COMPARING LISP, PROLOG AND OPS-5 LANGUAGES WITH THE LANGUAGE JULIA

Edimar Veríssimo da Silva

INPE, ARTIFICIAL INTELLIGENCE - CAP-354
São José dos Campos, SP
yugi386@yahoo.com.br
May, 2017

ABSTRACT

This article presents LISP, PROLOG, OPS-5 and Julia as tools that can facilitate the development of applications with artificial intelligence, especially expert systems.

Key words: expert systems, lisp, prolog, ops-5, julia, artificial intelligence.

I. INTRODUCTION

In this work we present LISP, PROLOG, OPS-5 and Julia as languages indicated to create expert systems. We start with the presentation of the LISP language, based on the functional paradigm. Next we present the PROLOG language that fits the logical and declarative paradigm. Then we talk about OPS-5 and Julia, the first one based on rules or production systems and the last one is a modern language, created in 2012, multi-paradigm and focused on high performance scientific applications.

II. LISP LANGUAGE

LISP is a programming language created by John McCarthy in 1959 for symbolic processing, widely used in artificial intelligence [6]. McCarthy's motivation came from the idea of developing an algebraic language for list processing for AI (artificial intelligence) work [11]. It uses the functional programming paradigm although it supports many of the imperative language structures.

> *"LISP is the second oldest high-level programming language after Fortran and has changed a lot since its early days, and a number of dialects have existed throughout its history" [9].*

LISP is the acronym for List Processor and was originally conceived to be interpreted although currently a code written in LISP can be compiled thus bringing more performance in the execution of applications [8].

> *"The LISP language was inspired by the Lambda Calculus, a formalism developed in the 1930s by Alonzo Church. The Lambda Calculus can be considered as an abstract programming language, where the concept of Computing, that is, the ways in which functions can be combined to form other functions, appear in all its generality and in a pure form, without syntactic complications". [11]*

John McCarthy, its creator, recognized some difficulties encountered in language in those early years of development:

> *"As a programming language, the LISP had many limitations. Some of the most evident in the early 1960s were ultra-long numerical computation, the inability to represent objects by blocks of records (...) and the lack of a good system for input-output of symbolic expressions in conventional notations. All these problems and others should be corrected in the LISP 2. However, we had to make do with the LISP 1.5 developed at MIT, which corrected only the most flagrant deficiencies". [10]*

The basic types of data in LISP are atoms and lists. Atoms can be numbers, characters or symbols. A list is a sequence of atoms and/or other lists. Through atoms and lists it is possible to create very interesting applications that make use of artificial intelligence.

The software MYCIN, a pioneering system for medical diagnostics, was one of the precursors of expert systems [12] and was written in LISP language.

One of the best known applications developed with the LISP language was the software ELIZA, created by

Joseph Weizenbaum, which was the first natural language processing program in history [12]. The program shows the simulation of the conversation between a patient and his psychologist, and the software plays the role of the latter.

An example of dialogue with ELISA:

ELIZA: Please tell me your problem.
USER: I feel alone this Friday.
ELIZA: Do you often feel lonely on Friday?
USER: No. I think it might be work fatigue.
ELIZA: Aren't you just saying to be negative?
USER: I need to try something different to have fun.
ELIZA: What does trying something different to have fun mean to you?
USER: I would like to see a play.
ELIZA: Can you talk more about it?
USER: It seems that in the theater people are happier.
ELIZA: Maybe they are happier.
USER: What do you say about unhappiness?
ELIZA: Does anyone believe I told you?
USER: No. Actually I am here to understand myself better.
ELIZA: Why not?
USER: I'm afraid of the future.
ELIZA: Do you like being afraid of the future?

Figure 1: A version of ELIZA with a modern layout can be accessed at http://www.masswerk.at/eliza/.

Although not perfect, ELIZA is impressed by the logical way in which the dialogue was conducted and the impressive fact that the software was built in the mid 19601's in the early days of natural language processing. The LISP language certainly contributed a lot to the development of this software at that time.

