \textbf{OrdTree}(t) \rightharpoonup \textbf{Pkt} \times \textbf{RioTree}(t)$ such that

$$\frac{\text{pop}(p) = (\text{pkt}, p')}{\text{pop}(\text{Leaf}(p, c), \text{Leaf}) = (\text{pkt}, \text{Leaf}(p', c))} \qquad \frac{\begin{array}{c}\exists 1 \leq i \leq |qs|.\ \text{pop}(qs[i], os[i]) = (\text{pkt}, q) \\ \forall 1 \leq j \leq |qs|.\ j \neq i \wedge \text{pop}(qs[j], os[j]) = (\text{pkt}', q') \implies rs[i] < rs[j]\end{array}}{\text{pop}(\text{Internal}(qs), \text{Internal}(rs, os)) = (\text{pkt}, \text{Internal}(qs[q/i]))}$$

Informally, we recursively pop the smallest ranked, poppable subtree.

2 PIFO & Rio Controls

$$\frac{z_{\text{pre-push}}(s, \text{pkt}) = (r, s') \qquad \text{push}(q, \text{pkt}, r) = q'}{\text{push}((s, q, z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}}), \text{pkt}) = (s', q', z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}})} \text{ RioCtrl-Push}$$

$$\frac{z_{\text{pre-pop}}(s) = (o, s') \qquad \text{pop}(q, o) = (\text{pkt}, q') \qquad z_{\text{post-pop}}(s', \text{pkt}) = s''}{\text{pop}((s, q, z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}})) = (\text{pkt}, (s'', q', z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}}))} \text{ RioCtrl-Pop}$$

$$\frac{z_{\text{pre-push}}(s, \text{pkt}) = (pt, s') \qquad \text{push}(q, \text{pkt}, pt) = q'}{\text{push}((s, q, z_{\text{pre-push}}, z_{\text{post-pop}}), \text{pkt}) = (s', q', z_{\text{pre-push}}, z_{\text{post-pop}})} \text{ PIFOCtrl-Push}$$

$$\frac{\text{pop}(q) = (\text{pkt}, q') \qquad z_{\text{post-pop}}(s, \text{pkt}) = s'}{\text{pop}((s, q, z_{\text{pre-push}}, z_{\text{post-pop}})) = (\text{pkt}, (s', q', z_{\text{pre-push}}, z_{\text{post-pop}}))} \text{ PIFOCtrl-Pop}$$

Figure 1. Pushing and Popping Controls

For these notes, we'll refer to *controls* from citeOG as *PIFO controls*.

Definition 2.1. Let $t \in \textbf{Topo}$ and *scheduling transactions* $z_{\text{pre-push}}$ and $z_{\text{post-pop}}$ be partial functions

$$\begin{aligned} z_{\text{pre-push}} &: \textbf{St} \times \textbf{Pkt} \rightharpoonup \textbf{Path}(t) \times \textbf{St} \\ z_{\text{post-pop}} &: \textbf{St} \times \textbf{Pkt} \rightharpoonup \textbf{St} \end{aligned}$$

Define $\textbf{PIFOControl}(t, z_{\text{pre-push}}, z_{\text{post-pop}})$ to be the set of quadruples

$$(s, q, z_{\text{pre-push}}, z_{\text{post-pop}})$$

where $s \in \textbf{St}$ and $q \in \textbf{PIFOTree}(t)$.

Definition 2.2. Let $t \in \textbf{Topo}$ and *scheduling transactions* $z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}}$ be partial functions

$$\begin{aligned} z_{\text{pre-push}} &: \textbf{St} \times \textbf{Pkt} \rightharpoonup \textbf{Rk} \times \textbf{St} \\ z_{\text{pre-pop}} &: \textbf{St} \rightharpoonup \textbf{OrdTree}(t) \times \textbf{St} \\ z_{\text{post-pop}} &: \textbf{St} \times \textbf{Pkt} \rightharpoonup \textbf{St} \end{aligned}$$

Define $\textbf{RioControl}(t, z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}})$ to be the set of quintuples

$$(s, q, z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}})$$

where $s \in \textbf{St}$ and $q \in \textbf{RioTree}(t)$.

