Seminar Report

Om
An Esoteric Programming Language

Sabine Oberleiter
s.oberleiter@student.uibk.ac.at

17 July 2015

Supervisor: Thomas Powell

Abstract

This report gives an introduction to the esoteric programming language Om. The aim of this report is to explain how the basic ideas of the language are implemented and which of the main ideas are not implemented in the actual version. For a better understanding of the language and the individual components, some example programs and operation mode are analyzed in the report. Furthermore, the actual situation around the programming language is described.

Contents

1 Introduction

1.1 History And 'Vision'

The Om Programming Lanuguage is planned as a very simple functional language with a really minimalistic syntax. The language is actually part of esoteric languages which means languages with novel concepts, created for tests, new ideas or just for joke. The creation of the Om Programming Language is from beginning until now just a one-person project which started in November, 2012, Om is created by the software developer Jason Erb alias sparist. Jason Erb comes from Ontario, Canada and tries to create a new programming language with some really interesting behaviours. Actual Om is on a very early proof of concept which implies that it is not complete nor stationary. The Language should reach significant changes on it's way to version 1.0 expected are significant changes on Om's way to version 1.0 it is a minimalistic , maximally-simple language with just 3 elements [4]

1.2 Esoteric Languages

Esoteric programming languages are programming languages that have different characteristics: On one side there is the kind of esoteric languages that are being developed for testing purposes only and not for practical use. Some are designed for specific algorithms or different syntaxes for testing. In addition, there are languages that are designed just for fun and just a "boozy" evening inscribedable as their origin. Some developers of such languages, just want to try out new concepts and unconventional appraches to languages and the success and development of the language depends on a functioning concept.
With esotericism in the true sense, as "secret" doctrine or teaching for limited number of people esoteric programming languages have little to do: In this field, we used esotericism as "outlandish, not general-purpose language and the majority of people have no or limited access to these languages. [1]

1.3 Om In Detail

Jason Erb as developer defines his language in the following words:

> 'Om is an experimental high-level programming language with the goal of maximal simplicity. It is a concatenative, homoiconic, embeddable programming and algorithm notation language, implemented in C++, with minimal syntax (only three elements),

> prefix notation (whereby functions manipulate the remainder of the program itself), and ‘panmorphic ‘typing (allowing programming without data types).‘[4]

For better understanding of the definition some special expressions will be explained:

1.3.1 Concatenative

This expression means that any Om program evaluates to a function, every function takes a program as input and return a program afterwards as output. If two programs will be concatenated with a separator the language returns the evaluation of the composition of the corresponding functions. [3]

1.3.2 Homoiconic

Some programming languages have this property where the program structure is similar to its own syntax, the program's internal representation can be inferred by reading the text's layout. Homoiconic languages has the same structure as its abstract syntax tree. [4]

1.3.3 Prefix Notation

A function takes the results of a program as its input and returns again a program as output, this output is the input for the next coming function and so on.[4]
Prefix Notation offers some really great advantages:

- A Stack can't get an underflow.
- Om stores a composed stored function instead of a data stack in memory.
- The Om evaluater is able to read, parse and evaluate input streams in a single pass and sends the result to the output stream directly after evaluating.
- Optimizing functions due to read only the required data into memory.
- Hints to the users can be provided from an integrated development.

1.3.4 Panmorphic Typing

Panmorphic typing enables a programming without data types. Every data type is just represented as an operand within the language and for this reason any operation will accept any operand as a valid input. The operation is free to process the data however is appropriate, and any operand that it produces as output can be interrogated and processed by next operation in the same way. [2]

1.4 Actual Use Of Language

Om can be used for building a stand-alone interpreter from a script generated build project, which means a code file can be written, saved and just be compiled and executed. The second possibility of using is to include it as a C++ header-only library, this allows us to use it similiar as a normal library and a handling very near to C++. [2]

1.5 Syntax

The programming language Om has a really minimal syntax, where each Om program is a combination of only three elements:

- Separator
- Operand
- Operator

These different elements can be used in following mode: Om uses prefix notation without

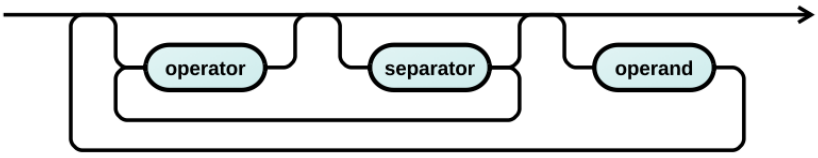

Figure 1: Syntax Usage

being a stack language and so functions manipulate the remainder of the program itself. [2]

1.5.1 Separator

A separator is an atomic program element just used to separate other elements and there are only 3 different ways to separate these elements: space, tab and line. Separators are discarded from input, they are just inserted between output terms. [2]