III. PROLOG LANGUAGE

The **PROLOG** language was created in 1972 by Colmerauer and Roussel [1] and since then it has been used in specialist systems, understanding natural language, symbolic computing applications, project automation, analysis of biochemical structures and in other areas [3].

The name of the language comes from **PROgramming in LOGic** and is a specific language for the development of artificial intelligence systems as well as the LISP language [2].

> *"PROLOG is a declarative language, that is, instead of the program stipulating the way to reach the solution step by step, as it happens in procedural or object-oriented languages, it provides a description of the problem to be computed using a collection of facts and rules (logic) that indicate how the proposed problem should be solved". [3]*

The PROLOG language is based on a structure that basically involves 3 concepts:

1. Facts: represent the information that forms the knowledge base. They are the "axioms" of the system. A fact can be any information like: "17 is a prime number" or "the sum of the inner angles of a triangle is 180º" or "Mary is Julia's sister". A fact is always true.

2. Rules: They define the conditions that must be met for a certain statement to be considered true [4]. A rule is divided into two parts: on the left we have the conclusion and on the right we have the condition for the conclusion to be true [2]. Rules specify something that "can be true if some conditions are met" [4].

3. Questions or Queries: Allow you to search the knowledge base for answers among all possible solutions [2]. "The PROLOG system accepts the facts and rules as a set of axioms and the user's query as a theorem to be proved". [4]

"The main motivation for using logical programming is to allow programmers to describe what they want separately from how to achieve this goal. This is based on the premise that any algorithm consists of two parts: a logical specification, the logic, and a description of how to execute this specification, the control". [5]

IV. OPS-5 LANGUAGE

The OPS-5 language was created by Charles Forgy in 1981 with the primary objective of enabling the development of expert systems [13]. Because it is a rules-based language it obeys the production system containing a working memory (set of data to search for a solution), a production memory (set of rules in the form SE... THEN) and a rule interpreter that performs a comparative operation to determine which rule will be used in a given processing cycle [14].

> *"The rules of OPS-5 are completely independent of each other. They can be placed in production memory in any order. If the data in the work memory corresponds to the conditions of a rule in the production memory, the actions of the rule occur. Possible actions include changing the contents of the work memory (which can then*

correspond to the conditions of another rule), reading information from a file, displaying information on the screen, and calling up external programs. In some versions of the language, the actions of a rule can cause a new rule to be created, allowing systems to develop that learn". [15]

Among the main characteristics of language are efficiency, generality and powerful mechanisms of pattern marriages [13]. The expert system is usually characterized by a search process with the help of rules since the latter, alone, do not solve the problem.

The internal inference mechanism of OPS-5 starts from a set of data in the working memory and causes a sequence of rules to be applied until a goal is satisfied. The other prominent inference procedure of the expert system is the backward chaining in which a goal to be achieved causes the sub-objectives to be satisfied until a problem solution is found. Backward chaining can be implemented in OPS-5 with some work [15].

"OPS-5 has been used successfully in numerous applications. The most famous of these is the R1 expert system for configuring VAX computers (McDermott 1980). OPS-5 has also been used for image analysis (McKeown et al. 1985)". [13]

Currently it seems that language is a little forgotten in view of the emergence of new languages, but there are still some references about it[1].

V. JULIA LANGUAGE

The Julia language was first created in 2009 by Jeff Bezanson, Stefan Karpinski, Viral B. Shah and Alan Edelman and the first version appeared in 2012. It is a multiparadigm language (functional, imperative, object oriented) and was influenced by languages like MATLAB, LISP, Ruby, Python, among others. The language was thought for scientific computation, fast like C or Fortran, but of simple use with the objective of facilitating computational modeling [16].

Among Julia's main characteristics we have: multiple dispatch (allows defining different behaviors for the same function from different combinations of the types of its arguments), macros similar to LISP, it is possible to call libraries of Python and C, parallelism and distributed computing, user-defined types as fast as native ones, excellent performance, agility to develop programs [17].

