Y-Combinator based Continuation-Passing Style Technique in Python Programming

Songphon Klabwong
songphon@rmutt.ac.th
Faculty of Science and Technology
Rajamangala University of Technology Thanyaburi
Thailand

ABSTRACT

Lambda calculus has the equivalent expressive power compared to the Turing model. It is the origin of the functional programming approach. One of the most important concepts is Y-Combinator. It enables the way to perform recursion by anonymous function. In this study, we proposed the study on Y-Combinator of typed-lambda calculus based on python language. The study shows the way implementing tail-call recursion. It employs the Continuation-Passing Style (CPS) technique to send the context of execution along with the call. As a result, the code written using CPS is guaranteed to be side-effect free. Consequently, it is efficient to be executed on concurrency environment.

Keywords: Y-Combinator, Continuation, Python, Lambda Calculus, Anonymous Recursion

INTRODUCTION

In recent years, concurrent processing environments dominate computing world by two main factors: the proliferation of multicore CPUs and asynchronous environment of internet. As a result, the programming environments of traditional approaches are no longer suitable for many applications especially on the concurrent execution environment where several processes perform their tasks separately but need of synchronization among them. Concurrent programming is one of approaches to deal with the need of concurrent environment. It comprises several programming technique. Functional programming is one of the programming techniques which employ the notion of mathematical function. One important concept of the mathematical function is that it has no side-effect on the computation i.e. whenever the input of an individual function is the same; the result is always the same.

In this study, we employ and combine two programming techniques: Y-Combinator and Continuation Passing Technique to create a general form function in Python programming language.

METHOD

Continuation-Passing Style (CPS) is the way to control the program flow via a function context. The flow of program is passed through function parameter. However, CPS needs the predefine function to work with. This imposes the limitation of usage since we could not perform the

anonymous recursion. The anonymous function definition and invocation are pervasive concept in functional programming. At this point, Y-Combinator is introduced to our study. Y-Combinator is one of the technique is functional programming that enables anonymous recursion.
We start with the Y-combinator definition. Because python programming language is call-by-

$$\lambda f.(\lambda x.x(x))(\lambda y.f(\lambda a.y(y)(a)))$$

In the Python form we can write down as:

(lambda f: lambda x(x))(lambda y:f(lambda *a: y(y)(*a))))

Note that the star symbol is to indicate the list parameter in python since we might need multiple parameters function. At this point we introduce the continuation by defining the closure function to capture an execution context. The result is:

(lambda f: lambda x(x))(lambda y:f(lambda *a: y(y)(*a))))(lambda f: lambda n,c:f(n,c))

Finally we need the seed value to start the execution of function. It can be defined as a zero-parameter function and can be written down in Python form as:

(lambda f: lambda x(x))(lambda y:f(lambda *a: y(y)(*a))))
(lambda f: lambda n,c:f(n,c))(k,lambda i:i)

where k is zero-parameter function and lambda i:I represent the identify function.

RESULT

The function of Y-Combinator based CPS is guaranteed to be side-effect free, hence , can be separated to multiple small pieces of task and concurrently process over the computing network or cluster. It also enables the anonymous way to perform task which gain more flexibility of CPS.

DISCUSSION

Even though, the study shows the flexibility of using CPS over Y-Combinator. One might concerns about its performance. We believe that the real question is that: can the multiple-concurrent processing yields the better performance compares to the overhead of task separation.

REFERENCES

Field A.J., & Harrison P.G. (1988). *Functional Programming.* Addison-Wesley.

Gabriel, R. P. (1988). *The Why of Y.*

Norvig, P. (1990). *Self-Reproducing Programs in Common Lisp.* Computer Science Division, University of California Berkeley, CA 94720.