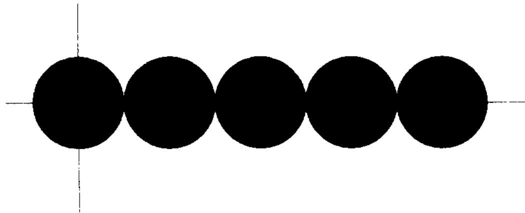
Exercise 7.5 Enhance the *Expr* data type with *variables* and *let expressions,* similar in intent to Haskell's variables and let expressions, although you may assume that the let expression does not allow recursive definitions. Also enhance the *evaluate* function to yield the proper value. For example:

evaluate (*Let* "x" (*C* 5) (*V* "x" :+ *V* "x"))
$\Rightarrow$ 10

where *Let* and *V* are the new constructors in the *Expr* data type. Unbound variables should be treated as errors.



ɔol

Figure 8.2: Five Unit Circles

Exercise 8.1 Define a function that creates five circles (of the same size and orientation shown in Fig. 8.2) by intersecting a region containing an infinite sequence of circles with a suitably sized and positioned rectangle. What are the trade-offs in using this kind of definition compared to *fiveCircles* defined above?

Exercise 8.3 Define a function *annulus* :: *Radius* → *Radius* → *Region* that creates an annulus, or "donut," whose inner radius is the first argument, and outer is the second.

Exercise 9.2 Show that *flip* (*flip f*) is the same as *f*.

Exercise 9.9 Suppose we define a function *fix* as:

fix f = *f* (*fix f*)

What is the principal type of *fix*? (This is tricky!) Suppose further that we have a recursive function:

remainder :: *Integer* → *Integer* → *Integer*
remainder a b = **if** *a* < *b* **then** *a*
          else *remainder* (*a* − *b*) *b*

Rewrite this function using *fix* so that it is not recursive. (Also tricky!) Do you think that this process can be applied to *any* recursive function?

Exercise 9.10 Rewrite this example:

map (\x → (x + 1)/2) *xs*

using a composition of sections.

Exercise 9.11 Consider the expression:

map f (*map g xs*)

Rewrite this using function composition and a single call to *map*. Then rewrite the earlier example:

꞉o