Jeff Bezanson and the team explained why they created a new language:

"We're greedy: we want more. We want a language that is open source, with a liberal license. We want the speed of C with the dynamism of Ruby. We want a language that has homoiconicity, with real macros like Lisp, but with obvious and familiar mathematical notation like Matlab. We want something as useful for general programming as Python; as easy for statistics as R; as natural for string processing as Perl; and as powerful for linear algebra as Matlab... Something that's pretty simple to learn, but still satisfies the most serious hackers. We want it to be interactive and easily compiled." [18]

According to its creators, Julia's just-in-time compiler allows the language to match the C language in terms of performance in various scenarios. It also has an expressive amount of additional packages to increase the language's resources such as packages for modeling and emulating circuits, packages for generating graphs, solving differential algebraic equations, dealing with astronomical and astrophysical routines, creating executable code instead of the interpreted program, using stochastic systems and many others [16][2].

"As long as the language of the developers is more difficult to understand than the language of the users, numerical computing will always be impaired. This is an essential part of Julia's design philosophy: all basic functionality should be possible to implement in Julia - never force the programmer to resort to the use of C or Fortran. Julia solves the problem of both languages". [19]

The core of Julia's code is formed by 11,000 lines of C, 4,000 lines of C++ and 3,500 lines of Scheme. The language authors argue that it has significantly less low-level code to maintain than most script languages. The standard library contains 25,000 lines of code written in Julia and provides over 300 numerical functions needed in technical computing environments [20]. Of course, these figures exclude the additional packages and libraries that are huge.

Julia has an elegant writing syntax and in fact has been fast when running its code. It is a promising language but still too young for more complete evaluations.

VI. LANGUAGE COMPARISON

We will now continue to make a comparison between LISP, PROLOG, OPS-5 and Julia in some aspects.

MAIN PARADIGM	
LISP	Functional (also procedural and object-oriented - *Common Lisp*).
PROLOG	Logical-declaratory.
OPS-5	Based on rules.
JULIA	Multiparadigm: functional, procedural, object-oriented.

Table 1: Main Paradigm

1 http://www.paxmondeo.com/Products/OPS-5.

2 See complete list of packages at http://pkg.julialang.org/

EXECUTION MODEL	
LISP	Interpreted (can be compiled as well).
PROLOG	Interpreted (in reality pre-compiled).
OPS-5	Interpreted.
JULIA	Interpreted (can be compiled through additional packages).

Table 2: Execution Model

INFLUENCED BY	
LISP	Calc lambda.
PROLOG	Warren's abstract machine.
OPS-5	Rete algorithm.
JULIA	MATLAB, Scheme, Common Lisp, Lisp, C, Python, Perl, Ruby, R, Lua.

Table 3: Influences

Performance testing

To perform the performance test we decided to calculate the factorial of 100,000. In the LISP language we used the following algorithm (The interpreter used was clisp5 to run the script):

```
(defun fatorial (n)
  (setq x 1)
  (loop for counter from 2 to n do
          (setq x (* x counter))
  )
  x
)
```

In the case of the PROLOG language we use the SWI-Prolog interpreter. The interpreter pre-compiles the script (swipl -o fatorial -c fatorial.pl) before it can be used. Below we see the script that was used:

```
fatorial(0,1).
fatorial(N,X) :- N1 is N - 1, fatorial(N1,X1), X is N * X1.
```

Finally for the Julia language we use the Julia interpreter version 0.5.1 (2017-03-05 13:25 UTC). Follow the script that was used:

```
factorial(big(100000))
```

The experiments were conducted on a notebook with Intel® processor Core™ i5-3210M CPU @ 2.50GHz × 4, with 6 GB of RAM, using the Linux Ubuntu 16.04 LTS operating system.

Unfortunately we were not able to measure the performance relative to the OPS-5 language because we could not find an interpreter compatible with the Linux operating system.

