(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 11, Issue 3, 2022

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

A functional approach to natural language

Šemsudin A. Plojović [1] Bucea-Manea-Tonis Radu [2], Valentin Kuleto, [3] Beteringhe Adrian,[4] Milena P. Ilić [5] ,

[1] Information Technology School, s.plojovic@its.edu.rs,
[2] Hyperion University of Bucharest; radumanea34@gmail.com
[3] Information Technology School, valentin.kuleto@its.edu.rs
[4] Danubius University of Galati; adrianbeteringhe@univ-danubius.ro
[5]Information Technology School, milena.ilic@its.edu.rs

Abstract. *This article does not aim to erase the border, rather vague, between natural and artificial languages, but to offer the researcher in the field of language analysis a useful tool for their logical generation and interpretation. After processing the available literature and setting the theoretical framework, a case study was conducted concerning the application of the functional approach in natural languages, and the obtained results indicate the following conclusion: The only difference between functional and natural language is that the condition of interrupting recursive calls may be external to the calling auto function in the case of natural language. Basically, once the message is understood or transmitted, it leaves the loop, this aspect being part of the communication protocol between people.*

Keywords: programming language, natural languages, functional approach

JEL Codes: L86

How to cite: Bucea-Manea-Țoniş, R., Kuleto, V., Beteringhe, A., Ilić, M., & Plojović, Š. (2022). A FUNCTIONAL APPROACH TO NATURAL LANGUAGE. *Journal of Economic Development, Environment and People, 11*(3), 36-44. doi:http://dx.doi.org/10.26458/jedep.v11i3.773

1. Introduction

This article does not aim to erase the border, rather vague, between natural and artificial languages, but to offer the researcher in the field of language analysis a useful tool for their logical generation and interpretation. Programming languages are formal, artificial languages, created to develop, implement, and verify software. There is an ever-present dilemma about the type of language that should be used when programming certain functionality in software development. Natural languages are easier for developers to use, but they lack precision and wide application. On the other hand, formal languages focus on the details

+ Corresponding author. Tel.: +381603907000; fax: +38120385822. *E-mail address*: semsudin.plojovic@its.edu.rs .

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 11, Issue 3, 2022

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

of the software, but they are difficult so developers know them less and apply them more difficult. That is why in this paper the authors analyze the differences in the application of natural and functional languages through the example of processing abstract syntax trees.

It is also necessary to pay attention to the complexity of the code that needs to be typed in order to perform the functionality of the software. If the language used allows a simpler construction of the command, with fewer lines of code and, in total, fewer individual commands, then this allows for faster code creation with less possibility of error, while at the same time relieving resources. Based on the aforementioned differences, the authors in this paper deal with the analysis of the application of functional and natural programming languages and the dilemma of application to cases.

2. Literature analysis

The functional approach to natural languages was and still is a very interesting topic among authors in the field of programming process optimization. Liu, X., & Wu, D. (Liu, X., & Wu, D., 2018) wrote a whole chapter on connections and similarities between functional and natural programming languages and the possibilities of applications of natural programming languages. These authors emphasize the changing environment as well as the fact that the possibilities of applying natural languages are developing more and more. Especially if you take into account the assistance of artificial intelligence in programming. Ernest (Ernest M.D., 2017) refers to possibilities of Natural Language Processing in software development and states the differences between the development of natural languages in programming and Natural language processing trends. On the other side, Lee (Lee K.D., 2011) is investigating natural languages in the scope of Event Driven Programming. Pane with colleagues (Pane J.F et all, 2002) did a whole study about new techniques to find more usable programming systems. Sarkar and his team (Sarkar et all, 2014) were searching for a simple interaction technique for exploratory data modeling by end users. Liu and Wu (Liu, X., & Wu, D. 2018) conducted research on trends in expansion from Natural Language to Programming Language.

