Functional Programming Using the New C++ Standard

Article · December 2015

DOI: 10.26458/jedep.v4i4.128

Rocsana Bucea-Manea-Tonis
Spiru Haret University, Bucharest
40 PUBLICATIONS **13** CITATIONS

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 4, Issue 4, 2015

URL: http://jedep.spiruharet.ro
e-mail: office_jedep@spiruharet.ro

Functional Programming Using the New C++ Standard

Radu BUCEA-MANEA-ŢONIŞ[1], Rocsana ŢONIŞ (BUCEA-MANEA)[2]

Abstract. *In our article we demonstrate that the new C++ standard is the one preferred by economists and big software companies to implement AI and decision making algorithms. Today software technology evolves very quickly, to an old paradigm, called functional programming. This paradigm uses lambda functions ready to be used where declared instead of function pointers. The long term target is to let compilers evaluate rather than execute a program/function. The new C++14 standard allows lambda calculus as we demonstrate in the applicative section of implementing conditionals, booleans and numbers.*

Keywords: lambda, combinator, predicate, stochastic neoclassical growth model.

JEL Codes: M15

1. Introduction

As Phil Johnson stated, *economists have to write code in order to run and evaluate their complex mathematical models and simulations*. They want to improve their functional skills by understanding lambda calculus and improve de efficiency of programs by making them semantically transparent. A common approach to improve semantics of a programming language involves a translation of a given language into a language that represents labels and jumps as functions [Komendantsky, 2009]. Semantics gives meaning to a language, by assigning mathematical objects as values to its terms, that might be easily done using the new C++14 standard.

Extending functionality using callback functions is one of the goals of functional programming.[Pouliasis, 2014] After a literature review, in articles and archives available on international databases, such as Web of Science, EBSCOhost, IS journals and IS conference proceedings we found out that in near future might be possible to let compiler evaluate rather than execute a program. The evaluation may involve making calls to external resources and eventually receiving answers from such calls, so that the internal evaluation can proceed [AlTurki, 2015]. In our article we demonstrate how the newly C++14 standard can be used in functional programming by implementing conditionals, booleans and numbers.

[1] PhD., Coordinator for Open source MySQL Browser for Windows https://mysqlbrowser.codeplex.com/, Stefanini Romania, e-mail: radub_m@yahoo.com

[2] Spiru Haret university, PhD Candidate, University Politehnica of Bucharest, Romania, e-mail: rocsense39@yahoo.com

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 4, Issue 4, 2015

URL: http://jedep.spiruharet.ro
e-mail: office_jedep@spiruharet.ro

Implementing the stochastic neoclassical growth model, the workhorse of modern macroeconomics in C++11, results that *C++ compilers have advanced enough that, contrary to the situation in the 1990s and some folk wisdom, C++ code runs slightly faster (5-7 percent) than Fortran code.* [Borağan, 2014]

After [Hughes, 1990], using higher-order functions and lazy evaluation one can modularize programs in new and useful ways, gluing functions together, gluing programs together, and thus making functional programming languages appropriate for building AI applications.

D.Syme found that functional programming is simple, efficient and fun to work with, making worth by Microsoft to invest in F#, a programming language influenced by C# and OCAML, having orthogonal & unified constructs [Syme, 2008]

2. Research methodology:

The research methodology consists of the literature study with the focus on the C++14 standard that allow implementation of lambda calculus. Our study has 3 steps:

1. Choosing databases to search articles, archives and scratch. In this respect we have chosen Web of Science, EBSCOhost, IS journals and IS conference proceedings;
2. Screening papers and extracting information that might be transformed in knowledge. There have been chosen especially new articles. For the selection we asked several keywords, such as: *lambda calculus, functional programming*, etc.;
3. Using C++14 standard we made small implementations of lambda calculus, such as combinators, conditionals, booleans and numbers.

3. Literature review

In the nineteen century we assist at the reborn of logic at such magnitude that no one had seen it before for 2000 years. First steps where done by G. Boole (1815-1864) that define a new branch of mathematics based on a couple of truth functions (future operators) defined on a set made by only two values: true and false (0 and 1). [Rojas, 1998]

Further on, important steps were made in formal logic starting with G.Frege (1848-1925) proposing predicate logic to reveal underneath layers of propositional logic through analytic methods, continuing with B.Russel (1872-1970) and A.N.Whitehead (1861-1947) trying to set mathematics on logical based ground (Principia Mathematica).