The results of this experiment are summarized in Table 4:

PERFORMANCE FACTORIAL OF 100,000	
LISP	33,000 milliseconds (33 seconds)
PROLOG	130,000 milliseconds (2 minutes and 10 seconds)
OPS-5[8]	-
JULIA	150 milliseconds

Table 4: Calculating factor of 100,000

VII. CONCLUSION

This work briefly presented the programming languages LISP, PROLOG, OPS-5 and Julia. The objective was to know a little about the structure of these languages and verify their applicability for the construction of expert systems.

LISP and PROLOG are languages with very good features to build expert systems. LISP has in its favour the long "road" time and many successful applications. PROLOG is a language with a perfect mechanism for searching a large "disorderly" database and extracting useful information. The OPS-5 language works with the rules system and it seems to have been created for very specific applications and is not suitable for developing more common applications. We found a source code in OPS-5 to perform the factorial calculation (experiment done to test the performance of languages) but we could not run it due to the absence of an interpreter for Linux. However, it draws attention to the amount of code to make this simple calculation. See Annex I. Finally we have the Julia language which was, by far, the one with the highest performance in the factor calculation test. Besides having an elegant and easy syntax, the promises made by their creators in terms of their performance seem to have been fulfilled.

An important observation is that there is no better language than the other, but languages to achieve different goals. The Julia language was created with the goal of being as versatile and fast as possible. However, because it is a very new language, we cannot say whether this ambitious objective has been or will ever be achieved.

REFERENCES

[1] Silva, Renato Rocha; Lima, Sérgio Muinhos Barroso. **Consultas em Bancos de Dados Utilizando Linguagem Natural.** Faculdade Metodista Granbery. Disponível em: <http://re.granbery.edu.br/artigos/MjQ0.pdf> Acessado em: 11/05/2017, 21:20h.

[2] Ramos, Carlos. **Algoritmia Avançada/Teórico-Práticas/3ºano/LEI.** Instituto Superior de Engenharia do Porto. Gecad. Disponível em: <http://www.dei.isep.ipp.pt/~jtavares/ALGAV/downloads/ALGAV_TP_aula2.pdf> Acessado em: 11/05/2017, 21:05h.

[3] Dantas, Luciano Assis. **Descobrindo o PROLOG.** Artigo escrito para o site linha de código. Disponível em: <http://www.linhadecodigo.com.br/artigo/1697/descobrindo-o-PROLOG.aspx> Acessado em: 11/05/2017, 21:04h.

[4] Vicente, André Abe. **Apostila PROLOG.** Universidade Estadual do Oeste do Paraná. Curso de Bacharelado em Informática. Cascavel, Setembro de 2005.
Disponível em:
<http://jeiks.net/wp-content/uploads/2012/05/Apostila_PROLOG.pdf>
Acessado em: 11/05/2017, 19:34h

[5] Oliveira, George Souza, Silva, Anderson Faustino. **PROLOG: A Linguagem, A Máquina Abstrata de Warren e Implementações.** Universidade Estadual de Maringá.
Disponível em: <https://www.google.com.br/url?sa=t&rct=j&q=&esrc=s&source=web&cd=8&cad=rja&uact=8&ved=0ahUKEwjFlNushunTAhWFl5AKHT4iCfwQFghOMAc&url=http%3A%2F%2Fseer.ufrgs.br%2Frita%2Farticle%2Fdownload%2Frita_v20_n2_p155WesleyVol20Nr2_214%2F25453&usg=AFQjCNHddN5TYmph_BUPgLgNwSBKweoLhQ >
Acessado em: 11/05/2017, 21:10h.

[6] Song, Siang Wun. **Uma Aula Prática sobre LISP.** Universidade de São Paulo – IME/USP. MAC 5710 - Estruturas de Dados – 2008. Disponível em: <https://www.ime.usp.br/~song/mac5710/slides/10lisp.pdf > Acessado em: 12/05/2017, 19:26h.

[7] Alvarez, Miguel Angel. **O que é Lisp?** Artifo do site criarweb.com. Disponível em:
<http://www.criarweb.com/artigos/238.php > Acessado em: 12/05/2017, 19:34h.

