Analysis of Eolang efficiency
Stage IV

HSE Team
hsalekh@hse.ru
HSE
Moscow,Russia

Abstract

EO programming language is a new object-oriented programming language that has been going through many phases of development in a research and development project. In this phase, we compare Eolang efficiency to C++ and Java; detect differences; identify pros and cons by SWOT; define metrics to compare, create benchmark tests, and write test cases as well as automate testing and bench-marking to run for different EO compiler versions. The main goal is to analyze Eolang efficiency.

Keywords: SWOT, efficiency, metrics, benchmark, test cases

1 Introduction

This report includes metrics analysis, SWOT analysis, benchmark description and results. Benchmarks and tests are available at GitHub repository and integrated into release pipeline in the form of Continuous Integration (CI).

2 SWOT Analysis

It is important to note that there are several points that are difficult to compare for a number of reasons, as far as various programming languages concerned. One of the controversial points is that Eolang is generally positioned as a language intended for static code analysis, which, however, is not explicitly advertised and there is no special emphasis on this in the current project. However, this leads to the fact that initially it is essentially a subject-oriented, and not a universal language that does not allow effectively displaying not only the styles of writing programs, but also having a much smaller set of expressive means for describing algorithms and data.

At the same time, since the task is to conduct a certain analysis, as well as to compare in terms of efficiency with a pair of universal languages, a number of points may be made and further detailed.

2.1 Strength

Here, in general, a fairly simple semantic model of the language can be noted, which is due to the initially laid down idea of forming an "elegant" programming style. Based on this, the program contains only objects, the semantics of which allows them to be used as actions (directives) that provide both a description of the functionality of algorithms and the structuring of data. This allows you to form a fairly compact semantic model, which is the strength of the language.

The solutions proposed in the language increase the reliability of the generated code, albeit often at the expense of efficiency. But for the main target of analysis and reliability improvement, this is not a significant factor.

When compared with the C ++ and Java languages, it can be noted that in these languages there are many unreliable constructs for programming. The languages themselves have many redundant and overlapping constructs, which often do not allow generating unambiguous and reliable code.

Compactness, extensibility and openness can be used to describe the strength of Eolang.

2.2 Weakness

One of the weaknesses is that the limited capabilities of the language do not allow it to be used in many subject areas in comparison with C ++ and Java. That is, where high performance computing is required.

Another point to mention is lack of tools that provide support for parallel and distributed computing, which is currently used in one form or another in almost all modern programming languages.

And not enough attention is paid to the formation of the type system. Using a typeless solution can turn out to be unreliable in many situations, which, in turn, may lead to difficulties associated with static code analysis. In addition, in some cases, in order to increase control over data when writing programs in Eolang, additional constructions will have to be introduced to model data types and explicitly check them either during static analysis or at runtime.

For C ++ and Java, static typing is used to control the data at compile time. In addition, these languages support dynamic typing due to Object-Oriented polymorphism and the possibility of dynamic type checking at runtime.

Lastly, conceptual incompleteness.

2.3 Opportunity

Most likely it is prudent to start with the possibilities, since they determine the specifics of the language. The limitations by means of the OO paradigm and recursive computations based on the absence of object mutability possibly simplifies the code for static analysis, but at the same time significantly reduces the number of effective techniques used in real programming. When creating algorithms, you often have to write longer and more inefficient code, which is difficult to further optimize when reduced to a real executor. At the same time, as the practice of using functional programming languages shows, the use of similar techniques increases the reliability of programs and ensures the formation of controlled code.

Note: If one tries to inflate the topic, then it is possible to make concrete what is not (intentionally) in comparison with existing languages. For example, there are no pointers, a small set of atomic data types, etc.

When comparing with C ++ and Java, it is enough to note here that both proposed languages are universal and include tools for writing programs that allows one to choose between reliable and efficient programming. In principle, it is possible to list these tools, emphasizing what is not in Eolang, emphasizing that this significantly expands the possibilities of programming, but often at the expense of the reliability of the code.

Additionally, there is potential formalizability of semantics, presence of formal calculus of objects, convergence of object and functional paradigms, potentially short and reliable code.

2.4 Threat

Among the main threats, we could considered the use of the language not for its main purpose, which can lead to writing code that will be less expressive than the code written in C ++ and Java. At the same time, attempts to model the constructs of these languages in Eolang can lead to more cumbersome and less reliable code. In particular, explicit modeling of data types will require validation at runtime, or may lead to the development of additional analysis programs to isolate and match data types during the static analysis phase.

It may also be worth noting the problem associated with the lack of type control when entering data, when the incoming data in the presence of a typeless language model will be difficult to control. There are similar threats in other languages, but the presence of a static type system or explicit dynamic typing allows the ability to control the input and transformation of data directly using language constructs without additional modeling.

Also, there is not a well-defined/traditional concept of "object".

3 Metrics to compare

It is no possible to come up with metrics here. Even the most complex algorithms in this situation will clarify the metrics. The bottom line is that any algorithm, in the presence of certain means in the language, will be expressed in approximately the same way as in another language. Take Fibonacci or factorial, etc. Maybe take the number of objects used in the developed object or program in Eolang as a metric of complexity? This is understandable, since only objects exist in the language.

It is certainly possible to write code in different languages and compare in terms of length. That is, to collect statistical metrics for expressiveness. But here it is already possible to use examples equivalent to those already written. Large code is unlikely to provide more information.

Another option for comparison might be to try to see what the modeling in Eolang of constructs and operators of traditional languages pours into. That is, what the Eolang implementation will result in, such as support for types, loop operators, assignments, etc.

Basically, talking about languages, and not their instrumental support.

1. Their focus on certain classes of tasks (how universal the language is)
2. Basic programming paradigm
3. Tools to simplify programming (garbage collectors, etc.) and so on.

4 Comparison

In order to compare Eolang efficiency to C++ and Java and detect the differences, we select a number of algorithms to implement in all three languages. The following algorithms were chosen:

1. Prim's algorithm
2. Dijkstra's algorithm
3. Kruskal's algorithm
4. Ford-fulkerson's algorithm

The justification for choosing these specific algorithms is that graph algorithms are usually complex, contain loops and recursive calls, and therefore, you can test them and compare them base on many different criteria.

4.1 Criteria for comparison

There are many criteria important to comparing or evaluating general purpose programming languages:

1. Simplicity of language constructs, which relates to ease of programming
2. Readability, which relates to maintainability, an important factor as many programs greatly outlive their expected lifetimes
3. How tuned a language's features are for a particular application (e.g., Perl relates well to text processing)
4. Compilation speed
5. Runtime efficiency, in terms of speed and machine resources
6. Library support
7. Debugging help
8. Language safety
9. Longevity of language and compiler tools
10. Portability across platforms and machine architectures

The criteria are equally important because they affect the development cost and effort required over the lifetime of the program, and also affect the usefulness and quality of the developed program.

5 Continuous Integration (CI)

6 Conclusion

References