3. Background

Functional programming languages are based on the concept of mathematical functions (Liu, X., & Wu, D. 2018). The main goal of functional languages is to imitate mathematical functions and a result is an approach to programming that is fundamentally different from imperative programming. The basic abstraction in imperative programming languages is abstraction flow control, in Object Oriented languages it is an object, and in functional languages, it is a function. If we want to have complete transparency of references, we must not allow any side effects. In practice, it is very difficult because some algorithms are fundamentally different based on a change of state (eg random), while some functions exist only because of their side effects (eg scanf, i.e. input/output functions) (Liu, X., & Wu, D., 2018). Depending on the presence of imperative features, there is a division between purely functional languages (without side effects) and those that are not pure. (Zhenjiang H., John H., Meng W., 2015).

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 11, Issue 3, 2022

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

Natural programming languages are those of the programming paradigms that determine the programming style. This form of programming deals with working with objects. We are talking about program elements that have their own states and behavior, i.e. each of the objects has some characteristics that define it. This practically means that program elements are represented through objects and that the program is realized through changing the state of objects. (Lee (Lee K.D., 2011)

When programming in one of the object-oriented languages, the focus is on classes. It is about defining the attributes of the object and its behavior in relation to the states in which it is located (Ernest M.D., 2017). Every time you want to create an object, the program will first look at the class. One of the main advantages of this programming approach is that it allows developers to create modules that do not need to change when a new type of object is added. In that case, the programmer simply creates a new object that inherits the properties of the existing object. Because of this, object-oriented programs are easy to change.

Native programming languages are primarily intended for people, that is, programmers so that they can more easily transfer their thoughts into computer commands. In this case, it is necessary to translate the native language into machine language and then compile the application. it is an additional complication as well as additional processes (Pane J.F et all, 2002). In native languages, the code looks like it's simpler, and essentially additional processing operations are performed. That is why the authors analyze the comparison of these two approaches in order to see some of the basic advantages and disadvantages (Sarkar et all, 2014).

The examples were described using an experimental HolyC language, written by Terry A. Davis, which meets several conditions necessary for the extended use of language in a functional sense, accord. to [1]:

1.functions that are invoked without arguments (or without overriding any default arguments), for eg.:

```
I32 foo(I32 a=2, I32=3){
return a+b;}
I32 x;
x=foo;//5
```

2.it’s a-okay to have default args at any point in the function definition, for eg.:

```
U0 CopyTo(char *source="T:/Home/File1", char *dest) {
// ... }
CopyTo(,"T:/Home/File2");
```

3.functions can have variable argument counts, specified with (...) in the function definition, for eg.:

```
U0  foo(…){
U32 i, sum;
sum=0;
```

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 11, Issue 3, 2022

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

for (i=0;i<argc;i++){

sum+=argv[i];

“foo= %d\n”, sum;}

foo(2,3,4);//foo=9

4. Content

In this part of the paper, we are presenting theoretical frames that our study is based on. A theoretic framework has consisted of these two Theorems:

Theorem 1. A statement is a function that satisfies at least one of the following conditions:

1.1. It has a potentially infinite number of arguments:

U0 P(...){};//eg. P(a,b,c);

1.2. It has one or more default parameters:

U0 P(arg0=””, …){};//eg. P(,d,e,f);

1.3. Presents a logical order of arguments, eg:

U0 P(s, v, o){}.

Theorem 2. A discourse approached as a chain with significant purposes of some statements, accord. to [2] is a recursive call on an optimized stack architecture (Tail Call) in which at least the order of the parameters changes, for example:

U0 P(v,s,o){return P(s,v,o)}.

U0 foo(U32 a='i', U32 b='a'){

"T%cC/n",a;

return foo(b,a);}

foo;//Tic Tac Tic Tac…

These theorems would be foundation that our case study is based on..

5. Case study

We are using Treehugger Javascript library for processing abstract syntax trees (AST) that consist of three parts, accord. to [5]:

- A generic ASTs representation format that can be used to represent programs written in any language;
- A set of generic traversals to query, manipulate and annotate these ASTs;
- A Javascript parser with type-inference and type structure reconstructing capabilities.

