A Syntax-driven Approach for Natural Language to Programming Language Translation to Realizing Literate Programming in Java

by

Hung Phan

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Hridesh Rajan , Major Professor
Mary Jones
Bjork Petersen

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation/thesis. The Graduate College will ensure this dissertation/thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2018

DEDICATION

I would like to dedicate this thesis to my family. Without whose support I would not have been able to complete this work. I would also like to thank my friends and family for their loving guidance and financial assistance during the writing of this work.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

ACKNOWLEDGMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Hridesh Rajan for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. August Tanner and Dr. Lewis Hargrave. I would additionally like to thank Dr. Tanner for his guidance throughout the initial stages of my graduate career and Dr. Hargrave for his inspirational teaching style.

ABSTRACT

Computer Programming should be considered as art . Literate Programming (LP) (Knuth (1984)) is a programming paradigm proposed by Donald Knuth to realize this idea. LP helps developers to make the source code to be more like a literature, by the ability of integrate the explanation in natural language (NL) along with original source code. However, since its appearance in 1984, LP has not been used as a popular programming paradigm. One of the challenges is that developers need to write the corresponding source code manually for any NL parts they defined to make the program compilable. In this project, we proposed Natural Language to Programming Language (NLPL) system, to reduce the effort of writing code. Our NLPL system allows developers to write explaination as code comment in NL and automatically generating the respected source code following the requirements of the NL parts. The code generation is done by the NLVisitor module we developed for providing translation rules at Natural Language syntax level and alleviating indirect references problems between languages. The experiment on the set of 52 comments from the code suggestion tool AnyCode (Gvero and Kuncak (2015)) achieves the result at 75% in top-1 accuracy, which outperforms this prior work (62%).

CHAPTER 1. OVERVIEW

1.1 Literate Programming

The notion of Literate Programming (LP) provides a potential programming paradigm which brings advantages to software developer. Considering that writing source code need to see as an art work, LP handles the input source code by two part: documentation and implementation. The implementation part contains the source code in a specific programming language like Java or C/ C++. Unlike original programming language paradigm, the full implementation of a source code file can be splits in several parts, which are interleaved by documentation parts. documentation parts usually provide the description of the expected code in natural language. Each documentation part are corresponded with a implementation part, in which defined by the same title prefix.

An example of a LP file is shown in figure 1. This file, which is written by C++ published from cite, shows the destructor of the Xstring class. The implementation part of this method is defined in other area and it is aliased by a documentation part. The documentation part contains the description of its corresponding code.

LP provide advantages to software developers. This programming paradigm allows users to understand source code better. It provide a layer to help developers able to read both of natural language representation and programming language representation of the code. In example showed in figure 1, developers know what action need to do for the destructor in the natural language part, which is decrementing a variable, and how to do the action by a decrement assignment and an if statement. Besides, LP helps developer read the documentation part aligned with the programming part, which they can check the explanation of behaviors of each program element instead of reading the implementation and documentation separately. This way of representation make developers imagine about how the source code work for each implementation part and be able to debug the code easier. In overall, LP provides the most understandable form of source code,

including what does the code do and how it work by natural language and programming language .

1.2 Challenges of realizing Literate Programming

Realizing LP requires both representation of documentation and implementation parts for each source code file. There are two possible trends to derive these parts, however they faces the problems of effort consuming and low accuracy.

1.2.1 Manually writing documentation and implementation

Since the appearance in 1984, LP has not been applied as a popular programming language) This fact is surprising since LP provides advantages for code understanding. According to]cite, the main reason is that manually writing the code given documentation parts and manually documented the code given implementation parts are expensive tasks, especially with people who only have background in writing documentation or only have background in writing the code.

For people who are focusing their work on writing software documentation, doing the implementation part of the documentation are usually infeasible task for them to complete. They know about what elements need to be defined or handled in the implementation and know how program elements involved implementation parts. However, their knowledge is represented by natural language. The most straight forward way for them is learning about programming language to have background of writing code. However, it is not a cheap stacks since it requires 4 years on average for training a Computer Science undergraduate student in the US. Natural Language and Programming Language has many differences how they abstracting the software explanation. For example, in natural language, people can use some indirect reference like pronoun such as "it", "they" while their is no such pronoun representation in some common programming languages like java. Besides, people who are writing software requirements might need to work with projects in several languages instead of a single programming language, cause it is impossible for them to infer the implementation parts even if they already have background in one language.

From vice versa, people who have experiences in doing the implementation parts of the code might have more background to manually get the documents the code they write. This could be possible and recommended for any developers who are students to do the documents correspond to their small software projects, to make them understand the code better and check the bug in the code faster if bugs exist. However, in industrial projects which contains more than 1000 line of codes, the cost for writing documentation related to each parts of code are expensive, while it can slow down the development process of a software project as half, since developers need to write the explanation for the code they write. In addition, the code does not stay the same and code are updated days by days following requirements of users. This fact brought challenges for software developers to manually write documentation although they have good background for explaining the code in natural language. In overall, manually writing both documentation parts and implementation are expensive. So that, approach for automatically deriving documentation and implementation are important.

