Summary 3 Jianfeng Jia

Recursive Programming

E. W. Dijkstra

In this paper, Dijskstra introduced the idea of Recursive Programming. The basic concept of it is using Stack to store a sequence of information units. Each of the units contains the parameters, local variables, and the intermediate results. The parameters includes the function argument, and the “link” which comprised all the data necessary for the continuation of the main program when the subroutine has been completed; The local variables are no longer of interest as soon as the subroutine finished.

The “return address” is for sure part of the “link” data. Besides that, the subroutine is in general a piece of program that takes full advantage of the flexibility. It would imply a series of changes within those unit. Therefore, we need to record sufficient data regarding to the state of the main computation in order to reconstruct it later.

By reserve the enough information, nothing forbids a subroutine A calls A again. The subroutine only has to appear in memory once, but it have more than one simultaneous state in the stack.

The essence of functional programming

This paper explores the use monads to structure functional programs. The author introduced how to use monads to mimic the effect of impure features such as exceptions, state and continuations. Formally, a monad consists of a type constructor M and two operations, *bind* and *return*. The operations must fulfill several properties to allow the correct composition of monadic functions (i.e. functions that use values from the monad as their arguments or return value). The return operation takes a value from a plain type and puts it into a container using the constructor, creating a monadic value. The bind operation performs the reverse process, extracting the original value from the container and passing it to the associated next function in the pipeline, possibly with additional checks and

transformations. From the examples given by this paper, all the “side effect” is *return* as a whole values, and then the next function need to a *bind* to decompose it to get the parameters and the state, exceptions, … etc. In this way all the function is pure.