We test the parser for the next AST (fig.2), following the grammar:

S -> NP, VP;

VP -> V, NP;

VP -> V, where:

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 11, Issue 3, 2022

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

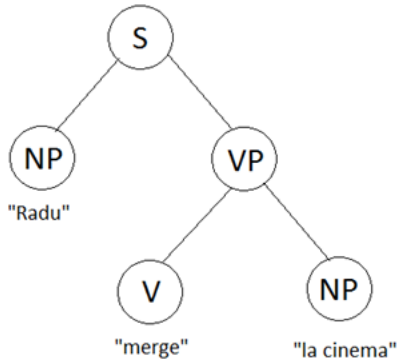

Fig.1. Parsing tree

```
let bf = function deref (a) {
    return a.n;};
var node = tree.parse('S(NP("Radu"), VP(V("merge"), NP("la cinema")))');
node.collectTopDown(
 "NP(n)", bf,
 "V(n)", bf).debug();
```

The result may be seen in the next image (fig.2):

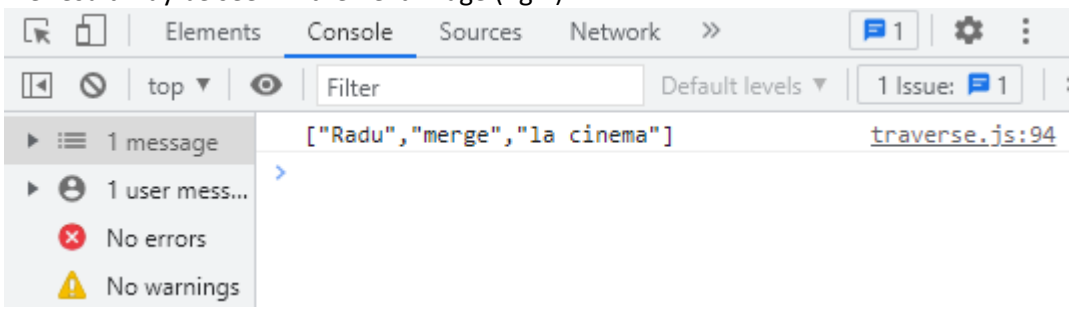

Fig.2. Chrome Javascript console

Basically, the collectTopDown method traverses the tree top to bottom until finding a match, collecting all matches and returning them as a list. At every node, we apply transformations on data that can be patterns, transformation functions, or a combination of both. When a transformation function matches, it returns a new AST node. If the transformation function returned false, the traversal parsing would have proceeded down to the last nodes.

Similar functionality can be obtained on binary arithmetic operations using the implicit holyc parser capabilities to evaluate multiple function applications using parentheses. For eg. an AST may be drawn for the following arithmetic expression: 2*3+4, see fig.3:

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 11, Issue 3, 2022

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

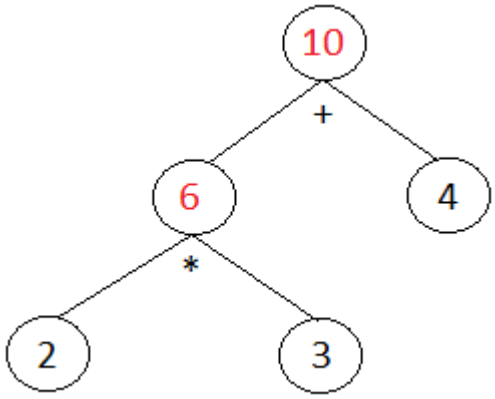

Fig 3. AST for binary arithmetic operations

```
I32 Mul(I32 a=2, I32 b=3){
   return a*b;
}
I32 Sum(I32 a, I32 b=4){
   return a+b;
}//Sum(Mul,)); =10
```

```
T:/Home>I32 Mul(I32 a=2, I32 b=3){
T:/Home>return a*b;}
T:/Home>I32 Sum(I32 a, I32 b=4){
T:/Home>return a+b;}
T:/Home>Sum(Mul,4);
0.000136s ans=0x0000000A=10
T:/Home>Sum(Mul,);
0.000055s ans=0x0000000A=10
T:/Home>
```

Fig 4. Result of applying the two functions on TempleOS Shell

In order to mock the result obtained by treehugger.js we just have to define a function for each language construct. In fact we have only two stringify functions of arity 1 and 2, but for clarity purposes we decided to create more:

```
function Det(det){
   return(det);
}
function V(v){
   return(v);
}
```