[8] Fred, Ana. **Introdução ao Lisp.** 2002. Disponível em: <http://users.isr.ist.utl.pt/~lmmc/iasd/Lisp_intro.pdf> Acessado em: 12/05/2017, 19:41h.

[9] Tutorials point. **Learn LISP**. Copyright 2014 by Tutorials Point (I) Pvt. Ltd. Disponível em: <https://www.tutorialspoint.com/lisp/lisp_tutorial.pdf> Acessado em: 12/05/2017, 19:43h.

[10] McCarthy, John. **History of Lisp.** Artificial Intelligence Laboratory Stanford University, 12 February 1979. Disponível em:
<http://www-formal.stanford.edu/jmc/history/lisp/lisp.html > Acessado em 12/05/2017, 21:36h.

[11] Araújo, Valéria. **Linguagens de programação LISP e PROLOG.** 22/09/2003.
Disponível em: <http://www.zemoleza.com.br/trabalho-academico/exatas/informatica/linguagens-de-programacao-lisp-e-prolog/ > Acessado em: 12/05/2017, 21:52h.

[12] Meidanis, João. **Paradigmas de programação Lisp.** Copyright 2011 J. Meidanis. Disponível em: <http://www.ic.unicamp.br/~meidanis/courses/mc336/2011s2/lisp/apostila-lisp.pdf > Acessado em: 12/05/2017, 19:53h.

[13] Velasco, Flávio Roberto Dias. **Uma implementação da linguagem OPS-5 para computadores compatíveis com o IBM-PC.** Trabalho submetido para apresentação no 49 Simpósio Brasileiro de Inteligência Artificial, de 13 a 16 de outubro de 1987, Uberlândia, MG.
Disponível em:
<http://mtc-m12.sid.inpe.br/col/sid.inpe.br/iris@1912/2005/07.19.01.12.44/doc/INPE%204291.pdf> Acessado em: 13/05/2017, 13:23h.

[14] Gomide, Fernando. **Sistemas Baseados em Regras.** EA 072 Inteligência Artificial em Aplicações. UNICAMP. Disponível em:
<http://www.dca.fee.unicamp.br/~gomide/courses/EA072/transp/EA072SistemasBaseadosRegras6.pdf> Acessado em: 13/05/2017, 13:28h.

[15] PCAI. **OPS Programming Language.** Disponível em: <http://www.pcai.com/web/ai_info/pcai_ops.html> Acessado em: 13/05/2017, 13:39h

[16] Site oficial. **Linguagem Julia**. https://julialang.org/. Acessado em: 13/05/2017, 18:17h

[17] Correia, Allan Lucio; Anjos, Maryklayne Araujo dos. **Uma Breve Análise da Linguagem Julia.** Universidade Federal de Alagoas, Arapiraca-AL, 2015. Disponível em: <https://www.youtube.com/watch?v=pS0XXKhDMY0>
Acessado em: 13/05/2017, 18:51h.

[18] Ventura, Felipe. **A ambiciosa linguagem de programação que quer substituir Python, R e Matlab.** GIZMODO Brasil, 4 de fevereiro de 2014 às 11:00. Disponível em: <http://gizmodo.uol.com.br/julia-linguagem-programacao/>
Acessado em: 13/05/2017, 19:18h.

[19] Bezanson, Jeff;Edelman, Alan; Karpinski, Stefan; Shah, Viral B. Julia: **A Fresh Approach to Numerical Computing.** Society for Industrial and Applied Mathematics, 2017. Vol. 59, Nº. 1, pp. 65-98.
Disponível em: <https://julialang.org/publications/julia-fresh-approach-BEKS.pdf > Acessado em: 13/05/2017, 20:11h.

[20] Bezanson, Jeff;Edelman, Alan; Karpinski, Stefan; Shah, Viral B. Julia: **A Fast Dynamic Language for Technical Computing.** **S**eptember 25, 2012. Disponível em: <https://arxiv.org/pdf/1209.5145.pdf > Acessado em: Acessado em: 13/05/2017, 20:12h.