Suggesting Meaningful Variable Names for Decompiled Code: A Machine Translation Approach

Alan Jaffe
Carnegie Mellon University
Pittsburgh, PA, USA
apjaffe@andrew.cmu.edu

ABSTRACT

Decompiled code lacks meaningful variable names. We used statistical machine translation to suggest variable names that are *natural* given the context. This technique has previously been successfully applied to obfuscated JavaScript code, but decompiled C code poses unique challenges in constructing an aligned corpus and selecting the best translation from among several candidates.

CCS CONCEPTS

• **Software and its engineering → Software reverse engineering**;

KEYWORDS

Reverse engineering, Statistical machine translation, Decompilation

ACM Reference Format:
Alan Jaffe. 2017. Suggesting Meaningful Variable Names for Decompiled Code: A Machine Translation Approach. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4–8, 2017 (ESEC/FSE'17),* 3 pages.
https://doi.org/10.1145/3106237.3121274

1 RESEARCH PROBLEM AND MOTIVATION

The process of *compiling* converts source code meant to be read and written by humans into binary meant to be executed by computers. As such, there is little consideration in compilation for human-centric aspects such as readability. The reverse process, *decompilation*, is an essential part of reverse engineering efforts, aimed at finding security vulnerabilities, analyzing malware, and maintaining legacy code, just to name a few; in contrast to compiling, decompilation is a process in which human users are essential. Fortunately, decompilers such as Hex-Rays [9] and Phoenix [17] exist, and can be used to recover source code from binaries. However, compiled code loses all variable names; consequently, most current decompilers will replace any pre-compilation variable names with uninformative numbered identifiers in the decompiled output.

A great deal of effort goes into selecting meaningful variable names while writing code, as source code is a primary means of communication for programmers [12]. Moreover, useful variable names make it easier to understand the code [14]. Even single-letter variable names can be helpful when selected carefully [2]. In practice, reverse engineers may devote substantial effort to manually fixing placeholder variable names when inspecting decompiled code [4, 6, 8]. This process could be made easier by automatically suggesting more "natural" variable names than the default placeholders, as shown in Figure 1. This is exactly the goal of this research.

2 BACKGROUND AND RELATED WORK

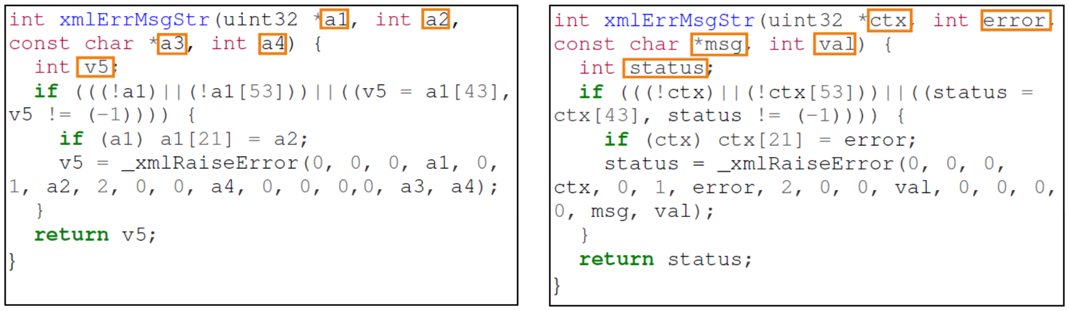
Code is *natural* [5, 10]. The formal language permits great variety, yet real code repeats common patterns in predictable ways. Identifier names are not selected randomly, but rather to fit naturally into the context. Fortunately, there is a plentiful supply of *natural* code in "Big Code" archives such as GitHub. In recent years, statistical techniques have been developed to apply this huge trove of developer knowledge to software engineering problems, including code completion [16], cross-language porting [11], and deobfuscation [15].

For instance, Allamanis et al. [1] used a simple n-gram model to predict better variable names. This approach intends to improve code that is already relatively high quality, rather than generating identifiers from scratch for decompiled code. It achieves high accuracy by frequently choosing to retain the existing identifier, which is only effective when the existing identifiers are already reasonably high quality.

The JSNice tool [15], using the Nice2Predict framework [3], applies a statistical approach to deobfuscation by attempting to recover variable names in obfuscated JavaScript. JSNice manages to improve the rate of successfully recovered variable names by 38.1% compared to a baseline which keeps all variable names unchanged. The approach used by JSNice is language-dependent; it models dependencies between variables using Conditional Random Fields, which requires non-trivial amounts of feature engineering and static analysis.

Most recently, an alternative and seemingly more generalizable deobfuscation approach has been proposed [18], which uses statistical machine translation (SMT) to recover obfuscated variable names. This approach does not require static analysis, and performs comparably well to JSNice [18]. Seeing the success of this approach, we apply it here to decompiled C code.

