Comparative Analysis of Functional Programming Languages: Miranda and LFE

José Ricardo Serathiuk da Silveira*

2023

Introduction

This study dives into the characteristics of LFE (Lisp Flavoured Erlang) and Miranda languages. Through a comparative analysis, we aim to unravel the distinct features of each language.

Miranda

The Miranda language was created in 1983 by David Turner in the company Research Software Ltd. of England. The first public release was released in 1985. It was created for research and for teaching functional programming. It was the main inspiration for the Haskell language. It was first described on the paper "Miranda: A non-strict functional language with polymorphic types". It was a proprietary language until 2020. In 2020, David Turner announced Miranda as a open source language using the BSD license, with an update for modern computer architectures. David Turner died in 2023, aged 77.

Named Functions

The function in Miranda has the follow structure:

```
[function name] p1, p2, ... = [Function body]
```

Example:

```
sum_numbers a b = a + b
```

*ricardo@serathiuk.com

To call the function you can use the follow code:

```
sum_numbers 2 3
```

Anonymous Functions

According Glynn e Pope (1999), all Miranda functions needs to be named.

Modules

In Miranda, the absence of a Module structure is compensated by the presence of an "Abstract Type"structure, identified by the keyword abstype. This powerful feature empowers you to craft structures reminiscent of Modules in other functional languages.

```
abstype type_declaration
with function_type_signature_1
     ...
     function_type_signature_N

type_declaration_instantiation

function_definition_1
...
function_definition_N
```

Example:

```
abstype stack *
with empty::stack *
     push::*->stack *->stack *
     pop::stack *->stack *
     top::stack *->*
     isempty::stack *->bool
     showstack::(*->[char])->stack *->[char]

stack * == [*]
empty = []
push a x = a:x
pop(a:x) = x
top(a:x) = a
isempty x = (x=[])
showstNck f [] = "empty"
showstack f (a:x) = "(push " ++ f a ++ " " ++ showstack f x ++ ")"

teststack = push 1 (push 2 (push 3 empty))
```

User Defined Types

The programmer can create 'Abstract Types' like specified in Modules section. Another User Defined Type is Algebraic Types:

```
new_type_name::= Value1 | Value2 | ... | ValueN
```

Examples:

```
switch::= On | Off
rgb::= Rgb(num,num,num)

switch=On
rgb_color=Rgb(0,47,167)
```

Recursive Function

```
|| recursive factorial
|| whitespace is significant - notice the use of layout in the example

rec_factorial n = 1, if n=0
                = n * rec_factorial (n-1), n>0

|| alternate version using pattern matching:
pattern_factorial 0     = 1
pattern_factorial (k+1) = (k+1) * pattern_factorial k
```

Lazy or Strict Evaluation

According to (WIKIPEDIA, 2021), Lazy Evaluation in Miranda works as default of the language, without any type of instruction or command.

Static or Dynamic Typing

Miranda is a strongly typed language with Dynamic Typing.

```
|| Boolean
authorized = True
linked = False

|| Integer
age = 25
answer_for_everything = 42

|| Real Numbers
```

```
price = 25.2
weight = 345.23

|| String
name = "Zaphod"
surname = "Beeblebrox"

|| Character
first_letter = 'a'
second_letter = 'b'
```

Tooling: Project Creation Commands, Computational Notebook Usage and Browser-Based Execution

There is no concept of a 'project' in Miranda. Instead, there is a tool called Mira, which serves as the compiler for the language and the default development environment. Additionally, there is no browser-based execution.

The user has the ability to run 'Mira,' which functions as the compiler, editor, and REPL. Within the REPL, the user can type '/edit' to open the default editor (vi). Upon saving the file, the Mira tool will proceed to compile the code.

An extension for IntelliJ, developed by **??**), adds Miranda support to the IDE, providing syntax highlighting. However, the use of 'mira' remains necessary.

Concurrency Support

In Turner (1985) and Clack, Myers e Poon (2011), no mechanism for multithreading or parallelism is cited. The language defaults to immutability, help in a possible concurrency control. However, there are no mechanisms for any kind of parallelism.

Pattern Matching

The default template of pattern matching:

```
function name pattern 1 = function body 1
function name pattern 2 = function body 2
. . .
. . .
function name pattern N = function body N
```

A simple example:

```
dayofweek 1 = "Sunday"
dayofweek 2 = "Monday"
dayofweek 3 = "Tuesday"
dayofweek 4 = "Wednesday"
dayofweek 5 = "Thursday"
dayofweek 6 = "Friday"
```

```
dayofweek 7 = "Saturday"
dayofweek x = "Invalid day"
```

Another examples:

```
|| Comparing
bothequal (x,x) = True
bothequal (x,y) = False

|| Comparing 2
bothzero (0,0) = True
bothzero (x,y) = False

|| Partial matching
twenty_four_hour (x,"a.m.") = x
twenty_four_hour (x,"p.m.") = x + 12

//Lists
gethead a:x = a
gettail a:x = x
```

Error Handling

There's a keyword called "error"that signals an error and terminates the program. However, I haven't discovered a method to control it, akin to the 'throw' and 'catch' mechanisms in modern languages.

I believe it might be achievable using a similar approach as in the Elixir language, but I'm uncertain if there's a recognized convention for Miranda.

```
dayofweek 1 = ("ok", "Sunday")
dayofweek 2 = ("ok", "Monday")
dayofweek 3 = ("ok", "Tuesday")
dayofweek 4 = ("ok", "Wednesday")
dayofweek 5 = ("ok", "Thursday")
dayofweek 6 = ("ok", "Friday")
dayofweek 7 = ("ok", "Saturday")
dayofweek x = ("error", "Invalid day")

showmessage ("ok", day) = "It's "++day
showmessage ("error", msg) = msg
```

Community and Ecosystem

The compiler serves as the official ecosystem for Miranda. However, my search hasn't unveiled any signs of Miranda gaining traction within a local community, be it in a university, school, company, or any other domain.

