PIEO Trees for Fun and Profit

We assume familiarity with [MLF+23], adopt its notational conventions, and borrow many of its definitions!

1 Structure & Semantics

Our PIEO trees are near identical to PIFO trees, with only two modifications:

(1) All nodes hold PIEOs instead of PIFOs
(2) Internal PIEOs hold just enough extra data to compute predicates of packets that live in subtrees.

To make "just enough extra data" precise, let f be a predicate on $\mathbf{Pkt}$. We'll presuppose the existence of a set $\mathbf{Data}_f$ and associated surjective map $\mathrm{data}_f : \mathbf{Pkt} \to \mathbf{Data}_f$. This map and set are such that, for $\mathrm{pkt}, \mathrm{pkt}' \in \mathbf{Pkt}$,

$$\mathrm{data}_f(\mathrm{pkt}) = \mathrm{data}_f(\mathrm{pkt}') \implies f(\mathrm{pkt}) = f(\mathrm{pkt}')$$

In other words, all preimages of $d \in \mathbf{Data}_f$ evaluate to the same truth value under f. For this reason,

Abuse of Notation 1.1. We'll often write $f(d)$ when we mean $f(\mathrm{pkt})$ for some $\mathrm{pkt} \in f^{-1}(d)$.

Broadly, this set $\mathbf{Data}_f$ is designed to encode just the parts of $\mathbf{Pkt}$ necessary for computing $f(\mathrm{pkt})$.

Example 1.2. For any f, it is legal to set $\mathrm{Data}_f = \mathbf{Pkt}$ and data_f to be the identity map. We'd expect this to work since each $\mathrm{pkt} \in \mathbf{Pkt}$ certainly holds all the information necessary to compute $f(\mathrm{pkt})$.

We can now formally define PIEO trees!

Definition 1.3 (PIEO tree)**.** Let $t \in \mathbf{Topo}$ be a topology and f a predicate over $\mathbf{Pkt}$. The set of *PIEO trees* over t and f, denoted $\mathbf{PIEOTree}(t, f)$, is defined inductively by

$$\frac{p \in \mathbf{PIEO}(\mathbf{Pkt}, f)}{\mathrm{Leaf}(p) \in \mathbf{PIEOTree}(*)} \qquad \frac{n \in \mathbb{N} \qquad ts \in \mathbf{Topo}^n \qquad p \in \mathbf{PIEO}(\{1, \ldots, n\} \times \mathbf{Data}_f(\mathbf{Pkt}), f) \qquad \forall i \in [1, n].\ qs[i] \in \mathbf{PIEOTree}(ts[i])}{\mathrm{Internal}(qs, p) \in \mathbf{PIEO}(\mathrm{Node}(ts))}$$

References

[MLF+23] Anshuman Mohan, Yunhe Liu, Nate Foster, Tobias Kappé, and Dexter Kozen. Formal abstractions for packet scheduling, 2023.