While the idea of applying statistical machine translation to deobfuscation is relatively recent, SMT has long been used successfuly in translating natural languages [13]. Statistical machine translation



```
int xmlErrMsgStr(uint32 *a1, int a2,
const char *a3, int a4) {
  int v5;
  if (((!a1)||(!a1[53]))||((v5 = a1[43],
v5 != (-1)))) {
    if (a1) a1[21] = a2;
    v5 = _xmlRaiseError(0, 0, 0, a1, 0,
1, a2, 2, 0, 0, a4, 0, 0, 0,0, a3, a4);
  }
  return v5;
}
```

```
int xmlErrMsgStr(uint32 *ctx, int error,
const char *msg, int val) {
  int status;
  if (((!ctx)||(!ctx[53]))||((status =
ctx[43], status != (-1)))) {
    if (ctx) ctx[21] = error;
    status = _xmlRaiseError(0, 0, 0,
ctx, 0, 1, error, 2, 0, 0, val, 0, 0, 0,
0, msg, val);
  }
  return status;
}
```

Figure 1: On the left is the original decompiled C, with uninformative variable names. On the right, we have the suggested names. The true names for $a1$ through $a4$ respectively were *ctxt*, *error*, *msg*, and *val*. $v5$ is an extraneous variable generated by the decompiler (not in the original source code), and yet the system still proposes a reasonable name for it, *status*.

uses the noisy channel model, which treats the source language as a distorted version of the target language. Bayes' theorem is used to break the process into two models. The translation model is trained using aligned parallel data to predict $p(f|e)$, the probability that a particular target language sentence e would be translated into a particular source language sentence f. Meanwhile, the language model uses a monolingual corpus to predict $p(e)$, the probability of generating a particular target language sentence.

Non-statistical approaches to identifier renaming have also been attempted, although they are not the focus of this work. For instance, the DREAM decompiler by Yakdan et al. [20] detects API calls and uses the appropriate parameter names as identifiers. Likewise, DREAM++ by Yakdan et al. [19] extends DREAM with the ability to rename identifiers that are detected as loop counters, array indices, or boolean variables. However, these are strictly rule-based approaches, limited by their reliance on a fixed set of patterns.

3 APPROACH AND UNIQUENESS

Our unique approach is to apply the SMT framework to the problem of suggesting natural identifier names for decompiled C binaries. We use the well-known SMT tool Moses [13], which trains a translation model using aligned parallel data and separately trains a language model based on a monolingual corpus. Obtaining the necessary aligned parallel data poses unique challenges for decompiled C binaries, since the decompilation process distorts the structure of the underlying source code. We generated a parallel corpus by compiling and then decompiling C projects on GitHub, then applying a novel algorithm to align the identifiers appropriately.

We also faced the challenge of selecting a consistent replacement among multiple translations proposed for each variable name.

4 RESULTS AND CONTRIBUTIONS

This work provides three main contributions:

(1) A novel method to extract aligned parallel training data
(2) A statistical machine translation framework for suggesting variable names in decompiled code using Moses SMT
(3) An evaluation of this approach on a held-out test sample

4.1 Extracting Aligned Training Data

We used GHTorrent [7] to select 20225 C-based projects (8.4 billion lines) from GitHub. Each project was compiled, then decompiled using Hex-Rays decompiler [9]. We aligned the decompiled code with the original source code to determine the true variable names. We assumed that the order that the variables are first used in the source code will be the same as in the decompiled code. Then we calculated the similarity between each pair of variables and used a polynomial time dynamic programming algorithm to pick the matching with the best total similarity.

In order to assign these similarity scores, we computed signatures for each variable. Each time the variable is used, we added an entry to the signature containing the nesting depth of the usage in terms of loops, return statements, function calls, and unary operations. The similarity of two variables is defined as the Levenshtein distance between the respective signatures.

We checked the accuracy of the alignment procedure by compiling some binaries with debug symbols (using gcc's `-g` flag), allowing the decompiler to determine the true variable names, and then attempting to recompute them using the alignment procedure. 67.6% of the variables were successfully aligned.

4.2 Machine Translation Framework

The aligned corpus is used to train the Moses SMT translation model. Moses is unaware of the syntactic structure of the program, so when the same variable reoccurs, Moses may provide different translations. Therefore, we evaluate each candidate by substituting it into the program and using the monolingual language model to compute the total probability. That gives us a measure of how well the candidate fits into the contexts where it is used. We select the candidate with the highest probability and apply it to all occurrences.

We improved performance by renaming the variables automatically prior to providing them to Moses. We tested several methods, including *type*, *argument position*, and *max entropy line*. In each case, we replaced the variable name with a hash of the specified attributes. Different renaming methods had different tradeoffs, since the more informative variable names are also less likely to appear in the training set, which prevents them from being successfully translated.

Table 1: Accuracy

Exact Match Accuracy	
No renaming	22.10%
Rename by type	21.80%
Rename by type, argument position	24.00%
Rename by type, max entropy line	22.10%
Rename by type, arg. position, max entropy line	23.50%
Approximate Match Accuracy	
No renaming	24.10%
Rename by type	25.20%
Rename by type, argument position	27.20%
Rename by type, max entropy line	25.40%
Rename by type, arg. position, max entropy line	26.60%

4.3 Evaluation

Our primary evaluation metric is the percentage of variable names correctly recovered in our test set, shown in Table 1, which reached up to 24% (±1.2% with 95% confidence). Counting approximate matches (manually verified), we are able to recover up to 27.2% of the original variables (±1.2% with 95% confidence).

