CS 457/557 Functional Programming

Lecture 8
Regions
The Region Data Type

• A region represents an area on the two-dimensional Cartesian plane.
• It is represented by a tree-like data structure.

```haskell
data Region =
  Shape Shape               -- primitive shape
  | Translate Vector Region   -- translated region
  | Scale Vector Region       -- scaled region
  | Complement Region         -- inverse of region
  | Region `Union` Region     -- union of regions
  | Region `Intersect` Region -- intersection of regions
  | Empty
  deriving Show

type Vector = (Float, Float)
```
Questions about Regions

• Why is Region tree-like?

• What is the strategy for writing functions over regions?

• Is there a fold-function for regions?
  – How many parameters does it have?
  – What is its type?

• Can one define infinite regions?

• What does a region mean?
Sets and Characteristic Functions

• How can we represent an infinite set in Haskell? E.g.:
  – the set of all even numbers
  – the set of all prime numbers

• We could use an infinite list, but then searching it might take a very long time! (Membership becomes semi-decidable.)

• The characteristic function for a set containing elements of type \( z \) is a function of type \( z \to \text{Bool} \) that indicates whether or not a given element is in the set. Since that information completely characterizes a set, we can use it to represent a set:

\[
\text{type Set a = a \to \text{Bool}}
\]

• For example:

\[
\text{even} :: \text{Set Integer} \quad -- \text{Integer} \to \text{Bool}
\]

\[
\text{even } x = (x \ `\text{mod}` 2) == 0
\]
Combining Sets

• If sets are represented by characteristic functions, then how do we represent the:
  – union of two sets?
  – intersection of two sets?
  – complement of a set?

• In-class exercise – define the following Haskell functions:

```haskell
s1 `union` s2 =

s1 `intersect` s2 =

complement s =
```

• We will use these later to define similar operations on regions.
Why Regions?

Regions (as defined in the text) are interesting because:

- They allow us to build complex “shapes” from simpler ones.
- They illustrate the use of tree-like data structures.
- They “solve” the problem of having rectangles and ellipses centered about the origin.
- Their meaning can be given as characteristic functions, since a region denotes the set of points contained within it.
Characteristic Functions for Regions

• We define the meaning of regions by a function:
  \( \text{containsR} :: \text{Region} \rightarrow \text{Coordinate} \rightarrow \text{Bool} \)

• Here \( \text{type coordinate} = (\text{Float}, \text{Float}) \)

• Note that \( \text{containsR} \ r :: \text{Coordinate} \rightarrow \text{Bool} \), which is a characteristic function. So \( \text{containsR} \) “gives meaning to” regions.

• Another way to see this:
  \( \text{containsR} :: \text{Region} \rightarrow \text{Set Coordinate} \)

• We can define \( \text{containsR} \) recursively, using pattern matching over the structure of a Region.

• Since the base cases of the recursion are primitive shapes, we also need a function that gives meaning to primitive shapes; we will call this function \( \text{containsS} \).
Rectangle s1 s2 `containsS` (x,y)
= let t1 = s1/2
  t2 = s2/2
  in -t1<=x && x<=t1 && -t2<=y && y<=t2
Ellipse

Ellipse $r_1 \ r_2$ `containsS` $(x, y)$

$= (x/r_1)^2 + (y/r_2)^2 \leq 1$
The Left Side of a Line

For a ray directed from point $a$ to point $b$, a point $p$ is to the left of the ray (facing from $a$ to $b$) when:

$$\text{isLeftOf} :: \text{Coordinate} -> \text{Ray} -> \text{Bool}$$

$$(px,py) \text{ `isLeftOf` } ((ax,ay),(bx,by))$$

$$= \text{let } (s,t) = (px-ax, py-ay)$$
$$\quad (u,v) = (px-bx, py-by)$$
$$\quad \text{in } s*v >= t*u$$

$$(p = (px,py))$$

$$a = (ax,ay)$$

$$b = (bx,by)$$

\[
\text{type Ray} = (\text{Coordinate, Coordinate})
\]
Polygon

A point \( p \) is contained within a (convex) polygon if it is to the left of every side, when they are followed in counter-clockwise order.

