TALLINN UNIVERSITY OF TECHNOLOGY

RUBY

Analysis of Programming Languages (IAG0450)

Anna Dočekalová
(156586IV)

Tallinn 2015

Contents

Introduction

Ruby

Ruby can be described as a dynamic, reflective, object-oriented and general purpose programming language [1]. Its creator Yukihiro Matsumoto released it in public in 1995. As an inspiration for its creation were used five other programming languages (Perl, Smalltalk, Eiffel, Ada and Lisp). That's why Ruby combines functional and imperative programming. In 2006 it gained massive acceptance and now is ranked among the 10 most popular languages worldwide. [2]

Philosophy of the language

The author created Ruby with the intention for it to be object oriented and easy to use scripting language, because he was not satisfied with Perl or Python at that time. He describes Ruby as a language with Lisp core, Smalltalk object system and Perl practical utility. [1]

Matsumoto said that the primary purpose of the language is for programmers to be productive and enjoy programming. He wants to focus on human needs and comfort while programming, not primarily on the machines. [3]

Essential characteristics

Syntactic and semantic definition

When we compare Ruby language to Java, C or C++, there are many syntactical differences. They include operators, variable naming, loop constructions, differences in keywords or functions and many others. In this chapter I will describe the most crucial ones.

Operators

Here will be listed operator differences from the previously mentioned languages [4]:

- "<" operator that denotes inheritance between two classes
- "<=>" comparison operator. Returns -1, 0, or +1 depending on whether the first object is less than, equal to, or greater than the other object. It is used to implement the usual comparison operators (<, <=, ==, >=, >).
- "<<" concatenation operator and binary left shift operator
- "=" operator supports multiple assignments: `i1, i2 = i2, i1`
- "=>" associative operator. Can be used e.g. for associating a variable name with Exception object for exception handling.
- "=~" operator used for matching, e.g. a regular expression in a string. A regular expression is constructed by placing a string between two "/" signs, e.g.: `puts(/abc/=~'xyabc')` will find return a position of the regular expression (position number 2).
- ". ." and „. . ." range creation, e.g. (1..10) creates a range from 1 to 10, (1...10) creates a range from 1 to 9.
- „===" tests wheter a value is a member of a range

Iterations and blocks

The looping is not done over an index, but with use of **do…end** blocks similar to this:

```
some_list.each do |this_item|
  # We're inside the block.
  # deal with this_item.
End
```

Apart from `each`, different blocks and functions can be used to iterate over an object, e.g. `collect`, `inject`, `sort`, and many others. [5]

```
b3 = [1,2,3].collect{|x| x*2}
```

In this example, the `collect` function creates new array from the results of the mathematical function inside `collect`. The result is stored in `b3` variable (it's the return value of the block). This syntax is very similar to Smalltak language. [4]

Assignments

In Ruby, everything is an expression and has its value (or `nil`). That's why e.g. `if else end` construction can be assigned to a variable. The return value will be the value of the code following `if` or `else` keyword. In the previous example, also the block (its result) was assigned to a variable. [5]

Naming of variables

In order of recognizing an identifier as a certain variable type, it must meet the following requirements [5]:

- Constant - starts with a capital letter
- Global variable - starts with a "$" sign
- Instance variable – starts with a "@" sign
- Class variable – starts with a "@@" sign

Other differences

Every value, except for `nil` and `false` is considered as true (even zero). In Ruby, everything is an object. Because of Smalltalk influence, Ruby is a purely object oriented language, as shows the following example: `5.times { print "Hello world!" }` [5]

Reliability

It is a dynamically-typed language, which means that as many operations as possible are resolved at the run-time, e.g. [6]:

- type of a variables – one variable name can be used in the code with different data types
- libraries needed to be used together with our code
- order of method calling

It is possible to modify the code during its execution – to enter some new lines of the code, which will execute even if we don't restart the program run (with the MRI interpreter) [4].

It has a built-in garbage collector, so the memory is freed automatically [4].

Exceptions are handled with help of Exception class (or it's descendants) [4]:

```
begin
# Some code which may cause an exception
rescue <Exception Class>
# Code to recover from the exception
else
# optional section - executes if no exception occurs
ensure
# optional section - always executes, useful e.g. for closing a file
end
```

Fast translation

It is an interpreted language. The first standard interpreter was the MRI (Matz's Ruby Interpreter) and was used until the version 1.9 [1]. From version 1.9., it was replaced by YARV (Yet Another Ruby VM) [1]. This started to use compilation into intermediate code to improve speed of interpreting. The speed was improved seven times compared to the older MRI when using code of the Fibonacci sequence computation (1.83 sec) [7].

