Advanced Programming 2025
Class-based Embedding
Assignment 10

May 15 2025

1 Goal

After making this assignment you can design and implement a class-based embedded DSL. The DSL for this assignment are geometric shapes.

The shapes that we consider can be a *circle*, the *intersection* of two shapes, the *translation* of a shape by a vector indicated by its end point, the *inversion* of a shape, or the *scaling* of a shape. The function `inside` checks if a point is inside such a shape. We can implement this by the definitions below (these definitions can be found in the file `shapeStart.icl` in Brightspace):

```
:: Point = {x :: Real, y :: Real}
:: Shape
 = Circle | Intersection Shape Shape | Trans Point Shape | Invert Shape | Scale Real Shape

instance + Point where (+) p q = {x = p.x + q.x, y = p.y + q.y}
instance - Point where (-) p q = {x = p.x - q.x, y = p.y - q.y}
instance toString Point where toString p = "{x=" <+ p.x <+ ",␣y=" <+ p.y <+ "}"
instance * Shape where (*) x y = Intersection x y
instance * Bool  where (*) x y = x && y
instance ~ Shape where ~ s = Invert s
instance ~ Bool  where ~ b = not b

inside :: Point Shape -> Bool
inside point shape
  = case shape of
      Circle          = point.x * point.x + point.y * point.y <= 1.0
      Scale r shape   = inside {x = point.x / r, y = point.y / r} shape
      Trans t shape   = inside (point - t) shape
      Invert  shape   = not (inside point shape)
      Intersection x y = inside point x && inside point y
```

The running example in this assignment is the following expression.

```
e_plain :: Bool
e_plain
  = let p0   = {x=6.0, y=0.0} in
    let p1   = {x=3.0, y=0.0} in
    let disc = Scale 5.0 Circle in
    let lens = disc * (Trans p1 disc) in
    ~ (inside p0 lens) * inside p0 (~ lens)
```

Evaluation of this expression should yield `True`.

2 Assignment

2.1 DSL design

The given embedding of the shape DSL in Clean works fine until we need multiple views. Design a class-based variant of the DSL that allows multiple views and is type-safe. The changes to the running example should be minimal. You can reuse the types `Point` and `Shape`. You mainly have to add a `lit` to lift constants to the DSL and have to replace the **`let`** definitions by your own definition primitive.

2.2 Evaluation

Implement an evaluation view of your DSL. We suggest that you use the given type `:: E a = E a` for this. Check that evaluation of the adapted example yields the correct result. You can implement the Monad machinery for `E`, but that might be a little overkill.

2.3 Counting circles

Implement a view that counts the number of primitive circles in an expression. For the running example this should be just 1, inside the definition of `disc`.

2.4 Printing

Implement a printing view for this DSL. We suggest that you use the given definitions as a starting point. This will be similar to what we have seen before.

Deadline

To receive feedback, hand in your solution before May 21 23:59h.