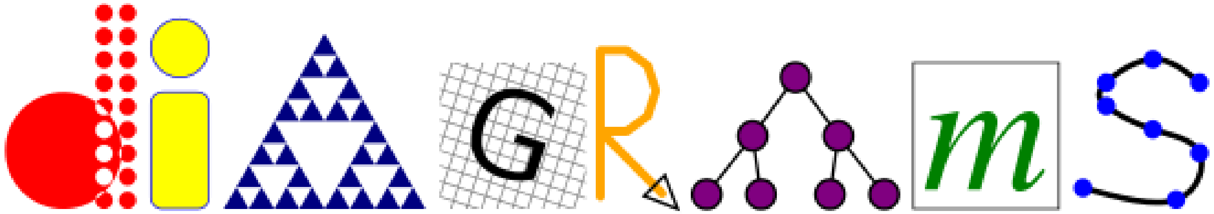
5. Extended example: Diagrams

Embedded DSL for vector graphics.

`http://projects.haskell.org/diagrams/`



We'll build a simpler language in the same style.
(We'll borrow `diagrams` functionality for SVG output.)

5.1. Shapes

Deep embedding:

```
data Shape
    = Rectangle Double Double   -- width, height
    | Ellipse Double Double     -- xradius, yradius
    | Triangle Double           -- side length (equilateral)
```

Not very exciting, because not recursive.

5.2. Styles

```
type StyleSheet = [Styling]
data Styling
    = FillColour Col
    | StrokeColour Col
    | StrokeWidth Double

red, blue, green, yellow, brown, black ... :: Col
```

Default is for no fill, and very thin black strokes.

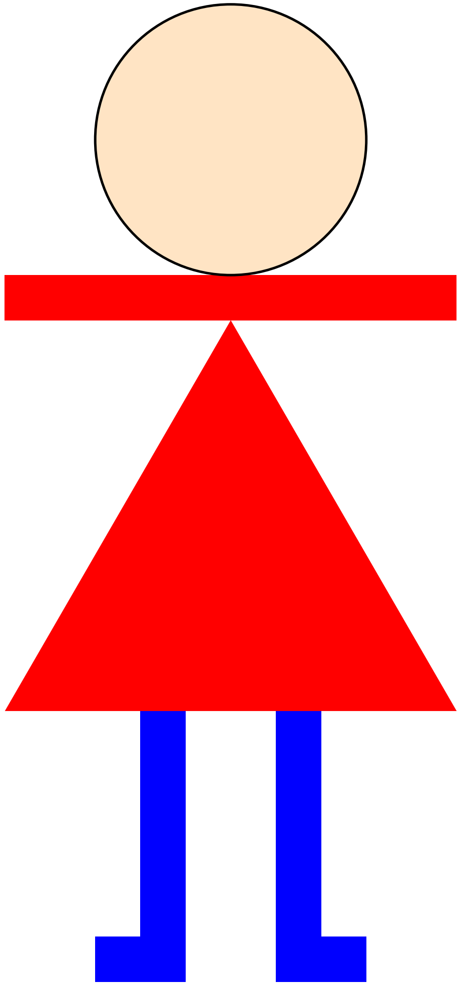
5.3. Pictures

```
data Picture
   = Place StyleSheet Shape
   | Above Picture Picture
   | Beside Picture Picture
```

Alignment is by centres.

5.4. Red dress and blue stockings

```
figure :: Picture
figure =
  Place [StrokeWidth 0.1, FillColour bisque]
          (Ellipse 3 3) ‘Above‘
  Place [FillColour red, StrokeWidth 0]
          (Rectangle 10 1) ‘Above‘
  Place [...] (Triangle 10) ‘Above‘
  (Place [...] (Rectangle 1 5) ‘Beside‘
   Place [StrokeWidth 0] (Rectangle 2 5) ‘Beside‘
   Place [...] (Rectangle 1 5)) ‘Above‘
  (Place ... ‘Beside‘ ...)
```



(Note blank rectangle.)

