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.

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 \text{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

\[
\text{Rectangle } s_1 \ s_2 \ `\text{containsS}` \ (x, y) \\
= \text{let } \ t_1 = s_1/2 \\
\quad \ t_2 = s_2/2 \\
\quad \text{in } \ -t_1 \leq x \ \&\& \ x \leq t_1 \ \&\& \ -t_2 \leq y \ \&\& \ y \leq t_2
\]
Ellipse

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

$= \frac{x}{r_1}^2 + \frac{y}{r_2}^2 \leq 1$
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:

\[
\text{isLeftOf} :: \text{Coordinate} \to \text{Ray} \to \text{Bool} \\
(p_x, p_y) \ \text{isLeftOf} \ ((a_x, a_y), (b_x, b_y)) \\
= \text{let } (s, t) = (p_x-a_x, p_y-a_y) \\
\quad (u, v) = (p_x-b_x, p_y-b_y) \\
\quad \text{in } s*v \geq t*u
\]

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

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

\[
\text{RtTriangle } s_1 \ s_2 \ `\text{containsS` } \ p \\
= \text{Polygon } [(0,0),(s_1,0),(0,s_2)] \ `\text{containsS` } \ p
\]
Putting it all Together

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

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

• Note that, for any $r_1$, $r_2$, and $r_3$:
  
  \[(r_1 \cup (r_2 \cup r_3)) \text{ contains } R \text{ } \equiv \text{ iff and only if:} \]
  
  \[(r_1 \cup r_2) \cup r_3) \text{ contains } R \equiv p\]

  which we can abbreviate as:
  
  \[(r_1 \cup (r_2 \cup r_3)) \equiv ((r_1 \cup r_2) \cup r_3)\]

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

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

\[(r_1 \text{ `Union` } (r_2 \text{ `Union` } r_3)) \text{ `containsR` } p\]
\[= (r_1 \text{ `containsR` } p) \, || \, ((r_2 \text{ `Union` } r_3) \text{ `containsR` } p)\]
\[= (r_1 \text{ `containsR` } p) \, || \, ((r_2 \text{ `containsR` } p) \, || \, (r_3 \text{ `containsR` } p))\]
\[= (((r_1 \text{ `containsR` } p) \, || \, (r_2 \text{ `containsR` } p)) \, || \, (r_3 \text{ `containsR` } p))\]
\[= ((r_1 \text{ `Union` } r_2) \text{ `containsR` } p) \, || \, (r_3 \text{ `containsR` } p)\]
\[= ((r_1 \text{ `Union` } r_2) \text{ `Union` } r_3) \text{ `containsR` } p\]

(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) $\text{Union}$ and $\text{Intersect}$ are associative.

2) $\text{Union}$ and $\text{Intersect}$ are commutative.

3) $\text{Union}$ and $\text{Intersect}$ are distributive.

4) $\text{Empty}$ and $\text{univ} = \text{Complement} \ \text{Empty}$ are zeros for $\text{Union}$ and $\text{Intersect}$, respectively.

5) $r \ \text{Union} \ \text{Complement} \ r \equiv \text{univ}$ and $r \ \text{Intersect} \ \text{Complement} \ r \equiv \text{Empty}$

This set of axioms captures what is called a *boolean algebra*. 