2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) | 978-1-6654-1219-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICSE-Companion52605.2021.00117

RPT: Effective and Efficient Retrieval of Program Translations from Big Code

Binger Chen
TU Berlin
chen@tu-berlin.de

Ziawasch Abedjan
Leibniz Universität Hannover & L3S Research Center
abedjan@dbs.uni-hannover.de

***Abstract*—Program translation is a growing demand in software engineering. Manual program translation requires programming expertise in source and target language. One way to automate this process is to make use of the big data of programs, i.e., Big Code. In particular, one can search for program translations in Big Code. However, existing code retrieval techniques are not designed for cross-language code retrieval. Other data-driven approaches require human efforts in constructing cross-language parallel datasets to train translation models. In this paper, we present RPT, a novel code translation retrieval system. We propose a lightweight but informative program representation, which can be generalized to all imperative PLs. Furthermore, we present our index structure and hierarchical filtering mechanism for efficient code retrieval from a Big Code database.**

I. Introduction

Nowadays, numerous programs are being developed that require translations in other programming languages (PLs) to be further studied, reproduced, or applied on heterogeneous platforms. When the developers do not make the program translation efforts themselves, users have to manually rewrite the software in the needed PL, which is a time-consuming and error-prone process. Since traditional methods based on rule-based compilers or cross-language interpreters are hard-wired and require heavy human intervention for adaptation, the data-driven techniques are getting more traction. Reuse of code from existing "Big Code" repositories, such as GitHub and Bitbucket, has the potential to support many programming tasks including program translation.

Existing data-driven techniques for program translation are based on statistical models, such as 1pSMT [4], mppSMT [5] and Tree2tree [1], which train a program translation model. These approaches usually require a *parallel dataset*, in which programs in different PLs are semantically aligned via manual efforts, to supervised learn the translation model. To avoid generating parallel datasets, recent work leverages a transfer learning approach from NLP [6]. They first train a model that denoises a randomly corrupted program and use it as a pre-trained program translation model, which is then optimized by back-translation method. However, by only relying on NLP features their approach neglects the special features of PLs. Furthermore, programs translated through the aforementioned approaches suffer from grammar mistakes because they are machine-generated programs. Therefore, they usually require additional human-supervision. Also, they are often confined to a few PLs because it is not trivial to extract general features that apply to every PL. Another promising direction is to retrieve similar code in target language directly from Big Code as potential translations. However, existing retrieval systems lack the proper capabilities for cross-language code retrieval [2], [3]. And instead of raw program input, they rely on queries consisting of semantically expressive keywords, descriptions, or user specifications.

In this paper, we propose RPT, a novel **program translation retrieval system**. Given a raw program in a given source PL and a target PL, RPT efficiently retrieves similar programs as potential translations from Big Code, using **a generalizable program representation, an appropriate index structure, and a hierarchical filtering mechanism**. Our approach does not require training data but can compete with existing translation models.

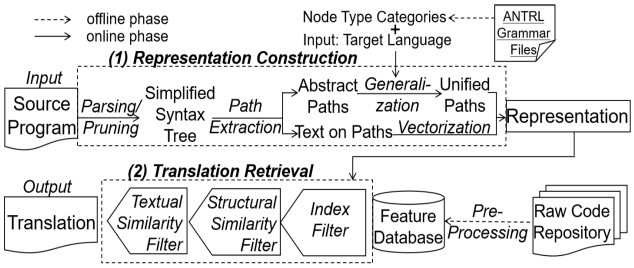

Fig. 1: RPT overview

II. Our approach

We first discuss the necessary program representation and then the employed retrieval process of RPT.

A. Program Representation

To identify cross-language code similarity, we need a unified representation that can be efficiently extracted from any given piece of code. Different from existing methods, RPT considers both structural and textual features and their dependencies. **Structural features** can be captured by either a comprehensive concrete syntax tree (CST) or an abstract syntax tree (AST). The CST retains all the details, making it complicated and verbose with redundant information. The AST has more abstract but less informative syntax. And the abstraction strongly differs for different PLs. Thus, we fall back on the low-level CST as a basis and take the philosophy of AST to construct a unified abstract representation. As shown in Figure 1, RPT first employs a left-to-right parser to parse the source code and generate the original CST, which contains all the nodes and branches of the program structure. Then it is pruned to a simplified syntax tree to

978-1-6654-1219-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ICSE-Companion52605.2021.00117