1.2.2 Automatically inferring implementation by Machine Translation

In translation between natural languages, phrase based machine translation (PBMT) and Neural Machine Translation (NMT) are two common machine translation (MT) approaches that show their effectiveness for the inference problems. This fact brings MT as the most used translation engines over the world).. We might think about a solution for the software engineering is that applying MT to inferring implementation from documentation, given the intuition that both implementation and documentation have the same intension of describing how computer program works. However, researches on using MT in this area still not have good results yet), due to several following problems.

1.2.2.1 Lack of high quality Parallel corpus

In natural language processing, current MT techniques rely on large scale corpus, which contains 10000 to millions pairs between different languages). Machine learning algorithms like Bayesian

learning are used to learn the mapping between source and target languages. The corpus needed to be prepared as pairs of sentences, and the target languages need to be written manually by linguistic experts. However, in software engineering, building such a parallel corpus for documentation and implementation manually are expensive. Another approach is to relying on documentation written as natural language code comments. This seems to be a good direction since large scale code corpus contains millions line of code (LOC) and there are textual description in the form of code comments for each source code files. However, this type of documentation doesn't guarantee to have the same intent of describing the behavior of related source code. For example. this documentation contains "TODO" tags which are automatically generated by the IDE, or contains comments that are actually source code which is commented by developers. In fact, such a corpus generated by documentation in the form of code comments and implementation contains noise and cause very low quality inference in MT models) . Along with the problems from the shortage of good parallel corpus, MT also facing the problems from the differences between natural language and programming language, A survey) summarized the main differences between two languages which hindered an MT models for learning mapping between two languages. In general, it comes from the different purpose of translation between natural language processing and program generation, and more important, natural language and programming language have different mechanism of indirect reference management).

1.2.2.2 MT for Natural Language to Natural Language: Descriptive Model

In Natural Language, the main purpose for inferring the target language is to helping user understand about the main idea of the source sentence. In the other words, it solved like providing the description for the content of source language, which include what is the main subject, what are the verbs and what are objects used in the sentence in a form of a sample subject-verb-object structure sentence. Let's see an example in figure). In this example, which I observed from NMT system for Germany to English translation published by tensorflow), the generated neural machine translation result shows almost the same with the expected sentence except one word

"Halef" which is a last name. From this generated sentence, users can easily understand the meaning of the sentence and this kind of missing words error are acceptable. In NL, MT worked as a descriptive model, which allows thhe translated result missed one or two words compared to the expected sentence while users still understand.

1.2.2.3 MT for Natural Language to Programming Language: Generative Model

In contrast with NL-to-NL translation, the direction of natural language to programming language requires a very precise generated result. We can look at an example in figure). SpecTrans) is a statistical machine translation system for inferring between documentation and implementation for java specification. The phase of inferring implementation returns 27 % of syntactically correct. SpecTrans returns 38 % of results are close to the expected result like this example. In the example in), although we only used two edit actions to get the correct result, the translated implementation are still syntactically incorrect and cannot use directly as an generated source code. The problem of syntactically incorrect doesn't appear only in documentation to implementation translation but also in translating between object oriented programming languages. SemSMT) shows that there are over 60% of translated results are syntactically incorrect. In overall, generating code from source, even if documentation or code from other programming language is an error-prone task by machine translation.

1.2.2.4 Differences in indirect references between Natural Language and Programming Language

) summarized problems of the generation of code from natural language of several existing tools. In general, the most important is that they have unique ways for handling indirect references. An example of such types of differences is shown by example). In this example, a variable x can be described by the pronoun "it" in natural language as an indirect reference to variable x. However, in the implementation, there is only one way for represent variable x. These differences brings

challenges for MT system to learn the mapping between languages. We will describe about types of indirect references between natural language and programming language in the next section.

1.3 Direction of a syntax-based approach for Natural Language to Programming Language translation

Realizing Literate Programming can take benefits from a module for automatically generating code from documentation. However, the building of such a module based on machine translation engines used in natural language processing faced challenges. In this project, we will rely on an alternative way to build a natural language to programming language based on a rule-based approach.

CHAPTER 2. REVIEW OF LITERATURE

CHAPTER 3. METHODS AND PROCEDURES

CHAPTER 4. EXPERIMENTAL

In this section, I want to answer on the research question: How well NLPL performs for translating from Natural Language comments to code.