5.5. Transformations

To align pictures, we'll need to translate them.

```
type Pos = Complex Double

data Transform
  = Identity
  | Translate Pos
  | Compose Transform Transform
```

We represent 2D point (x, y) by Haskell $(x :+ y) :: Complex\ Double$.

```
transformPos :: Transform → Pos → Pos
transformPos Identity        = id
transformPos (Translate p)   = (p+)
transformPos (Compose t u)   = transformPos t ∘ transformPos u
```

This is a deep embedding. How about shallow?

5.6. Simplified pictures

$$\mathbf{type}\ \mathit{Drawing} = [\,(\mathit{Transform}, \mathit{StyleSheet}, \mathit{Shape})\,] \quad \text{-- centred on origin}$$

$$\mathbf{type}\ \mathit{Extent} = (\mathit{Pos}, \mathit{Pos})$$

$$\begin{aligned}
&\mathit{unionExtent} :: \mathit{Extent} \rightarrow \mathit{Extent} \rightarrow \mathit{Extent}\\
&\mathit{unionExtent}\ (\mathit{llx}_1 :\!+ \mathit{lly}_1, \mathit{urx}_1 :\!+ \mathit{ury}_1)\ (\mathit{llx}_2 :\!+ \mathit{lly}_2, \mathit{urx}_2 :\!+ \mathit{ury}_2)\\
&\quad = (\mathit{min}\ \mathit{llx}_1\ \mathit{llx}_2 :\!+ \mathit{min}\ \mathit{lly}_1\ \mathit{lly}_2, \mathit{max}\ \mathit{urx}_1\ \mathit{urx}_2 :\!+ \mathit{max}\ \mathit{ury}_1\ \mathit{ury}_2)
\end{aligned}$$

$$\begin{aligned}
&\mathit{shapeExtent} :: \mathit{Shape} \rightarrow \mathit{Extent}\\
&\mathit{shapeExtent}\ (\mathit{Ellipse}\ \mathit{xr}\ \mathit{yr}) &&= (-(\mathit{xr} :\!+ \mathit{yr}), \mathit{xr} :\!+ \mathit{yr})\\
&\mathit{shapeExtent}\ (\mathit{Rectangle}\ w\ h) &&= (-({}^{w}/_{2} :\!+ {}^{h}/_{2}), {}^{w}/_{2} :\!+ {}^{h}/_{2})\\
&\mathit{shapeExtent}\ (\mathit{Triangle}\ s) &&= (-({}^{s}/_{2} :\!+ \sqrt{3} \times {}^{s}/_{4}), {}^{s}/_{2} :\!+ \sqrt{3} \times {}^{s}/_{4})
\end{aligned}$$

$$\begin{aligned}
&\mathit{drawingExtent} :: \mathit{Drawing} \rightarrow \mathit{Extent}\\
&\mathit{drawingExtent} = \mathit{foldr1}\ \mathit{unionExtent} \circ \mathit{map}\ \mathit{getExtent}\ \mathbf{where}\\
&\quad \mathit{getExtent}\ (t, \_, s) = \mathbf{let}\ (\mathit{ll}, \mathit{ur}) = \mathit{shapeExtent}\ s\\
&\qquad\qquad\qquad\qquad\quad\ \mathbf{in}\ (\mathit{transformPos}\ t\ \mathit{ll}, \mathit{transformPos}\ t\ \mathit{ur})
\end{aligned}$$

5.7. Simplifying pictures

```
drawPicture :: Picture → Drawing
drawPicture (Place u s)    = drawShape u s
drawPicture (Above p q)  = drawPicture p ‘aboveD‘ drawPicture q
drawPicture (Beside p q) = drawPicture p ‘besideD‘ drawPicture q
```

All the work is in the individual operations:

```
drawShape :: StyleSheet → Shape → Drawing
aboveD, besideD :: Drawing → Drawing → Drawing
```