Efficient object code

The YARV virtual machine uses instructions in VMDL (virtual machine description language), which can generate optimized code automatically. For example it includes special instructions for adding two integer numbers. This is useful, because every number is an object and the mathematical operator is in fact method of the number class (and can be redefined). When it encounters the + method and finds out it was not redefined, the special instruction is called and the adding of two objects is more effective. Otherwise more complicated method dispatch will be carried out. [7]

The AOT compilation (Ahead-of-Time) is also available. The Ruby code is firstly translated into a C code and then executed on the YARV, which is more effective than YARV instruction code. [7]

Orthogonality

„Orthogonality means that features can be used in any combination, that the combinations all make sense, and that the meaning of a given feature is *consistent*, regardless of the other features with which it is combined.“ [8]

The author of Ruby thinks that allowing to use any combination of a small feature or syntax leads to complexity of the language. The user can be confused and needs to think like a compiler to know, which syntactic combination is allowed. At one time, one functionality should be used. [9]

However, there can be found examples of orthogonality. On collections (e.g. arrays and hash lists) can be performed the same types of iterations - `each` and `collect` methods. [10]

Machine independence

Ruby is interpreted and a cross-platform language. The code written in Ruby can be run on Windows, Linux or Mac, if there is a platform-dependent Ruby interpreter provided. [4]

Desirable characteristics

Provability

It does not include any system to write a program and also it's proof of correctness.

It is possible to use unit tests. This is done by series of assertion methods, e.g.:

```
class MyTest < Test::Unit::TestCase
def test1
t = TestClass.new(10)
assert_equal(100, t.getVal)
end
```

The created test will fail, if the result of `t.getVal` is not equal to 100. [4]

Generality

The program is the more general the less different syntactic constructs it allows to use to perform the same functionality [11].

There can be found examples that show Ruby as a general language. An Array is a general object. It has methods **:pop** and **:push** (for adding and removing from the end of an Array) and **:shift** and **:unshift** (for adding and removing from the beginning of an Array). Therefore, besides the traditional functions of an array, it can be used as a Stack or a Queue. [12]

However some collections have two different methods that perform exactly the same thing - e.g. arrays have synonymic methods `size` and `length`. [12]

Consistency with commonly used notations

Users which were programming previously in C, C++ or Java will have to get used to using some new language constructs - their examples are listed in the syntax description. Among others, Ruby was influenced by Smalltalk and resembles it in many ways. On the other hand many of the syntax resembles also Perl, Python and even Java [5].

Ruby supports wide variety of syntax and in some cases, programmers can choose the one they prefer. For example [4]:

- Blocks can be enclosed by both curly brackets or by `do … end` keywords.
- Method will return last expression evaluation, but the `return` keyword at the end of methods can be also used.

Subsets

There are no subsets of Ruby in the meaning that there exists a programming language with a syntax that would be compatible to Ruby. However, besides official Ruby MRI implementation,

there is a version of Ruby (JRuby), which is compatible to Java Virtual Machine and it is able to run on it [13].

Uniformity

Uniform programming language should have different syntax for constructs with different meaning and similar syntax for constructs that have similar meaning [14].

Ruby is not uniform when using the "<<" operator. It can be used as a binary left shift operator: `a << 2`. At the same time, it is an operator for merging of objects, e. g [4]:

- concatenation of strings: `s = "Hello " << "world"`
- array merging:
```
a =[1,2,3]
b =[4,5,6]
a << b  #=> a=[1, 2, 3, [4, 5, 6]]
```

This operator can be used also for associating an object with a singleton class (a class to which only once object instance will be associated) [4]:

```
ob = Object.new
class << ob  # singleton class
    def congratulate( aStr )
        puts("Congratulations! #{aStr}")
    end
end
```

Extensibility

Ruby is a very flexible language. The user has the freedom to alter even essential and core classes and operators. Even the name of mathematical operators used for number operations can be redefined, e.g. [2]:

```
class Numeric
  def plus(x)
    self.+(x)
  end
end
```

Now, for adding two numbers, we can call: `y = 5.plus 6`

Summary

Ruby is a general-purpose programming language, which is purely object oriented. It offers wide range of functionality and syntactic language constructs for a programmer and it focuses on being user friendly. It can be consider reliable, because of its built-in garbage collector and exception handling. It is an interpreted language which originally uses MRI interpreter, however there is a more recent and faster implementation of the interpreter - YARV, which include better code optimization. Ruby is a cross-platform language and it is aimed to be non-orthogonal. It offers wide extensibility and flexibility.

Works cited

[1] https://en.wikipedia.org/wiki/Ruby_(programming_language)

[2] https://www.ruby-lang.org/en/about/

[3] https://en.wikipedia.org/wiki/Ruby_%28programming_language%29#Philosophy

[4] Huw Collingbourne. *The Book of Ruby*. No Starch Press, 2009. ISBN: 1593272944. Available at: http://www.sapphiresteel.com/ruby-programming/The-Book-Of-Ruby.html

[5] https://www.ruby-lang.org/en/documentation/ruby-from-other-languages/

[6] https://www.ruby-lang.org/en/documentation/ruby-from-other-languages/to-ruby-from-c-and-cpp/

[7] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.482.9851&rep=rep1&type=pdf

[8] Michael Scott. *Programming Language Pragmatics [online]*. Available at: http://brandonbyars.com/2008/07/21/orthogonality/

[9] http://www.artima.com/intv/rubyP.html

[10] http://www.tutorialspoint.com/ruby/ruby_iterators.htm

[11] http://jcsites.juniata.edu/faculty/rhodes/lt/plcriteria.htm

[12] http://ruby-doc.org/core-2.2.0/Array.html

[13] http://jruby.org/

[14] http://www.eecs.ucf.edu/~leavens/ComS541Fall97/hw-pages/comparing/