Interoperability

The latest builded versions:

- Windows/Cygwin version 2.041 (note this is for 32-bit Cygwin)
- Intel/Linux version 2.042 (replaces earlier versions 2.038, 2027)
- SUN/Solaris version 2.028
- Intel/Solaris version 2.038
- MacOS X version 2.044 (Intel)
- MacOS X version 2.032 (PPC)

The Miranda compiler can be built using the source code from version 2.066, and it is compatible with x64 operating systems. The same Miranda program can be compiled for all these platforms.

LFE

The Lisp Flavored Erlang (LFE) programming language was created by Robert Virding, with initial development beginning in 2007 and more focused work starting in 2008. Virding, an experienced Lisp programmer, was interest in implementing a Lisp language. He desired to implement a Lisp that would run on and integrate with Erlang, with a specific focus on compatibility with the BEAM (Erlang's virtual machine) and Erlang/OTP.

Robert Virding, a co-inventor of Erlang, played a significant role in the early stages of the Ericsson Computer Science Lab, contributing to Erlang's system design and libraries, as well as its compiler.

Named Functions

The function in LFE has the follow structure:

```
1 ([function name] (p1, p2, ...)
2     ([Function body]))
```

Example:

```
1 (sum_numbers (a b)
2     (+ a(e b))
```

To call the function you can use the follow code:

```
1 (sum_numbers 2 3)
```

Anonymous Functions

Creating and calling an anonymous function in LFE.

```
;;To create
(set add-function (lambda (a b) (+ a b)))

;; To call
(funcall add-function 3 4)

;; Completely anonymous
(funcall (lambda (a b) (+ a b)) 1 3)
```

Modules

Creating a module:

```
(defmodule tut1
  (export all))

(defun double (x)
  (* 2 x))
```

Creating a module using the module contract.

```
(defmodule tut1
  (export (double 1)))

(defun double (x)
  (* 2 x))
```

User Defined Types

The programmer can create records as own types in LFE. The record structure is:

```
;; To define the record
(defrecord <name-of-record> <field-name-1> <field-name-2> ...)

;; To create a record
(make-<name-of-record> <field-name-1> <field-value-1> <field-name-2> <field-value-2>)

;; Record structure (is a tuple)
#(<name-of-record> <field-value-1> <field-value-2>)
```

Example of record:

```
;; To define the record
(defrecord person name age)

;; To create a record
(make-person name "Joao" age 23)

;; Record structure (is a tuple)
#(person "Joao" 23)
```

The modules mentioned earlier can also be considered a user-defined type.

Recursive Function

Factorial using LFE.

```
(defmodule tut2
  (export (fac 1)))

(defun fac
  ((1) 1)
  ((n) (* n (fac (- n 1)))))
```

Lazy or Strict Evaluation

LFE is based on Erlang, which is a Strict evaluation language.

Static or Dynamic Typing

LFE shares the dynamic typing characteristic with Erlang, making it a versatile and flexible language.

Project Creation Command

LFE uses the rebar3 Erlang tool to create a new LFE project. The command to create a project:

```
rebar3 new lfe-lib my-test-lib
```

Computational Notebook Usage

I haven't found any software like Livebook or Jupyter that is compatible with the LFE language.

Browser-Based Execution

I haven't found any browser-based execution for LFE language.

Concurrency Support

LFE has concurrency support, and it's actually one of its core features. LFE is built on top of the Erlang Virtual Machine (BEAM), inheriting all of Erlang's capabilities, including its renowned concurrency model.

Pattern Matching

The Pattern Matching on LFE works like Erlang or Elixir.

Fibonnaci:

```
1 (defun fib
2   ((0) 0)
3   ((1) 1)
4   ((n) (+ (fib (- n 1)) (fib (- n 2)))))
```

Error Handling

Error handling in LFE, like in Erlang, is largely based on the "let it crash" philosophy. This approach doesn't mean that errors are ignored, but rather that systems are designed to fail gracefully and recover cleanly. You can use the try-catch clause to deal with program errors, like exceptions. I have not found any good documentation explaining how to use this command in LFE.

Community and Ecosystem

There is not an active community for LFE, compared with Elixir or Erlang. The main forum has no recent posts. The official documentation is incomplete. About the ecosystem, there is a REPL and tools based on Rebar3. Editors for Lisp-2+ based languages can be used to edit code in LFE.

Interoperability

According to LFE (2023), the programmer has "the ability and freedom to utilize Erlang and OTP libraries directly from a Lisp syntax". You can use all the libraries made for Erlang in LFE. But, I have not found evidence proving the inverse is true. If you can invoke LFE code in Erlang.

Referências

CLACK, C.; MYERS, C.; POON, E. *Programming with Miranda*. [s.n.], 2011. Disponível em: <http://www0.cs.ucl.ac.uk/teaching/3C11/book/Ch1-2.pdf>. Citado na página 4.

GLYNN, K.; POPE, B. *Haskell for Miranda Programmers*. 1999. Disponível em: <https://www.berniepope.id.au/assets/files/mira2hask.pdf>. Citado na página 2.

LFE, T. *LFE Documentation*. 2023. Disponível em: <https://lfe.io/learn/>. Citado na página 9.

TURNER, D. *Miranda system manual*. 1985. Disponível em: <https://www.cs.kent.ac.uk/people/staff/dat/miranda/manual/>. Citado na página 4.

WIKIPEDIA. *Lazy evaluation*. 2021. Disponível em: <https://en.wikipedia.org/wiki/Lazy_evaluation>. Citado na página 3.