5.8. Simplifying pictures

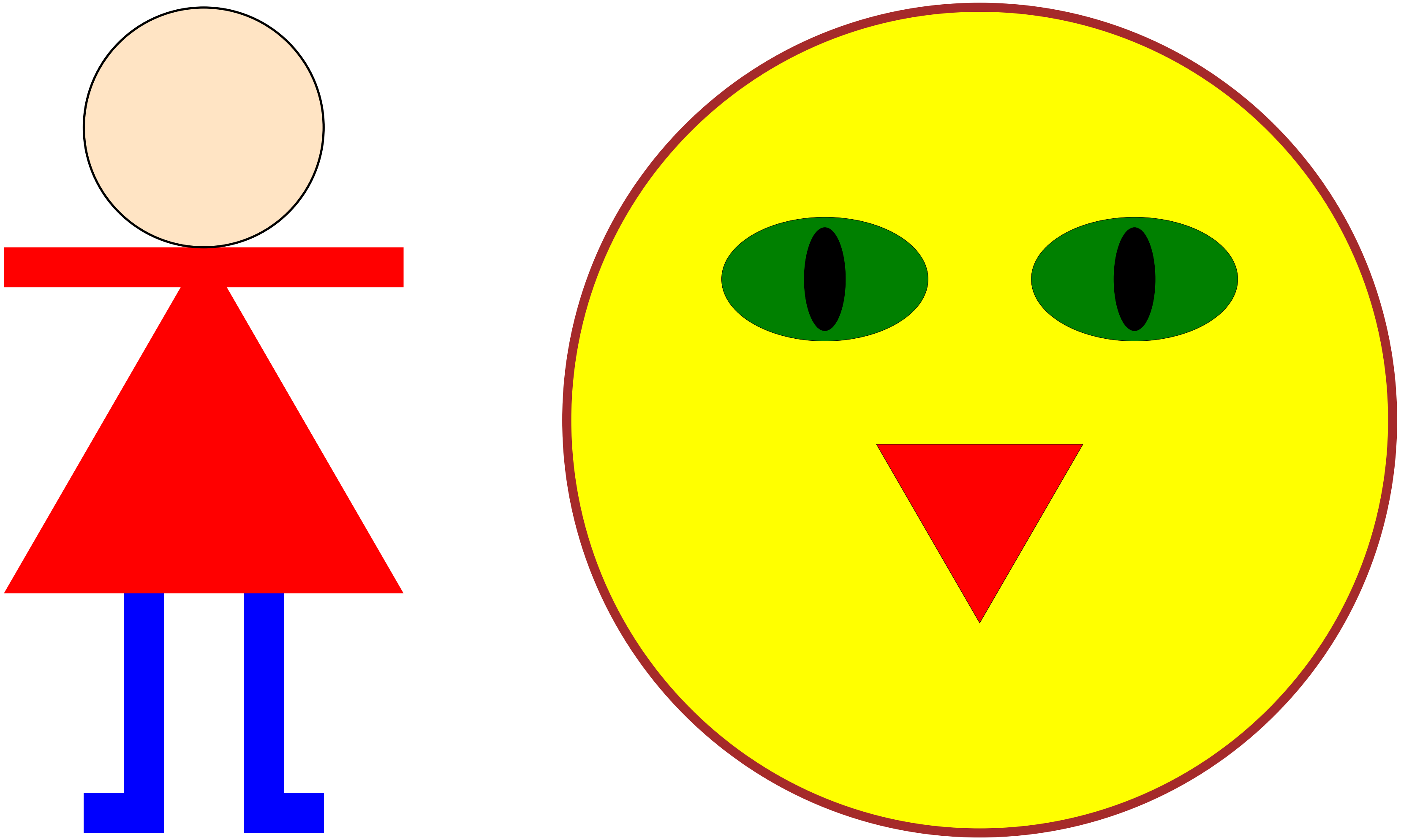
```
drawShape :: StyleSheet → Shape → Drawing
drawShape u s = [ (Identity, u, s) ]

aboveD, besideD :: Drawing → Drawing → Drawing
pd `aboveD` qd = transformDrawing (Translate (0 :+ qury)) pd ++
                 transformDrawing (Translate (0 :+ plly)) qd where
  (pllx :+ plly, pur)  = drawingExtent pd
  (qll, qurx :+ qury)  = drawingExtent qd

pd `besideD` qd = transformDrawing (Translate (qllx :+ 0)) pd ++
                  transformDrawing (Translate (purx :+ 0)) qd where
  (pll, purx :+ pury)  = drawingExtent pd
  (qllx :+ qlly, qur)  = drawingExtent qd

transformDrawing :: Transform → Drawing → Drawing
transformDrawing t = map (λ(t′, u, s) → (Compose t t′, u, s))
```