Another valuable direction in logic is combinatory logic with the contributions of M.I. Schönfinkel (1889-1942) that introduced logical combinators (e.g. I, K, S) that could be used to represent any logical expression and H.Curry (1900-1982) credited for the currying method used for the transformation of logical expressions.

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 4, Issue 4, 2015

URL: http://jedep.spiruharet.ro
e-mail: office_jedep@spiruharet.ro

G. Boole (1815-1864)

M.I. Schönfinkel (1889-1942)

A.Church(1903-1995)

After the cold shower provided by the Russel Paradox and K.Gödel(1906-1978) Incompleteness Theorem, A.Church(1903-1995) proposed lambda calculus as an alternative computing model for A.Turing (1912-1954) Automata, that served as base for the next generation of functional programming languages.

4. Lambda calculus

Lambda calculus is a theory of computable functions that focuses on evaluation of expressions resulted from applying functions to other expressions. An expression could be made from another function (abstraction), bound/unbound variables and it is evaluated from left to right. Not everything that looks like a function fits obviously into the λ-calculus; examples include metavariables, capturing substitution, and functions depending on intensional properties like free variables [Gabbay, 2009]. There are only two operators: λ that precedes an abstraction and. (dot) operator that precedes any expressions the function is applied to. Round parentheses are used for grouping terms but they have no meaning. There is typed and un-typed lambda calculus but we prefer the first for the sake of simplicity.

One example of lambda expression is the following:

$\lambda x.\lambda y.\lambda z.x\ y\ z$ (1)

, and using curried notation, we obtain:

$\lambda xyz.x\ y\ z$ (2)

4.1. Combinators

There are few combinators that help us write more concise lambda expressions:

1. Identity combinator:

$I = \lambda x.x \rightarrow x$ (3)

2. Constant combinator:

$K = \lambda x.k \rightarrow k$ (4)

3. The substitution combinator:

$S = \lambda xyz.xz(zy)$ (5)

4.2 Conditionals

For introducing a condition we introduce the pair structure which takes two false/true expressions and one predicate function that says what value to return: the first, or the second [Kieras, 2015]:

λx.λy.λf(f x y) (6)

, where

TRUE ← λx.λy.x (7)

, and

FALSE ← λx.λy.y (8)

One possible C++ implementation of lambda conditional function will be:

auto IF = [] (auto f, int a, int b) {return f(a,b);}; (9)

, where the predicate function may be:

auto TRUE= [] (int x,int y) {return x;}; (10)

, and

auto FALSE= [] (int x,int y) {return y;}; (11)

In C++11, lambda function parameters need to be declared with concrete types. C++14 relaxes this requirement, allowing lambda function parameters to be declared with the auto type specifier.

For example, the line:

std::cout<<IF(TRUE,2,1); (12)

, will output 2, and the line:

std::cout<<IF(FALSE,2,1); (13)

, will output 1.

C++14 allows captured members to be initialized with arbitrary expressions. This allows both capture by value-move and declaring arbitrary members of the lambda, without having a correspondingly named variable in an outer scope [Sutter, 2013].

4.3 Booleans

Pair structure could be employed to represent logical operators like NOT, AND, OR: [Eberl, 2011]

NOT ← λx.(IF FALSE TRUE x)
AND ← λx.λy.(IF y FALSE x) (14)
OR ← λx.λy.(IF TRUE y x)

One implementation of these basic logical functions will be:

auto OR = [&] (int x, int y) {if (TRUE(x,y)==0) return FALSE(x,y); else return TRUE(x,y);};
auto AND = [&] (int x, int y) {if (TRUE(x,y)==0) return TRUE(x,y); else return FALSE(x,y);};
auto NOT = [&] (auto f, int x, int y) {if(f(x, y)==1) return FALSE(x,y); else return TRUE(x,y);}; (15)

Reference all (&) option allows using predicate named functions, and turns the expressions in more readable form.