58% of the decompiled variables had no "correct" variable name assigned to them by the alignment process. Some are spurious variables generated by the decompiler. These unaligned variables are ignored since it's unknown whether they were correctly renamed.

5 CONCLUSIONS

This seems to be a promising method for improving variable names in decompiled code. While the variable names generated certainly aren't perfect, they improve substantially on the arbitrary numeric identifiers generated by the decompiler. Intuitively, these variable names seem to be helpful, but future work could use human studies to quantify the effect of these improved identifier names on code comprehension.

Future work could also focus on improving the quality of the suggestions. For instance, replacing the statistical machine translation with a neural-based translation model would be a promising avenue for future exploration. We hope to make the current tool available online for experimentation and future research.

REFERENCES

[1] Miltiadis Allamanis, Earl T Barr, Christian Bird, and Charles Sutton. 2014. Learning natural coding conventions. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 281–293.
[2] Gal Beniamini, Sarah Gingichashvili, Alon Klein Orbach, and Dror G Feitelson. 2017. Meaningful identifier names: the case of single-letter variables. In *Proceedings of the 25th International Conference on Program Comprehension*. IEEE Press, 45–54.
[3] Pavol Bielik, Veselin Raychev, and Martin Vechev. 2015. Programming with "Big Code": Lessons, Techniques and Applications. In *1st Summit on Advances in Programming Languages (SNAPL 2015) (Leibniz International Proceedings in Informatics (LIPIcs))*, Thomas Ball, Rastislav Bodik, Shriram Krishnamurthi, Benjamin S. Lerner, and Greg Morrisett (Eds.), Vol. 32. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 41–50. https://doi.org/10.4230/LIPIcs.SNAPL.2015.41
[4] Brendan Cleary, Christoph Treude, Fernando Figueira Filho, Margaret-Anne Storey, and Martin Salois. 2013. Improving Tool Support for Software Reverse Engineering in a Security Context. In *International Conference on Augmented Cognition*. Springer, 113–122.
[5] Premkumar Devanbu. 2015. New initiative: the naturalness of software. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 543–546.
[6] MV Emmerik and Trent Waddington. 2004. Using a decompiler for real-world source recovery. In *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*. IEEE, 27–36.
[7] Georgios Gousios. 2013. The GHTorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, Piscataway, NJ, USA, 233–236. http://dl.acm.org/citation.cfm?id=2487085.2487132
[8] Ilfak Guilfanov. 2008. Decompilers and beyond. *Black Hat USA* (2008).
[9] Hex-Rays. 2017. Hex-Rays 2.4. (2017). https://www.hex-rays.com
[10] Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the naturalness of software. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 837–847.
[11] Svetoslav Karaivanov, Veselin Raychev, and Martin Vechev. 2014. Phrase-based statistical translation of programming languages. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*. ACM, 173–184.
[12] Donald Ervin Knuth. 1984. Literate programming. *Comput. J.* 27, 2 (1984), 97–111.
[13] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open Source Toolkit for Statistical Machine Translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions (ACL '07)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 177–180. http://dl.acm.org/citation.cfm?id=1557769.1557821
[14] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. 2006. What's in a Name? A Study of Identifiers. In *Proceedings of the 14th IEEE International Conference on Program Comprehension (ICPC '06)*. IEEE Computer Society, Washington, DC, USA, 3–12. https://doi.org/10.1109/ICPC.2006.51
[15] Veselin Raychev, Martin Vechev, and Andreas Krause. 2015. Predicting Program Properties from "Big Code". In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15)*. ACM, New York, NY, USA, 111–124. https://doi.org/10.1145/2676726.2677009
[16] Veselin Raychev, Martin Vechev, and Eran Yahav. 2014. Code completion with statistical language models. In *ACM SIGPLAN Notices*, Vol. 49. ACM, 419–428.
[17] Edward J. Schwartz, JongHyup Lee, Maverick Woo, and David Brumley. 2013. Native x86 Decompilation Using Semantics-preserving Structural Analysis and Iterative Control-flow Structuring. In *Proceedings of the 22Nd USENIX Conference on Security (SEC'13)*. USENIX Association, Berkeley, CA, USA, 353–368. http://dl.acm.org/citation.cfm?id=2534766.2534797
[18] Bogdan Vasilescu, Casey Casalnuovo, and Premkumar Devanbu. 2017. Recovering Clear, Natural Identifiers from Obfuscated JavaScript Names. In *12th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM.
[19] Khaled Yakdan, Sergej Dechand, Elmar Gerhards-Padilla, and Matthew Smith. 2016. Helping Johnny to Analyze Malware: A Usability-Optimized Decompiler and Malware Analysis User Study. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 158–177.
[20] Khaled Yakdan, Sebastian Eschweiler, Elmar Gerhards-Padilla, and Matthew Smith. 2015. No More Gotos: Decompilation Using Pattern-Independent Control-Flow Structuring and Semantic-Preserving Transformations.. In *NDSS*.