4.1 Data Preparation

4.2 Result

No	Natural Language Comment	NLPL Top-1 result	Top-K Acc
		Expected result	
1	copy file fname to destination	FileUtils .copyFile(new File(fname) ,new File(destination))	1
		FileUtils .copyFile(new File(fname), new File(destination))	
2	load class "MyClass.class"	Thread .currentThread() .getContextClassLoader() .loadClass("MyClass .class")	1
		Thread .currentThread() .getContextClassLoader() .loadClass(MyClass .class)	
3	make file text.txt	new LineNumberReader(new InputStreamReader(new File("text .txt"))) .ready()	4
		new File(text .txt) .createNewFile()	

4	write ”hello” to file ”text.txt”	FileUtils .writeStringToFile(new File(”text .txt”) , ”hello”)	1
		FileUtils .writeStringToFile(new File(text .txt), hello)	
5	new buffered reader text.txt	new BufferedReader(new InputStream-Reader(”text .txt”))	1
		new BufferedReader(new InputStream-Reader(”text .txt”))	
6	open connection http://www.oracle.com/	new URL(”http://www .oracle .com/”) .openConnection()	1
		new URL(http://www .oracle .com/) .openConnection()	
7	create socket http://www.oracle.com/ 80	new Socket(”http://www .oracle .com/” , 80)	1
		new Socket(http://www .oracle .com/, 80)	
8	put a pair Mike , +41-345-89-23 into a map	new HashMap() .put(”Mike” , ”+41-345-89-23”)	1
		new HashMap() .put(Mike, +41-345-89-23)	
9	set thread max priority	Thread .currentThread() .setPrior-ity(Thread .MAX_PRIORITY)	1
		Thread .currentThread() .setPrior-ity(Thread .MAX_PRIORITY);	
10	set property gate.home to value http://gate.ac.uk/	new Properties() .setProperty(”gate .home” , ”http://gate .ac .uk/”)	1

		new Properties() .setProperty(gate .home, http://gate .ac .uk/)	
11	does the file 'text.txt' exist	new File("text .txt") .exists()	1
		new File("text .txt") .exists( )	
12	get thread id	Thread .currentThread() .getId()	1
		Thread .getCurrentThread() .getId()	
13	join thread	Thread .currentThread() .join()	1
		Thread .getCurrentThread() .join()	
14	delete file text.txt	FileDeleteStrategy .NORMAL .delete(new File("text .txt"))	1
		new File(text .txt) .delete()	
15	print exception ex stack trace	ex .printStackTrace()	1
		ex .printStackTrace()	
16	is text.txt directory	new File("text .txt") .isDirectory()	1
		new File(text .txt) .isDirectory()	
17	get thread stack trace	Thread .currentThread() .getStack-Trace()	1
		Thread .currentThread() .getStack-Trace()	
18	read line by line file text.txt	new LineNumberReader(new Input-StreamReader(new File("text .txt"))) .readLine()	1
		FileUtils .readLines(new File(text .txt))	
19	set time zone to GMT	Calendar .getInstance() .setTime-Zone(TimeZone .getTimeZone("GMT"))	1

		Calendar .getInstance() .setTime-Zone(TimeZone .getTimeZone(GMT))	
20	free memory	Runtime .getRuntime() .freeMemory()	1
		Runtime .getRuntime() .freeMemory()	
21	total memory	Runtime .getRuntime() .totalMemory()	1
		Runtime .getRuntime() .totalMemory()	
22	new data input stream text.txt	new DataInputStream(new FileInput-Stream("text .txt"))	1
		new DataInputStream(new FileInput-Stream(text .txt))	
23	rename file text1.txt to text2.txt	new File("text1 .txt") .renameTo(new File("text2 .txt"))	1
		new File(text1 .txt) .renameTo(new File(text2 .txt))	
24	move file text1.txt to text2.txt	FileUtils .moveFile(new File("text1 .txt") ,new File("text2 .txt"))	1
		FileUtils .moveFile(new File(text1 .txt), new File(text2 .txt))	
25	read utf from the file text.txt	FileUtils .readFileToString(new File("text .txt"))	2
		new DataInputStream(new FileInput-Stream(text .txt)) .readUTF()	
26	set thread min priority	Thread .currentThread() .setPrior-ity(Thread .MIN_PRIORITY)	1
		Thread .currentThread() .setPrior-ity(Thread .MIN PRIORITY)	