\[
\text{Polygon } \text{pts} \ `\text{containsS}` \ p \\
= \text{let } \text{shiftpts} = \text{tail pts} ++ \text{[head pts]} \\
\text{leftOfList} = \text{map (isLeftOfp } p) \\
(\text{zip pts shiftpts}) \\
\text{in } \text{foldr } (\&\&) \ \text{True leftOfList}
\]
Right Triangle

\[
\text{RtTriangle } s1 \ s2 \ `\text{containsS`} \ p \\
= \text{Polygon } [(0,0),(s1,0),(0,s2)] \ `\text{containsS`} \ p
\]
Putting it all Together

\[
\text{containsS} :: \text{Shape} \rightarrow \text{Coordinate} \rightarrow \text{Bool} \\
\text{Rectangle } s1 \ s2 \ `\text{containsS}` \ (x,y) \\
= \text{let } t1 = s1/2; \ t2 = s2/2 \\
\phantom{=} \quad \text{in } -t1 \leq x \land x \leq t1 \land -t2 \leq y \land y \leq t2 \\
\text{Ellipse } r1 \ r2 \ `\text{containsS}` \ (x,y) \\
= (x/r1)^2 + (y/r2)^2 \leq 1 \\
\text{Polygon } pts \ `\text{containsS}` \ p \\
= \text{let } shiftpts = \text{tail } pts ++ [\text{head } pts] \\
\phantom{=} \quad \text{leftOfList} = \\
\phantom{=} \quad \quad \text{map } (\text{isLeftOfp } p) \ (\text{zip } pts \ shiftpts) \\
\phantom{=} \quad \text{in } \text{foldr } (\land) \ True \ \text{leftOfList} \\
\text{RtTriangle } s1 \ s2 \ `\text{containsS}` \ p \\
= \text{Polygon } [(0,0),(s1,0),(0,s2)] \ `\text{containsS}` \ p
\]
Defining \texttt{containsR} using Recursion

\begin{verbatim}
containsR :: Region -> Coordinate -> Bool
Shape s `containsR` p = s `containsS` p
Translate (u,v) r `containsR` (x,y)
    = r `containsR` (x-u,y-v)
Scale (u,v) r     `containsR` (x,y)
    = r `containsR` (x/u,y/v)
Complement r      `containsR` p
    = not (r `containsR` p)
r1 `Union` r2     `containsR` p
    = r1 `containsR` p || r2 `containsR` p
r1 `Intersect` r2 `containsR` p
    = r1 `containsR` p && r2 `containsR` p
Empty     `containsR` p = False
\end{verbatim}
An Algebra of Regions

• Note that, for any $r_1$, $r_2$, and $r_3$:
  
  $$(r_1 \ `\ Union` \ (r_2 \ `\ Union` \ r_3)) \ `\ containsR` \ p$$

  if and only if:
  
  $$(r_1 \ `\ Union` \ r_2) \ `\ Union` \ r_3)) \ `\ containsR` \ p$$

  which we can abbreviate as:

  $$(r_1 \ `\ Union` \ (r_2 \ `\ Union` \ r_3))$$

  $$\equiv ((r_1 \ `\ Union` \ r_2) \ `\ Union` \ r_3)$$

• In other words, $\text{Union}$ is associative.

• We can prove this fact via calculation.
Proof of Associativity

(r1 `Union` (r2 `Union` r3)) `containsR` p
= (r1 `containsR` p) ||
    ((r2 `Union` r3) `containsR` p)
= (r1 `containsR` p) ||
    ((r2 `containsR` p) || (r3 `containsR` p))
= ((r1 `containsR` p) || (r2 `containsR` p)) ||
    (r3 `containsR` p)
= ((r1 `Union` r2) `containsR` p) ||
    (r3 `containsR` p)
= ((r1 `Union` r2) `Union` r3) `containsR` p

(Note that the proof depends on the associativity of (|||), which can also be proved by calculation, but we take as given.)
More Axioms

There are many useful axioms for regions:

1) \textbf{Union} and \textbf{Intersect} are associative.
2) \textbf{Union} and \textbf{Intersect} are commutative.
3) \textbf{Union} and \textbf{Intersect} are distributive.
4) \textbf{Empty} and \textbf{univ} = \textbf{Complement Empty} are zeros for \textbf{Union} and \textbf{Intersect}, respectively.
5) \textbf{r `Union` Complement r} \equiv \textbf{univ} \text{ and } \textbf{r `Intersect` Complement r} \equiv \textbf{Empty}

This set of axioms captures what is called a \textit{boolean algebra}.