Project 1 - Functional Pearls - Drawing Trees

June 9, 2022

Abby Audet
s212544

Johan Raunkjær Ott
s032060

Jinsong Li
s202354

Martin Mårtensson
s195469

Design of aesthetic pleasant renderings of trees

Hello

Property-Based Testing: Validation of rendering properties

In this section we describe how we test our implementation of designing aesthetically pleasant renderings of trees. We will describe the use of property based testing (PBT) for validating the four aesthetic rules described in [Functional Pearls: Drawing Trees]. Specifically, we use FsCheck.NUnit for integrating the FsCheck PBT tool into a unit testing framework. In separate subsections, we describe the four different aesthetic rules of the paper and specify how these rules can be described as boolean properties to be tested by FsCheck. Lastly, we analyze the notion of correctness as described in the paper and show how the correctness properties are tested, but first, we briefly describe how property based testing works with the simple case of the 'mean' function.

Simple case - the mean function

PBT concerns describing a property of a feature that should hold for all input and then test this for random input in order to ensure that the property holds for the implementation of the feature. The process is thus 1) Write a boolean function that describes the property 2) Use the FsCheck tool to create random input for the property 3) Run the test that includes the boolean function using the input generated by the FsCheck Tool. When using NUnit for testing purposes, we use the FsCheck.NUnit package that includes the attribute that should be added to the property based tests.

Let us consider the simple example of the mean function implemented as

```
let mean (x: float, y: float) : float =
    (x+y)/2.0
```

There are multiple properties that could be tested for this such as bounding properties (e.g. 'mean (x, y) =< max (x, y)' and 'mean (x, y) >= min (x, y)') and the symmetry property ('mean (x, y) = mean (y, x)'). For simplicity, we only implemented the symmetry property as

```
open FsCheck
let nf = NormalFloat.op_Explicit
let meanSymmetryProp (a,b) =
    mean (nf a, nf b) = mean (nf b, nf a)
```

such that the unit test is

```
open FsCheck.NUnit
[<Property>]
let symmetryOfMeanTest () =
    meanSymmetryProp
```

Notice that in the symmetry property, the floats are cast to the FsCheck type 'NormalFloat' that removes non-normal floats (e.g. 'nan' and 'infinity') from the randomly generated input since e.g. 'nan=nan' would return 'false'.

With this we have shown a simple example on how to use PBT and some of the pitfalls of using FsCheck. Next we use PBT to validate the implementation of the aesthetic rules that the tree design should obey.

Rule 1

'Two nodes at the same level should be placed at least a given distance apart.'

Rule 2

'A parent should be centred over its offspring.'

Rule 3

'Tree drawings should be symmetrical with respect to reflection—a tree and its mirror image should produce drawings that are reflections of each other. In particular, this means that symmetric trees will be rendered symmetrically. So, for example, Figure 1 shows two renderings, the first bad, the second good.'

Rule 4

'Identical subtrees should be rendered identically—their position in the larger tree should not affect their appearance. In Figure 2 the tree on the left fails the test, and the one on the right passes.'

Correctness

Visualization of trees

To visualize the tree we will need to map the tree structure into an image.

In the case of a tree of single letters we would need the following objects for each node:

- one letter
- one line from letter to its parent (except the root node which does not have a parent)

This task can easily be done using the SVG (Scalable Vector Graphics) format

SVG

SVG files are just text files following the XML (Extensible Markup Language) format.

An SVG representation example of just the root node would look like this:

```
<svg height="300" width="600">
<text x="300" y="0" fill="black">"A"</text>
</svg>
```

And if we add a child to the root node we will also need a line between the nodes

```
<svg height="300" width="600">
<text x="300" y="0" fill="black">"A"</text>
<text x="0" y="150" fill="black">"B"</text>
<line x1="300" y1="0" x2="0" y2="150" style="stroke:rgb(0,0,0);stroke-width:2"/>
</svg>
```

The mapping

The SVG format depends on absolute values