The result of calling OR lambda function is:

std::cout<<OR(1,0); // 1 (16)

, calling AND lambda function is:

std::cout<<AND(1,0);//0 (17)

,and calling NOT lambda function is:

std::cout<<NOT(TRUE,2,1);//1 (18)

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 4, Issue 4, 2015

URL: http://jedep.spiruharet.ro
e-mail: office_jedep@spiruharet.ro

4.4 Numbers

There is no explicit value for numbers in lambda calculus. The value of a number equals the number of times a function applies to itself, like f ○ f ○ f ○ f … or simply put: *f(f(f(f(…))))*[Goldberg, 2014]

The Y combinator allows infinite recursion by applying to itself an indefinite number of times:

Y ← λf.(λx.f(x x))(λx.f(x x)) (18)

Y f ← f (Y f) ≡ f (f (Y f)) ≡ f (f (f(Y f))) ≡…

One way to use recursion finding the nth prime number in C++ using lambda functions is the following:

*auto fact = [] (auto f, int n) {if (n==1) return 1; else return n*f(f,n-1);};* (19)

The result of calling the line

std::cout<<fact(fact,3); (20)

, will output 6, the (n-1) argument could be replaced by predecessor function PRED (n), like in the next example:

auto PRED = [] (int x) {if (x>0) return x-1; else return 0;}; (21)

, and the final form of fact expression will be:

*auto fact = [&] (auto f, int n) {if (n==1) return 1; else return n*f(f,PRED(n));};* (22)

6. Conclusions

The C++14 standard gets lambda calculus to new expressive power, namely a lambda expressions will work with any suitable type, implicitly deducing the return type. The new lambda easily captures by move and allows defining arbitrary new local variables in the lambda object [Vandevoorde, 2013].

Although there is quite a long road to make compiler evaluate rather than execute a program/function, now is the time to discover that C++ is not anymore just C with classes but it has changed dramatically, as [Feathers, 2010] had stated: *Object oriented programs makes code understandable by encapsulating moving parts, functional programming makes code understandable by minimizing moving parts*.

7. Acknowledgement

The work has been funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreement POSDRU/159/1.5/S/134398.

8. References

[1] V. Komendantsky, "*Denotational Semantics of Call-by-name Normalization in Lambda-mu Calculus*", Electronic Notes in Theoretical Computer Science, Vol. 225, 2 January 2009, pp. 161-179

[2] K. Pouliasis, G. Primiero, *"J-Calc: A Typed Lambda Calculus for Intuitionistic Justification Logic*" Electronic Notes in Theoretical Computer Science, Vol. 300, 21 January 2014, pp. 71-87

[3] M. A. AlTurki, J. Meseguer, *"Executable rewriting logic semantics of Orc and formal analysis of Orc programs",* Journal of Logical and Algebraic Methods in Programming, Vol. 84, Issue 4, July 2015, pp. 505-533

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 4, Issue 4, 2015

URL: http://jedep.spiruharet.ro
e-mail: office_jedep@spiruharet.ro

[4]S. Borağan Aruoba, J. Fernández-Villaverde, A Comparison of Programming Languages in Economics, NBER Working Paper No. 20263, Issued in June 2014, NBER Program(s): EFG, available on-line at: http://www.nber.org/papers/w20263

[5] J. Hughes, Why Functional Programming Matters, Research Topics in Functional Programming, ed. Addison-Wesley, 1990, pp 17–42., available on-line at https://www.cs.kent.ac.uk/people/staff/dat/miranda/whyfp90.pdf

[6] D. Syme, Why is Microsoft investing in Functional Programming?, available on-line at http://cufp.org/archive/2008/slides/SymeDon.pdf

[7] R. Rojas, *"A Tutorial Introduction to the Lambda Calculus",* FU Berlin, WS-97/98, available on-line at: http://www.inf.fu-berlin.de/lehre/WS03/alpi/lambda.pdf

[8] M. J. Gabbay, D. P. Mulligan, *"Two-level Lambda-calculus",* Electronic Notes in Theoretical Computer Science, Vol. 246, 3 August 2009, pp. 107-129

[9] D. Kieras, Using C++ Lambdas, University of Michigan, February 27, 2015, available on-line at: http://www.umich.edu/~eecs381/handouts/Lambda.pdf

[10] H. Sutter, Trip Report: ISO C++ Spring 2013 Meeting, June 14. 2013, available on-line at: https://isocpp.org/blog/2013/04/trip-report-iso-c-spring-2013-meeting

[11] M. Eberl, *"The untyped Lambda Calculus"*, August 21, 2011, available on-line at: http://home.in.tum.de/~eberlm/lambda_paper.pdf

[12] M. Goldberg, *The Lambda Calculus - Outline of Lectures*, August 29, 2014, available on-line at: http://www.little-lisper.org/website/files/lambda-calculus-tutorial.pdf

[13] D. Vandevoorde, V. Voutilainen, ISO/IEC JTC1 SC22 WG21, 17 April 2013, available on-line at: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3648.html