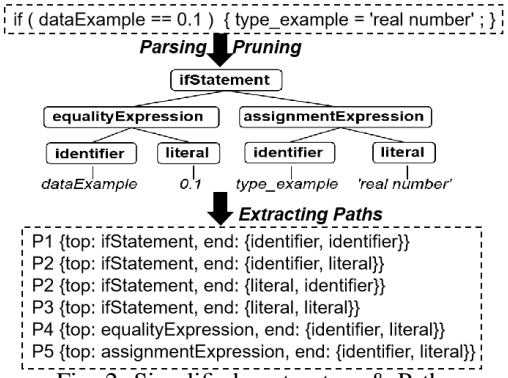

Fig. 2: Simplified syntax tree & Paths

reduce the computation complexity. The simplified tree of a JavaScript code example is shown in Figure 2. The 2-D tree structure is further simplified to a set of 1-D paths that connect the elements on the tree: RPT extracts *abstract* paths by dropping all the intermediate nodes as the leaf-nodes and root node enclose the most critical information. The root node summarizes the whole path and the leaf-nodes directly indicate the content on this path. As the writing habit of programmers and the coding conventions for a PL might differ, RPT ignores the order of left and right leaf-nodes. Figure 2 shows the five extracted path types. Finally, RPT generalizes these path types by matching all the node types across languages and substituting them with their category labels. Our approach leads to a more concise representation than CST but retains the original structural framework that is dropped in AST. Further, RPT only processes text that appears in the extracted paths. RPT does not remove and tokenize numeric values, such as hard-coded floating points and integers, as they might be integral to the program. RPT vectorizes these generated tokens through embeddings. The final **comprehensive representation** consists of three components: the list of path types that appear in a program, the frequency of different path types in a program, and the structure-dependent textual features based on the information of the relative position of text and structure.

B. Translation Retrieval

To make our approach scalable on big code, we implement a **hierarchical filtering mechanism** and a novel **index structure** for effective and efficient retrieval. The representation of each program is constructed offline and stored in a feature database. Our index structure is customized based on our representation. Two similar programs usually share some common path types with similar frequencies. Thus, the index harbors the frequencies of each path type per program. As the frequencies may not be exactly the same, we divide the frequency into multiple buckets and use the bucket intervals as indexes. Because the frequency of each path in each program roughly obeys exponential distribution, we uniformly fill out fixed size buckets with different frequency intervals. We name this as ***p**ath-type-**b**ucket-**i**ndex (*PBI*)*. After using the index, RPT filters candidates based on structural similarity first as it is more discriminative than textual similarity for a program

TABLE I: Program accuracy & time cost per translation

Project	RPT	Baselines			
		TRANSCODER	TREE2TREE	MPPSMT	1PSMT
Lucene	68.8%	53.0%	**72.8%**	40.0%	21.6%
POI	70.0%	51.0%	**72.2%**	48.2%	34.6%
IText	**73.3%**	45.9%	67.5%	40.6%	24.4%
JGit	**74.5%**	49.4%	68.7%	48.5%	23.0%
JTS	**69.1%**	43.2%	68.2%	26.3%	18.5%
ANTLR	**71.9%**	54.9%	58.3%	49.1%	11.5%
Time cost	**0.20s**	0.36s	0.23s	0.24s	0.34s

and its translation. This can also facilitate the dependency between features to influence the subsequent textual similarity filter. Further, RPT runs textual similarity filter to determine the final candidate. For the source program and each candidate, RPT calculates the weighted sum of both similarities.

III. EXPERIMENTS AND CONCLUSION

Experiments. We apply our approach on a Java to C# parallel dataset used in previous work [1], [4], [5]. We compare the results of effectiveness and efficiency of RPT with state-of-the-art baselines 1pSMT, mppSMT, Tree2tree ,and TransCoder. Our metric is *program accuracy* [1]: the percentage of the retrieved translations that are functionality the same as the ground truth in the dataset. The results in Table I show that RPT is competitive to all baselines despite the fact that RPT is fully unsupervised and does not reuse existing data without training any models. Moreover, we observe that for the failed cases the translations tend to appear in the retrieved top 10 list. Further, the efficiency of our retrieval based system is shown to be comparable to other baselines. We also compare our index PBI to a simple path type index. PBI leads to a runtime-improvement by two orders of magnitude at a scale of 3.8GB database.

Conclusion. We proposed RPT, a code-retrieval approach to support program translation with Big Code, which is competitive with existing translation methods. In **future work,** will augment the retrieval system with program generation to overcome the limitations of the database.

Data Availability. We published our code on https://github.com/BigDaMa/RPT.

Acknowledgements. This work was funded by the German Ministry for Education and Research as BIFOLD - Berlin Institute for the Foundations of Learning and Data (ref. 01IS18025A and ref. 01IS18037A).

REFERENCES

[1] Xinyun Chen, Chang Liu, and Dawn Song. Tree-to-tree neural networks for program translation. In *NeurIPS*, pages 2547–2557, 2018.
[2] Xiaodong Gu, Hongyu Zhang, and Sunghun Kim. Deep code search. In *ICSE*, pages 933–944, 2018.
[3] Fei Lv, Hongyu Zhang, Jian-guang Lou, Shaowei Wang, Dongmei Zhang, and Jianjun Zhao. Codehow: Effective code search based on api understanding and extended boolean model. In *ASE*, 2015.
[4] Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N Nguyen. Lexical statistical machine translation for language migration. In *ESEC/FSE*, pages 651–654, 2013.
[5] Anh Tuan Nguyen, Tung Thanh Nguyen, and Tien N Nguyen. Divide-and-conquer approach for multi-phase statistical migration for source code (t). In *ASE*, pages 585–596, 2015.
[6] Baptiste Rozière, Marie-Anne Lachaux, Lowik Chanussot, and Guillaume Lample. Unsupervised translation of programming languages. In *NeurIPS*, 2020.