Advanced Programming

Assignment 0

Per Steffen Czolbe, Konrad Gnoinski

Handed in: September 20, 2017

Contents

Assignment

To fulfil the assignment 0: “Look at these Curves”, a library for performing multiple actions on `Curves` has to be implemented. To fulfil the assignment 0: “Look at these Curves”, a library for performing multiple actions on `Curves` has to be implemented. A piecewise linear curve is a nonempty sequence of n points P_0, P_1, ... , P_{n-1}, connected by a sequence of linear segments: P_0P_1, $P1_1P_2$, ... , $P_{n-2}P_{n-1}$. We call P_0 the starting point, and P_{n-1} the end point of the curve.

Assessment: Ensuring Curves are never Empty

Part of the assignment is the definition of the data-type for `Curve`. A `Curve` is a set of connected `Points`. A `Curve` is never empty, it always consists of at least one point.

The constructor-function `curve :: Point -> [Point] -> Curve` mandated by the assignment guaranties that `Curve` is never empty. While the List of `Points` can be empty, the single starting `Point` passed as a parameter ensures that there is always at least one `Point` in lists created with this function.

For the definition of the data-type `Curve` multiple options exist. The following sections compares three of them.

1. A possible implementation for `Curve` would be `type Curve = [Point]`. While the list `Points` in the data-type definition could be empty, resulting in a `Curve` without a single `Point`, the constructor-function `curve :: Point -> [Point] -> Curve` still ensures that only `Curves` with at least one `Point` are created. However, this approach has the flaw that a client can just call functions of the `Curve` module by passing an empty list instead of a `Curve`, resulting in problems with empty `Curves`.

2. A better solution is defining `Curve` as `newtype Curve = Curve [Point]`. This way, an empty list passed as an argument does not qualify as a `Curve`. Clients are prevented from creating their own `Curve` via the constructor, and need to create curves via the exported constructor-function `curve :: Point -> [Point] -> Curve`, thus ensuring that clients can not create empty `Curves`.

However, this definition still allows for empty `Curves` to be created in the `Curve` Module. It relies on adherence to the programming practice of not creating empty `Curves` within the `Curve` module.

3. To enforce `Curves` with at least one `Point` even for code inside the `Curve` Module, a `Curve` could be defined as `data Curve = Point [Point]`. Now there is no possibility of a `Curve` having fever than one `Point`. However, this approach requires more overhead for coding operations working on a `Curve`. Standard functions like `map` and `fold` require a list of all `Points` to work on. Declaring the starting `Point` of a `Curve` separately from the remaining `Points` makes using these functions more cumbersome.

After consideration of these options, this implementation of the `Curve` library chose the 2nd option.

Implementation

This solution of the assignment is implemented in Haskell. This section gives an overview about how the code is organised, how to execute it and how to test it.

Files

The code is organised in multiple files:

- `src/Curves.hs`: a file containing the `Curves` module.
- `src/test/PointSpec.hs`: a file containing tests for `Point` and associated functions
- `src/test/CurveSpec.hs`: a file containing tests for `Curve` and associated functions
- `src/test/Spec.hs`: a file containing directives for the Hspec automatic test discovery

How to execute the Module

The `Curves` module can be imported into any Haskell program and executed alongside that program. Alternatively, it can be loaded into the GCHi, allowing the user to call the exported functions of the `Curves` module interactively.

Testing the Module

Next to being able to test the `Curves` module interactively, test files are provided in the `src/test` directory. The test files depend on the testing framework HSpec.

When the entire project is available, all test cases across all test files can be executed by using stack. After navigating to the project directory, execute:

```
$ stack test .
```

Alternatively, test cases of single files can be executed by opening them in GHCi and calling their main function interactively:

```
$ ghci src/test/CurveSpec.hs
*Main> :main
```

The file IO and SVG methods were tested interactively. No automated tests are implemented for these functions. Additionally, the online-TA at `find.incorrectness.dk` was used for testing the completed solution.

Test Results

The `Curves` module passed all mentioned tests.