Heinrich-Heine-University Düsseldorf
Computer Science Department
Software Engineering and Programming Languages
Philipp Körner

Functional Programming – WT 2023 / 2024
Reading Guide 6: Simplicity

Timeline: This unit should be completed by 20.11.2023.

1 Material

- Rich Hickey: Simple Made Easy `https://www.infoq.com/presentations/Simple-Made-Easy/`
- Rich Hickey: Simplicity Matters `https://www.youtube.com/watch?v=rI8tNMsozo0` (22:02 – 33:36)
- Ben Moseley, Peter Marks: Out of the Tar Pit `https://github.com/papers-we-love/papers-we-love/blob/master/design/out-of-the-tar-pit.pdf` (Sections 1 – 7)[1]
- Stuart Halloway: Simplicity Ain't Easy `https://www.youtube.com/watch?v=cidchWg74Y4` (optional)
- The Joy of Clojure, chapter 14 (optional)

2 Learning Outcomes

After completing this unit you should be able to

- explain and differentiate the terms simple, easy, hard and complex.
- recognize complexity.
- describe approaches that can be used to understand programs.
- name and identify causes of complexity.
- describe the influence of state.
- describe the differences in the approaches to complexity in functional and object-oriented programming.
- identify and differentiate essential and non-essential types of complexity.

[1] The relevant part of the article is relatively long, but easy to read.

3 Exercises

Exercise 6.1 (Fixed point algorithm)
The exercise from unit 5 asked you to implement Newton's method. Now, proceed as follows:

a) Write a function `(defn fixedpoint [F guess eps?] ...)`, which computes a fixed point of F from an initial value $guess$. The accuracy $eps?$ should itself be a function that receives two inputs: (the new and old value) and returns true if the two values match sufficiently well.

You can use the following code (or your own solution) as a start:

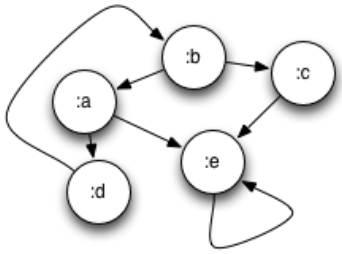
```
(defn newton [f f']
  (fn newton-fn [x eps]
    (let [next-x (- x (/ (f x)
                         (f' x)))]
      (if (< (abs (- next-x x)) eps)
        next-x
        (newton-fn next-x eps)))))
```

b) Write Newton's method (from as instance of this fixed point function.

c) Which aspects of the implementation of Newton's method have you abstracted and decomplected here?

Exercise 6.2 (Graph)
The following exercise asks you to implement different functions operating on a graph. As representation of the graph we use a map.

```
(def g {:nodes #{:a :c :b :d :e},
        :edges #{[:b :c] [:e :e]
                 [:c :e] [:a :d]
                 [:a :e] [:d :b]
                 [:b :a]}})
```



a) Write a function `(defn dom [g] ...)`, which returns a subset of nodes that have an outgoing edge.

b) Write a function `(defn ran [g] ...)`, which returns a subset of nodes that have an incoming edge.

c) Implement a function `(defn tc [g] ...)`, which takes a graph as a parameter and returns a graph describing the transitive closure of the input. *Note: Consider whether you can use the fixed point function or a generalized version of a fixed point function to do this.*

d) Implement the function `(defn trc [g] ...)`, which computes the transitive, reflexive closure.

e) Write a predicate `(defn path? [g start end] ...)`, that returns a truthy value if Graph g contains a path from $start$ to end or a falsey value if there is no such path. Make sure that your predicate also terminates if the graph is cyclic.

Questions

If you have any questions, please contact Philipp Körner (p.koerner@hhu.de) or post it to the Rocket.Chat group.