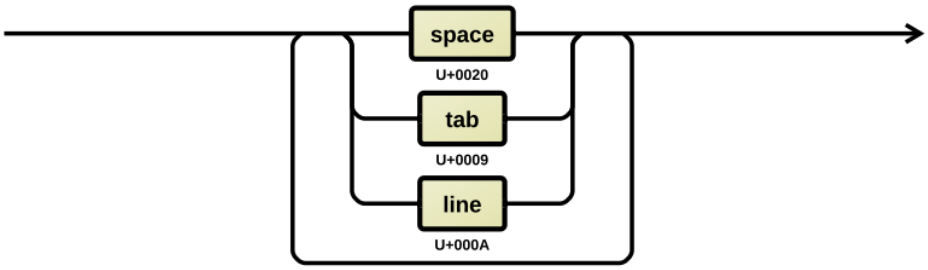

Figure 2: Separator Usage

1.5.2 Operand

An operand itself is a program that wraps a program. The operand evaluates to a constant function and pushes the operand to the ouput program, followed by all input terms. In Om there exist no traditional data types and so every data value is represented by operands. [2]

Figure 3: Operand Usage

1.5.3 Operator

An operator is an atomic element which is defined by any UTF-8 string and will be used as in the image below. An operator is evaluated to an operation that is defined for an operator around. If none exists, it is evaluated a constant function that pushes the operator onto the output program. [2]

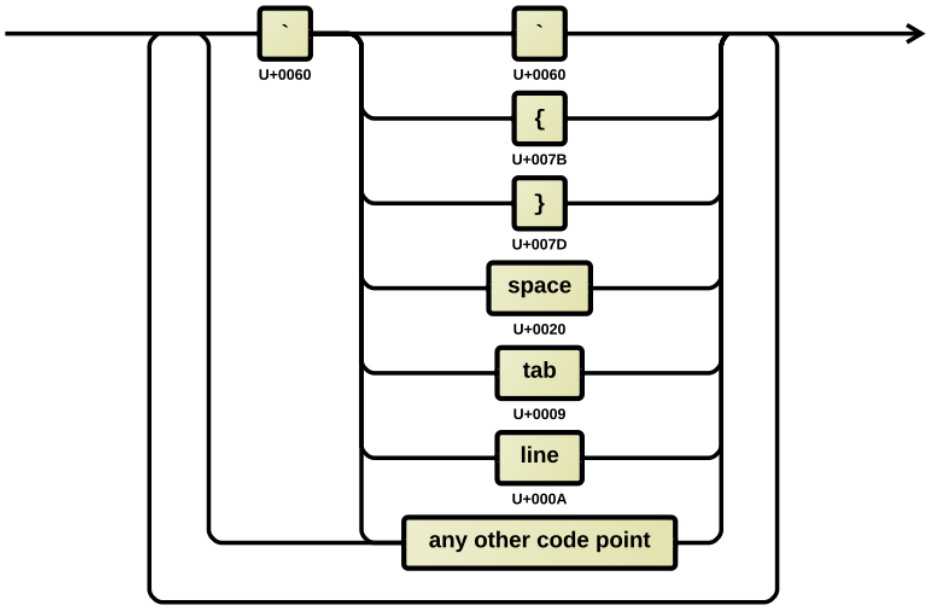

Figure 4: Operator Usage

2 Analogies

At the actual proof of concept of the programming language Om it is really hard to find similarities to other languages. Om offers a lot of possibilities which should be realised in future but are not really done yet.

2.1 ..To OCaml and Haskell

The similarity between languages like Haskell and OCaml is that they all are functional languages. OCaml and Haskell offer a lot of different functions and types which can be used for programming, Om instead is a panmorphic language, which offers to program without any datatype.
OCaml and Haskell use characters as internal type, Om doesn't use characters. Om uses in every part of the language UTF-8.
Writing a program in OCaml or Haskell might be hard in many different ways but it is really possible to do that and also to find a lot of help for programming in such a language. There are even a lot of people who know very well how OCaml or Haskell work and how to get familiar with. In Om there is just only one person who understand how to write some code and how to use the language or to get familiar with and this is Jason Erb, Om's developer himself.

2.2 ..To Other Esoteric Languages

Om has similarities and differences also with other esoteric languages. For example the similarities to Iota and Jot are that all three are fundamentally simple languages.
But there exist also differences: Both Iota and Jot don't have basics like named functions unlike Om where named functions exist. [5]

3 Example Code

The following Code parts of Om should show how the program can be used, and how the Om program code looks like. These are examples to show the drop and copy of operations in the curly brackets. [2]

3.1 Drop Example

```
drop {A}{B}{C}
output: {B}{C}
```

3.2 Copy Example

```
copy {A}{B}{C}
output: {A}{A}{B}{C}
```

3.3 Drop Copy Example

```
drop copy {A}
output: {A}
```

3.4 Copy Example

```
copy copy {A}
output: {A}{A}{A}
```

The small code examples show how really simple commands will be implemented in the Om programming language. These examples are easy to understand and to implement

for every person with basic experience in programming. [2]

3.5 Minutes Example

The next example is at the first view as simple as these examples above but on a second view, the program is really huge and needs a lot of steps to be evaluated:

```
define {minutes
        {dequote choose {minutes} {} = {:} <-[characters] } }
                { minutes {1:23} }


output: {23}
```

- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { **minutes** {1:23} }
- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { *dequote choose {minutes} {} = {:} <-[characters]* {1:23} }
- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote choose {minutes} {} = {:} **<-[characters] {1:23}** }
- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote choose {minutes} {} = {:} *{1} {:23}* }
- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote choose {minutes} {} **= {:} {1}** {:23} }
- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote choose {minutes} {} *{}* {:23} }
- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote **choose {minutes} {} {}** {:23} }
- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote *{minutes}* {:23} }
- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { **dequote {minutes}** {:23} }
- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { *minutes* {:23} }
- define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { **minutes** {:23} }

Figure 5: Working steps of minutes code - part 1

```
• define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote choose {minutes} {} = {:} <-[characters] {:23} }
• define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote choose {minutes} {} = {:} <-[characters] {:23} }
• define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote choose {minutes} {} = {:} {:} {23} }
• define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote choose {minutes} {} = {:} {:} {23} }
• define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote choose {minutes} {} {{:}} {23} }
• define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote choose {minutes} {} {{:}} {23} }
• define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote {} {23} }
• define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { dequote {} {23} }
• define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { {23} }
• define
  { minutes { dequote choose {minutes} {} = {:} <-[characters] } }
  { {23} }
• {23}
```

Figure 6: Working steps of minutes code - part 2

In the different converting lines of the minutes code every part of the code which will be replaced is marked **bold** and the parts which are marked *italicized* is the code which is at last replaced. Here it can be seen, that in Om a code of just 3 lines needs 22 evaluation steps to be executed. [2]

4 Importance and Disappearance

4.1 Importance

The elementary idea for Om was a simple, full-featured language for programming and algorithm notation. Basically the main idea is just the same as in many other functional programming languages like Haskell or OCaml, from this point of view Om could grow up and could possibly be important. In this current state there is a large scope for interpretation how the language itself will develop.
Om is a start up language with very slow progress which is due to the fact that developing

the language is just a one person project. The programming language is actual 2 and a half years old and there are too many things missing, to assess how important the language could be in the future.

4.2 Disapperance

With regard to the reasons mentioned above for Om's importance it is not really a big suprise that, except some students of a specialization seminar and Jason Erb himself, rather nobody would notice if Om disappeared.

5 Current Situation and Trends

5.1 Current Situation

Om is actual a language on the very beginning, so it is very difficult to find information on the language. Searching in libraries was completely unsatisfactory, as there is no printed information about Om in our latitudes. On the Internet, information about Om is also very limited, the majority of searchable information comes from the developer of the language.
Searching for information was, however, very helpful for learning about Hindu culture, the Sanskrit language, and how to find easily a course on meditation. Furthermore it was really interesting to see how many widespread the name "Om" is in our and other European countries.

5.2 Trends

The Om Main Page [2] invites everybody who is interested in helping Jason Erb on further development of the language. There are a lot of areas on the page that only turn on development and how this could be handled.
The page show how code can be forked from the GitHub repository or how the code can be builded. Further there is a lot of help about adding or removing files, adding operations and adding programs. Analyzing the code and test coverage is explained as well as submitting the changes. Afterwards on the page are some reporting issues and funding mentioned. [2]

6 Discussion and Forecast

6.1 Discussion

Working and programming with Om in the current state of development is nearly impossible for everybody except Jason Erb, which is his own wording. [5] The Om Main Page[2], where all information about the language and a documentation is posted, puts more emphasis on developing the language than on programming with it. Hopeful this changes with reaching Version 1.0 and programming will be engaged more in the center. Another point for discussion is that the documetation site only shows really simple and short examples, there is nothing that is just a bit more complex. The fibonacci sequence as an example could be very helpful for programmers to show how the language work and to give a first impression how the language can handle these complex things.

6.2 Forecast

With respect to the current version 0.1.3., it is not conceivable that there are many people who are using Om actively, especially given the low follower count on the Om homepage and GitHub page [2, 4].

If Om were to be finished and were to become usable, with its post-condition data types and the fact that its syntax contains only three elements it could become important, but in reality there are so many unfinished points that it is impossible to make a forecast for its future. In my personal opinion, because of all the outstanding issues it is difficult to imagine that the language will succeed. I think that developing a language with the main goal being to find "God's programming language" is too ambitious. It would have possibly been better to start with more natural and attainable goals so that step-by-step something meaningful could be achieved. The main problem with aiming so high is that motivation is lost once it is realised that the main goals cannot be reached after all.

References

[1] Anonymous. Esolang, the esoteric programming languages wiki, 2013. esolangs.org/wiki.

[2] s. Jason Erb. Om mainpage. sparist.github.io/Om/.

[3] s. Jason Erb. Concatenative languages, om, 2013. concatenative.org/wiki/view/Om.

[4] s. Jason Erb. sparist/om, 2014. github.com/sparist/Om.

[5] J. E. William Tanksley. Discuss the concatenative variety of computer languages, 2013. comments.gmane.org/gmane.comp.lang.concatenative/3695.