Open Access Library Journal
2020, Volume 7, e6318
ISSN Online: 2333-9721
ISSN Print: 2333-9705

Interlanguage Translation Utility with Integrated Machine Learning Algorithms

Suren Abazyan[1], Narek Mamikonyan[1], Vakhtang Janpoladov[2]

[1]Yerevan State University, Yerevan, Armenia
[2]Russian-Armenian University, Yerevan, Armenia
Email: su.abazyan@gmail.com

How to cite this paper: Abazyan, S., Mamikonyan, N. and Janpoladov, V. (2020) Interlanguage Translation Utility with Integrated Machine Learning Algorithms. *Open Access Library Journal*, **7**: e6318.
https://doi.org/10.4236/oalib.1106318

Received: April 12, 2020
Accepted: April 24, 2020
Published: April 27, 2020

Abstract

Same program can be written in different programming languages and different ways. One programming language will have advantages and disadvantages compared to another; hence sometimes it is needed to rewrite the code into another language to support this or that functionality. This paper demonstrates an algorithm of interlanguage translator, which is using intended machine learning (ML) algorithms to ensure higher translation rate. In the scope of research, tool for interlanguage translation is created and tested with two programming languages: Python and Java. Comparison results for translation accuracy are calculated, about 53% without AI algorithms and about 65% with the use of AI algorithms.

Subject Areas

Computer Engineering

Keywords

Python Translator, Compiler, Interlanguage Translator

1. Introduction

Translating code from one language to another is not just converting syntax between these languages. In this process conversion needs to be done maintaining code structure as much as possible. There are many code translators which are for translating from one specific language to another [1]-[6].

However, for translating between not only two exact languages, more complex methods need to be under consideration. In [7] an intermediate language is used for storing translation data while transferring into another language.

In this paper, we represent interlanguage translator that is powered with machine learning algorithm, which is comparing translation result if needed and reports difference between source and makes changes in translation algorithms. In this way translation is being done with higher accuracy. For adding new language in translator, extension of that language is needed, which is a basic map for those language keywords. If new language does not support any object (for example pointer in C++ or lambda in Python), translator algorithm tries to find analogue of that, with mentioning code part or line where this approximation has been done.

For proposed utility tests, Python and Java programming/scripting languages are selected with their extensions, because these are two most commonly used object-oriented languages.

Few acronyms used in the text are explained in Table 1.

2. Workflow and Engine

Main workflow of tool consists of two parts: programming to intermediate language translator and AI comparator/corrector (Figure 1).

Run of tool is starting with reading source code that needs to be translated and recognizing keywords to load libraries. After that libraries that are sourced into code are being found and located with individual ID number. This is for being able to link to these libraries if needed.

Next step is separating code into FRAMEs. Everything from first level of source code is being considered as TMODULES of GLOBAL_FRAME. All other FRAMEs are considered to be sub-FRAMEs of GLOBAL_FRAME too. After source code division into FRAMEs, TMODULES of each FRAME is being separated with unique ID, path, type, initial value and name. This ensures each TMODULE to have exact place in code after translations, also with keeping its type for being able to declare them in statically typed programming languages (like C++ or Java).

Paths for FRAMES, TMODULES and all attributes that are collected from source code, later will be stored in intermediate language (Figure 2).

In given example (Figure 2) G1 is abbreviation for GLOBAL_FRAME. All functions, classes, variables without parent FRAMEs are considered as child TMODULES of GLOBAL_FRAME. As all TMODULES have unique ID and exact place, same name variables from different FRAMEs will not be overlapped:

- TMODULE_{a}: G1/foo/a
- TMODULE_{a}_1: G1/bar/bar_foo/a

Table 1. Explanation for acronyms used in text.

Acronym	Explanation
TMODULE	Language keywords, variables, functions and all other data that is in code text and should be translated
FRAME	Loops, If-else conditions, functions, classes from source code
GLOBAL_FRAME	Main FRAME for source code