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 11, Issue 3, 2022

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

```
function N(n){
   return(n);
}
function S(np,vp){
   return np.concat(" ", vp)
}
function NP(det, n){
   return det.concat(" ", n)
}
function VP(v, np){
   return v.concat(" ", np);
}
console.log(S(NP(Det(""),N("Radu")),VP(V("poarta"),NP(Det("o"),N("caciula"))))); //Radu poarta o caciula
```

An optimize program using currying and only one binary function is the following:

```
function curry(f) { // curry(f) does the currying transform
  return function(a) {
    return function(b) {
      return f(a, b);
    };
  };
}
function sum(arg1,arg2){
   return arg1.concat(" ", arg2)
}
let S = curry(sum)
console.log(S(S("")("Radu"))(S("merge")(S("la")("cinema")))); //Radu merge la cinema
```

For a recursive solution by applying concatenating function after each word in the sentence, I recommend the following program:

```
let text = "Radu merge la cinema";
const arr = text.split(" ");
function sum(s,a){
```

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 11, Issue 3, 2022

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

```
    let str=s.concat(" ", a[0])
    if (a.shift() != undefined){
        return sum(str,a);
    }
    return console.log(str);
}
sum("",arr);//Radu merge la cinema undefined
```

6. Conclusion

After processing the available literature and setting the theoretical framework, a case study was conducted concerning the application of the functional approach in natural languages, and the obtained results indicate the following conclusion:

The only difference between functional and natural language is that the condition of interrupting recursive calls may be external to the calling auto function in the case of natural language. Basically, once the message is understood or transmitted, it leaves the loop, this aspect being part of the communication protocol between people.

Note. The examples had been tested with TempleOS [3] shell on qemu[4] emulator with the following command C:\Users\RADU\Downloads>qemu-system-x86_64 -boot d -cdrom templeoslite.iso -m 512

7. References

[1] Ernst, M.D. (2017) Natural Language is a Programming Language: Applying Natural Language Processing to Software Development, SNAPL: The 2nd Summit on Advances In Programming Languages. Editors: Ras Bodik and Shriram Krishnamurthi; Article No. 4; pp. 4:1–4:14 Leibniz International Proceedings in Informatics, Dagstuhl Publishing, Germany, DOI: 10.4230/LIPIcs.SNAPL.2017.4

[2] Green, T. R. G., & Petre, M. (1996). Usability analysis of visual programmingenvironments: A cognitive dimensions framework. Journal of Visual Languagesand Computing, 7(2), 131–174. doi:10.1006/jvlc.1996.0009

[3] Harrison, T., A Language Design Analysis of HolyC, https://harrisontotty.github.io/p/a-lang-design-analysis-of-holyc

[4] Lee, K.D. (2011). Event-Driven Programming. In: Python Programming Fundamentals. Undergraduate Topics in Computer Science. Springer, London. https://doi.org/10.1007/978-1-84996-537-8_

[5] Liu, X., & Wu, D. (2018). From Natural Language to Programming Language. In S. Goschnick (Ed.), Innovative Methods, User-Friendly Tools, Coding, and Design Approaches in People-Oriented Programming (pp. 110-130). IGI Global. https://doi.org/10.4018/978-1-5225-5969-6.ch004

[6] Pane J. F., Myers B. A. and Miller L. B., (2002) "Using HCI techniques to design a more usable programming system," Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments, 2002, pp.

(online) = ISSN 2285 – 3642
ISSN-L = 2285 – 3642
Journal of Economic Development, Environment and People
Volume 11, Issue 3, 2022

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

198-206, doi: 10.1109/HCC.2002.1046372

[7] Sarkar, A., Blackwell, A. F., Jamnik, M., & Spott, M. (2014). Teach and try: A simple interaction technique for exploratory data modeling by end users. In 2014IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)(pp. 53–56). IEEE. 10.1109/VLHCC.2014.6883022

[8] Segre, C., Istorie – cultura – critica, Ed.Univers, pg. 322

[9] Zhenjiang H., John H., Meng W., (2015) How functional programming mattered, National Science Review, Volume 2, Issue 3, September 2015, Pages 349–370, https://doi.org/10.1093/nsr/nwv042

[10] https://github.com/ajaxorg/treehugger

[11] https://www.qemu.org/

[12] https://www.templeos.org/