5.9. *InFrontOf*, *FlipV*



5.10. Generating SVG

```
type DiagramSVG = …

assemble :: Drawing → DiagramSVG
assemble = foldr1 atop ∘ map draw where
  draw (t, u, s) = transformDiagram t (diagramShape u s)

atop             :: DiagramSVG → DiagramSVG → DiagramSVG
diagramShape     :: StyleSheet → Shape → DiagramSVG
transformDiagram :: Transform → DiagramSVG → DiagramSVG
writeSVG         :: FilePath → DiagramSVG → IO ()
```

5.11. Transformations again

```
transformDiagram :: Transform → DiagramSVG → DiagramSVG
transformDiagram Identity                 = id
transformDiagram (Translate (x :+ y)) = translate (r2 (x, y))
transformDiagram (Compose t u)          = transformDiagram t ∘
                                            transformDiagram u
```

Recall earlier use:

```
transformPos :: Transform → Pos → Pos
transformPos Identity          = id
transformPos (Translate p)   = (p+)
transformPos (Compose t u) = transformPos t ∘ transformPos u
```

Shallow embedding with two observers?
Parametrized observer? Polymorphic observer?

5.12. Tiles

Extend *Shape* language with marked tiles:

```
type Markings = [[Pos]]
data Picture = ... | Tile Double Markings
```

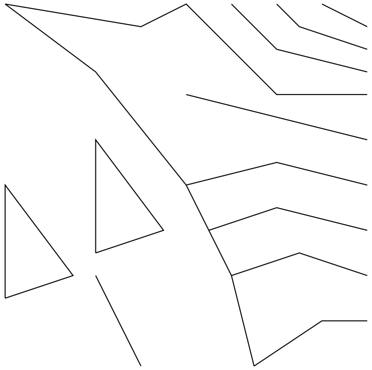
and *Transform* language with scaling and quarter-turns:

```
data Transform = ... | Expand Double Picture | Rot Picture
```

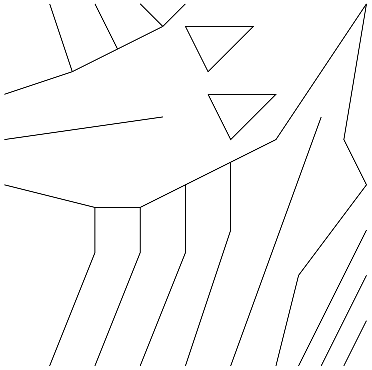
Some markings:

```
markingsP :: [[Pos]]
markingsP = [[(4 :+ 4), (6 :+ 0)],
             [(0 :+ 3), (3 :+ 4), (0 :+ 8), (0 :+ 3)],
             [(4 :+ 5), (7 :+ 6), (4 :+ 10), (4 :+ 5)],
             [(11 :+ 0), (10 :+ 4), (8 :+ 8), (4 :+ 13), (0 :+ 16)],
             [(11 :+ 0), (14 :+ 2), (16 :+ 2)] ...]
```

