15 Nov 2018

11 An Eager Functional Language

1. Motivation.

① In Chap10, we learnt the lambda calculus and the eager evaluation for it. They form the basis of most (or nearly all) call-by-value functional programming languages such as Ocaml, Clojure, Scala, Scheme etc. But developing such a real programming language (eager, functional) involves much more than including the lambda calculus and adopting the eager evaluation. The goal of this chapter and the subsequent few chapters is to understand these additional things.

② In the chapter, we will study two topics (mainly) related to the following questions.

i) In order to help solve most real-world computational problems naturally, a functional programming language should include constants and operations for primitive types, such as Int, and mechanisms for building data structures. How can we do this? How should we change the abstract grammar, the notion of canonical forms, evaluation relation and denotational semantics?

ii) Also, we need to support recursive definitions. What should we do?

③ We will focus on answering these questions. In Chap10, Reynolds explains a lot more than what we will cover. He even gives well-known examples (list-manipulation) of functional programming. If you are interested in them, have a look at Chap10.