Both controls admit push and pop operations:

$$\begin{aligned} \text{push} &: \textbf{PIFOControl}(t, z_{\text{pre-push}}, z_{\text{post-pop}}) \times \textbf{Pkt} \to \textbf{PIFOControl}(t, z_{\text{pre-push}}, z_{\text{post-pop}}) \\ \text{pop} &: \textbf{PIFOControl}(t, z_{\text{pre-push}}, z_{\text{post-pop}}) \to \textbf{Pkt} \times \textbf{PIFOControl}(t, z_{\text{pre-push}}, z_{\text{post-pop}}) \\ \text{push} &: \textbf{RioControl}(t, z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}}) \times \textbf{Pkt} \to \textbf{RioControl}(t, z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}}) \\ \text{pop} &: \textbf{RioControl}(t, z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}}) \to \textbf{Pkt} \times \textbf{RioControl}(t, z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}}) \end{aligned}$$

Their semantics are written out in full in Figure 1.

Definition 2.3. Define the $\to \subseteq \textbf{PIFOControl}(t, z_{\text{pre-push}}, z_{\text{post-pop}}) \times \textbf{PIFOControl}(t, z_{\text{pre-push}}, z_{\text{post-pop}})$ by

$$\frac{\text{pkt} \in \textbf{Pkt} \qquad \text{push}\, c\, \text{pkt} = c'}{c \to c'} \qquad\qquad \frac{\text{pop}\, c = (\text{pkt}, c')}{c \to c'}$$

i.e. $c \to c'$ if c' is a push or pop from c. We write $\to^*$ for the reflexive transitive closure of $\to$.

Definition 2.4. A partial function

$$f : \textbf{RioControl}(t, z_{\text{pre-push}}, z_{\text{pre-pop}}, z_{\text{post-pop}}) \rightharpoonup \textbf{PIFOControl}(t', z'_{\text{pre-push}}, z'_{\text{post-pop}})$$

is *simulation* if, for all $\text{pkt} \in \textbf{Pkt}$ and $f(c_1) = c_2$, the following conditions are satisfied:

(1) There exists $f(c_{\text{init}}) = c'_{\text{init}}$ where the trees in c_{init} and c'_{init} hold empty PIFOs at the leaves.
(2) If $c_{\text{init}} \to^* c_1$,

$$\text{pop}(c_1) \text{ is undefined} \implies \text{pop}(c_2) \text{ is undefined} \tag{a}$$

$$\text{pop}(c_1) = (\text{pkt}, c'_1) \implies \text{pop}(c_2) = (\text{pkt}, c'_2) \wedge f(c'_1) = c'_2 \tag{b}$$

$$\text{push}(c_1, \text{pkt}) = c'_1 \implies \text{push}(c_2, \text{pkt}) = c'_2 \wedge f(c'_1) = c'_2 \tag{c}$$

3 Enqueue and Dequeue-Side Equivalence

All further discussion takes **St** $=$ set of dictionaries mapping **string** $\to$ **float** and **Rk** $=$ **Class** $= \mathbb{N}$.

3.1 Round-Robin

```
def z_post-pop(st, pkt):
    f = flow(pkt)
    st["turn"] = float((f + 1) % n)
    return st
```

```
def z_pre-pop(st):
    turn = int(s["turn"])
    rs = []
    for i in range(n):
        rs.append((i - turn) % n)
    return Internal(rs, [Leaf] * n)
```

```
def z_pre-push(st, pkt):
    r = st["counter"]
    st["counter"] += 1
    return float_to_int(r)
```

(a) Rio Control

```
z_post-pop(st, pkt):
    f = flow(pkt)
    i = int(s["turn"])
    while i != f:
        st["r_" + str(i)] += n
        i = (i + 1) % n
    st["turn"] = float((f + 1) % n)
    return st
```

```
z_pre-push(st, pkt):
    r = st["counter"]
    st["count"] += 1
    f = flow(pkt)
    rank_ptr = "r_" + str(f)
    r_i = int(st[rank_ptr])
    st["rank_ptr"] += float(n)
    return ((f, r_i) :: r, st)
```

(b) PIFO Control

Figure 2. Scheduling Transactions

For $n \in \mathbb{N}$, let's put our theory to use by constructing PIFO and Rio controls for

$$\mathbf{rr}[(\mathbf{FIFO}[0], \mathbf{FIFO}[1], \ldots, \mathbf{FIFO}[n-1])]$$

Both controls use the same underlying topology, namely

$$t = \text{Node}((\underbrace{*, *, \ldots, *}_{n \text{ times}}))$$

Figure **??** describes the scheduling transactions
and showing they're in simulation. This would show the equivalence of enqueue and dequeue-side semantics for a specific type of program!