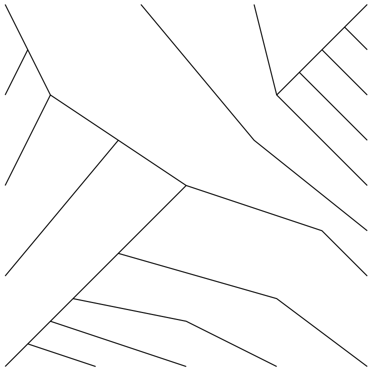
5.13. Four fish in boxes



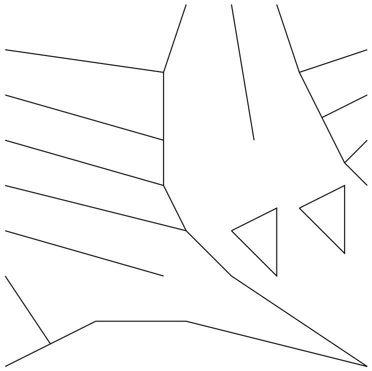
fishP



fishQ



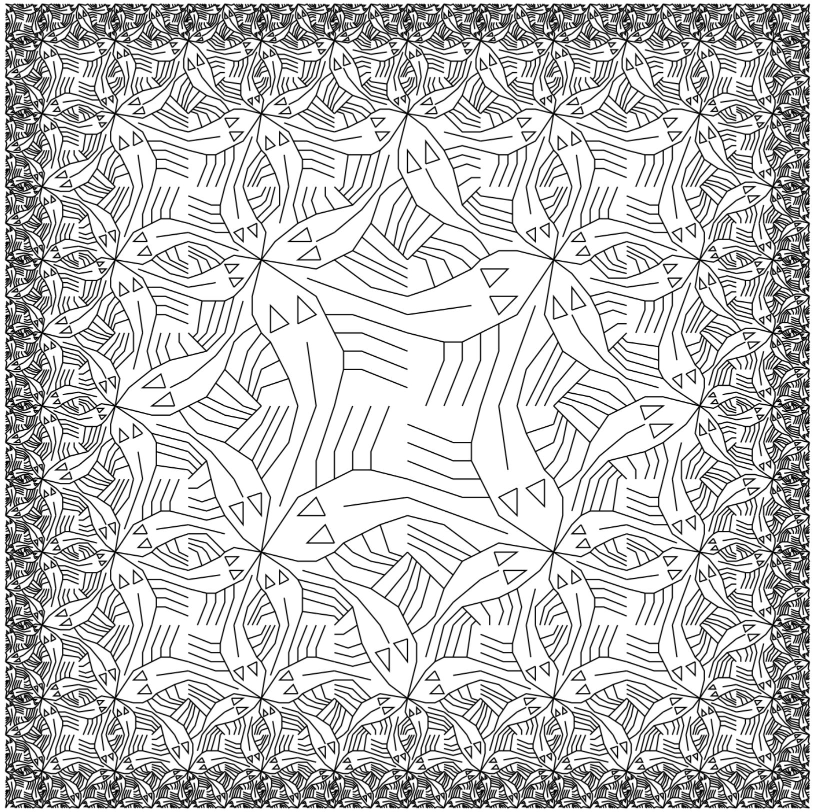
fishR

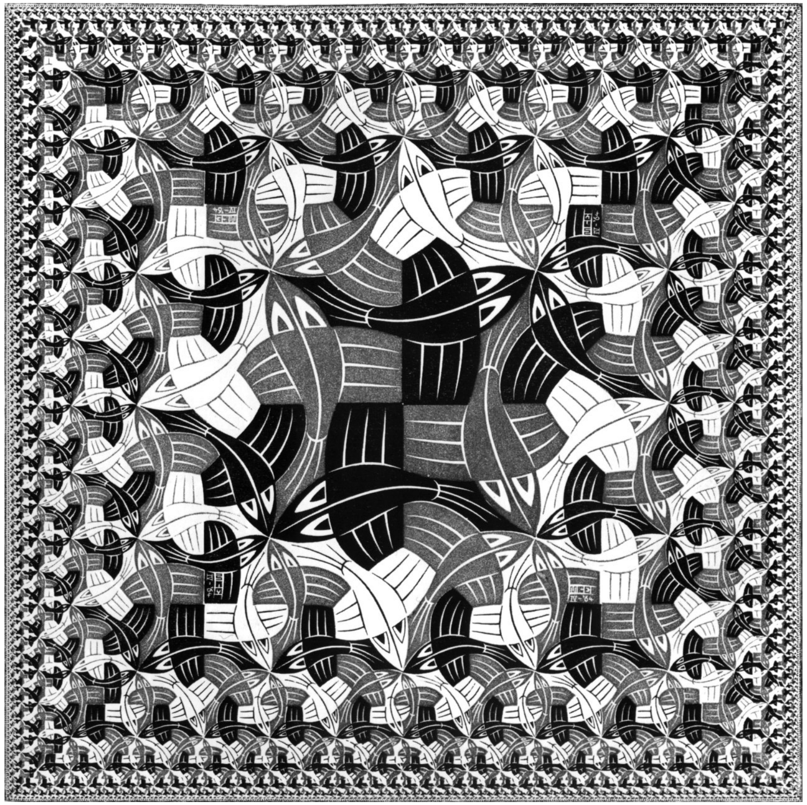


fishS

5.14. Square limit

With a little bit of scaling and rotation. . .





(After Henderson, *Functional Geometry*, 1982—after Escher, 1964.)