27	create panel and set layout to border	new Panel() .setLayout(new BorderLayout())	1
		new Panel() .setLayout(new BorderLayout())	
28	sort array	Arrays .sort(array)	1
		Arrays .sort(array)	
29	add label Names: to panel	new Panel() .add(new Label("Names:"))	1
		new Panel() .add(new Label(Names:))	
30	write 2015 to data output stream text.txt	new DataOutputStream(new FileOutputStream("text .txt")) .writeInt(2015)	1
		new DataOutputStream(new FileOutputStream(text .txt)) .write(2015)	
31	get date when file text.txt was last time modified	new File("text .txt") .lastModified()	>10
		new Date(new File(text .txt) .lastModified()) .getTime()	
32	check file text.txt read permission	AccessController .checkPermission(new FilePermission("text .txt" , "read"))	1
		AccessController .checkPermission(new FilePermission(text .txt, read))	
33	read lines with numbers from file text.txt	new LineNumberReader(new InputStreamReader(new File("text .txt"))) .readLine()	1

		new LineNumberReader(new InputStreamReader(new FileInputStream(text .txt))) .readLine()	
34	read from console	new BufferedReader(new InputStreamReader(System .in)) .read()	1
		new BufferedReader(new InputStreamReader(System .in)) .readLine()	
35	is file text.txt data available	new DataInputStream(new FileInputStream(new File("text .txt"))) .available()	1
		new DataInputStream(new FileInputStream(text .txt)) .available()	
36	get double value x	Integer .valueOf(x) .doubleValue()	>10
		Double .valueOf(x) .doubleValue()	
37	write object o to file output stream data.obj"	new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream("data .obj"))) .writeObject(o)	1
		new ObjectOutputStream(new BufferedOutputStream(new FileOutputStream(data .obj))) .writeObject(o)	
38	create bit set and set its 5th element to true	new BitSet() .set(5 , true)	1
		new BitSet() .set(5,true)	
39	accept request on port 80	new ServerSocket(80) .accept()	1
		new ServerSocket(80) .accept()	

40	get thread group	Thread .currentThread() .getThread-Group()	1
		Thread .currentThread() .getThread-Group()	
41	create panel and set layout to grid	new Panel() .setLayout(new GridLay-out())	>10
		new Panel() .setLayout(new Grid-BagLayout())	
42	get screen size	Toolkit .getDefaultToolkit() .getScreen-Size()	1
		Toolkit .getDefaultToolkit() .getScreen-Size()	
43	get splash screen graphics	? .getSplashScreen()	>10
		SplashScreen .getSplashScreen() .cre-ateGraphics()	
44	get display refresh rate	GraphicsEnvironment .getLocal-GraphicsEnvironment() .getDefault-ScreenDevice() .getDisplayMode() .getRefreshRate()	1
		GraphicsEnvironment .getLocal-GraphicsEnvironment() .getDefault-ScreenDevice() .getDisplayMode() .getRefreshRate()	
45	get keystroke modifiers	keystroke .getModifiers()	1
		KeyEvent .getKeyModifiers-Text(keystroke .getModifiers())	

46	generate RSA private key	new Thread(”RSA”) .yield()	>10
		KeyPairGenerator .getInstance(RSA) .generateKeyPair() .getPrivate()	
47	reverse list	Collections .reverseOrder()	>10
		Collections .reverse(list)	
48	intersection of rectangle 4 5 with rectangle 3 2	? .intersection(?)	>10
		new Rectangle(5, 4) .intersection(new Rectangle(3, 2))	
49	set cursor over label to hand	new AffineTransform() .setToIdentity()	>10
		label .setCursor(Cursor .getPredefined-Cursor(Cursor .HAND CURSOR))	
50	read big integer from console	new BufferedReader(new InputStream-Reader(System .in)) .read()	3
		new Scanner(System .in) .nextBigInte-ger()	
51	delete file text.txt when JVM ter-minates	new File(”text .txt”) .delete()	>10
		new File(text .txt) .deleteOnExit()	
52	get date instance for Germany	DateFormat .getDateTimeInstance() .getDateInstance()	>10
		DateFormat .getDate-TimeInstance(DateFormat .MEDIUM,DateFormat .MEDIUM, Locale .GERMANY)	

4.3 title

K	AnyCode	NLPL	Top-K of AnyCode	Top-K of NLPL
1	32	39	61.54%	75.00%
2	39	40	75.00%	76.92%
3	41	41	78.85%	78.85%
4	42	42	80.77%	80.77%
5	45	42	86.54%	80.77%

CHAPTER 5. SUMMARY AND DISCUSSION

REFERENCES

Gvero, T. and Kuncak, V. (2015). Synthesizing java expressions from free-form queries. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2015, pages 416–432, New York, NY, USA. ACM.

Knuth, D. E. (1984). Literate programming. *Comput. J.*, 27(2):97–111.

APPENDIX A. ADDITIONAL MATERIAL

This is now the same as any other chapter except that all sectioning levels below the chapter level must begin with the *-form of a sectioning command.

More stuff

Supplemental material.

APPENDIX B. STATISTICAL RESULTS

This is now the same as any other chapter except that all sectioning levels below the chapter level must begin with the *-form of a sectioning command.

Supplemental Statistics

More stuff.