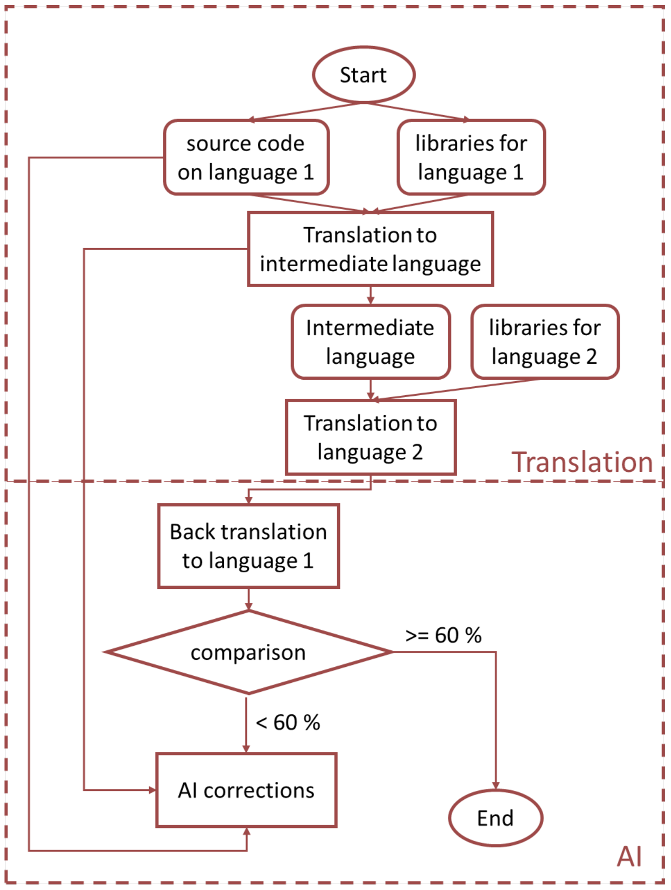
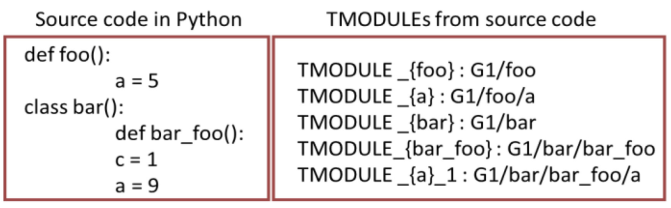


Figure 1. Explanation for acronyms used in text.



Source code in Python	TMODULEs from source code

def foo():
    a = 5
class bar():
    def bar_foo():
    c = 1
    a = 9
	TMODULE _{foo} : G1/foo
	TMODULE _{a} : G1/foo/a
	TMODULE _{bar} : G1/bar
	TMODULE_{bar_foo} : G1/bar/bar_foo
	TMODULE _{a}_1 : G1/bar/bar_foo/a

Figure 2. Data saved in intermediate language.

In general intermediate language that is used can be described as in below:

```
CODE_NAME: {
    GLOBAL_FRAME: {
        VAR: {    NAME: VARNAME, ID: VARID, INVAL: INITIAL_VALUE
                  TYPE: BOOL/STR/INT… , PATH: G1
        }
        FUNCTION: { NAME: FUNCNAME, ID: FUNCID, PARAM: Parameter
        list with initial values
        }
        FRAME_FUNC_A {  …  }

        }

        CONNECTION_LIST: {

            CONNECTION_1: { TMODULE_{ID_1} : TMODULE_{ID_2} }

            …..

        }

    }
```

Here CONNECTION_LIST is interaction between two or more TMODULES.

3. Translation to Second Language, Back Translation and AI

In translation to second language step, already translated into intermediate language TMODULES are being replaced with second language keywords. In this step, libraries for second language are included. If any function cannot be translated into second language, proper note is being written and outputed file contains empty module of second language. This is the part, where manual checks are needed [8].

If translation is done with success status, programm is waiting for command to either do back translation and comparison with source code or to exit and write translated file.

In back translation already translated into second language code is treated as source code and upper described methods are being applied to it. Later new code with language 1 being compared with source language 1 code. All differences are marked and based on that AI makes changes in TMODULE path or connection list.

Successful translation pass rate is selected to be 60%. This means comparison result less than 60% will force AI algorithms to make corresponding changes in translation algorithm. If comparison rate is higher than 60%, translation will be treated as succeed.

Translation rate is being obtained from two comparisons. First is direct text comparison between first language source and post-processed, back-translated first-language code. In this comparison, code blocks (this can be any TMODULE or any FRAME) which are matching but are located on different place of correct FRAME, are considered to be matching. Second comparison is two intermediate language comparisons. In this part, intermediate language that is obtained on first language into second language translation is compared with back translation intermediate language. In this step upper described method of approximation is applied. Final comparison result is the arithmetic mean of these two results.

4. Conclusion

In this paper interlanguage translation utility is presented. This utility is using intermediate level language, so that any other language translation can be added into the tool as an extension. After translation is done, tool performs minor changes to ensure higher source comparison rate. Tool can achieve about 65% comparison rate for translated files compared with source code.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] java2python. https://github.com/natural/java2python

[2] Varycode. http://www.thetaranights.com/varycode

[3] Tangible Software Solutions. https://www.tangiblesoftwaresolutions.com

[4] JLCA. http://support.microsoft.com/kb/819018

[5] BCX. http://bcx-basic.sourceforge.net

[6] Perthon. http://freshmeat.net/projects/perthon

[7] Coco, E.J., Osman, H.A. and Osman, N.I. (2018) JPT: A Simple Java-Python Translator. *Computer Applications: An International Journal* (CAIJ), **5**, 1-5. https://doi.org/10.5121/caij.2018.5201

[8] George, D., Girase, P., Gupta, M., Gupta, P. and Sharma, A. (2010) Programming Language Interconversion. *International Journal of Computer Applications*, **1**, 63-69. https://doi.org/10.5120/419-619