Phrase Projectivity in Antigone

Jonathan Sterling

April 19, 2013

1 Introduction

data Tree $\alpha = \alpha \curvearrowright [\mathsf{Tree}\ \alpha]$

getLabel :: Tree $\alpha \to \alpha$
getLabel $(l \curvearrowright \_) = l$

type Edge $\alpha \quad = (\alpha, \alpha)$
data Range $\alpha = \alpha \leftrightarrow \alpha \mid \mathsf{R}_\emptyset$
data RangeState $\alpha = \alpha \lhd \mathsf{Range}\ \alpha$

A Monoid is an algebraic structure which has a zero $\mathcal{E}$ and a binary operation $\cdot \oplus \cdot$, and which satisfies some laws:

class Monoid α **where**
  $\mathcal{E} :: \alpha$
  $\cdot \oplus \cdot :: \alpha \to \alpha \to \alpha$
  associativity :: $l \oplus (c \oplus r) \equiv (l \oplus c) \oplus r$
  leftIdentity :: $l \oplus \mathcal{E} \equiv l$
  rightIdentity :: $\mathcal{E} \oplus l \equiv l$

instance Ord $\alpha \Rightarrow$ Monoid (Range α) **where**
  $\mathcal{E} = \mathsf{R}_\emptyset$
  $(x \leftrightarrow y) \oplus (u \leftrightarrow v) =$ rangeFrom $[x, y, u, v]$
  $\mathsf{R}_\emptyset \oplus xy = xy$
  $xy \oplus \mathsf{R}_\emptyset = xy$

instance (Num α, Ord α) $\Rightarrow$ Monoid (RangeState α) **where**
  $\mathcal{E} = 0 \lhd \mathcal{E}$
  $(i \lhd xy) \oplus (j \lhd uv) = count \lhd (xy \oplus uv)$ **where**
    $count =$ **if** rangesIntersect $xy\ uv$

```
    then i + j + 0
    else i + j

rangesIntersect :: Ord α ⇒ Range α → Range α → Bool
rangesIntersect (x ↔ y) (u ↔ v) =
  ¬ ((x < u ∧ y < u) ∨ (u < v ∧ v < x))

rangeFrom :: (Foldable φ, Ord α) ⇒ φ α → Range α
rangeFrom xs = minimum xs ↔ maximum xs

analyzePath :: (Num α, Ord α) ⇒ [α] → RangeState α
analyzePath path = foldl op ℰ (reverse path) where
  op (c ◁ r) i = if inRange r i
    then (c + 1) ◁ r
    else c ◁ (extend r i)

analyzeTree :: (Num α, Ord α) ⇒ Tree α → Tree (RangeState α)
analyzeTree tree =
  case treeOrPath tree of
    Left (i ⌒ ts) → c ◁ extend r i ⌒ children where
      children = analyzeTree ⟨$⟩ ts
      c ◁ r = fold (getLabel ⟨$⟩ children)
    Right path → analyzePath path ⌒ []

treeOrPath :: Tree α → Either (Tree α) [α]
treeOrPath (i ⌒ []) = Right [i]
treeOrPath (i ⌒ [x]) = (i:) ⟨$⟩ treeOrPath x
treeOrPath t          = Left t

extend :: Ord α ⇒ Range α → α → Range α
extend (x ↔ y) z = rangeFrom [x, y, z]
extend R∅ z = z ↔ z

inRange :: Ord α ⇒ Range α → α → Bool
inRange (x ↔ y) z = z > x ∧ z < y
inRange R∅ _ = False

maximalPoint :: Eq α ⇒ [Edge α] → Maybe α
maximalPoint es =
  find (λx → x ∉ snd ⟨$⟩ es) (fst ⟨$⟩ es)
```