Mining Error-Handling Specifications for Systems Software

Daniel DeFreez
University of California, Davis
United States
dcdefreez@ucdavis.edu

ABSTRACT

This paper presents a technique for mining error-handling specifications from systems software. It presents a static analysis for detecting error handlers in low-level code, and it shows how function synonyms can be used to mine for error-handling specifications with only a few supporting examples.

CCS CONCEPTS

• **Software and its engineering → Automated static analysis**; **Error handling and recovery**;

KEYWORDS

program analysis, program embeddings, error handling, program comprehension, specification mining

ACM Reference Format:
Daniel DeFreez. 2018. Mining Error-Handling Specifications for Systems Software. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3236024.3275440

1 RESEARCH PROBLEM

When a programmer is writing systems error-handling code, it is often necessary to understand many implicit, undocumented specifications. The goal of this work is to automatically recover these error-handling specifications from real-world systems software such as the Linux kernel.

This task has two challenges. First, unlike languages such as Java with explicit error-handling features, error handlers in systems software typically use the return code idiom, where specific return value constants indicate that an error occurred [12]. The problem is to infer the error specifications for each function, i.e. the constants a function returns when a runtime error occurs. In Figure 1, for example, we see that the function `request_irq` returns a non-zero integer on error.

The second challenge is that error-handling specifications often involve very few examples, and mining techniques must rely on a large number of supporting examples lest they suffer from extreme false positive rates [13]. In Figure 1, after the calls to `pci_request_regions` and `pci_enable_device`, if `request_irq` returns an error the correct free function for that driver needs to be called in response. Because each driver defines its own free function, there exist only a handful of calls to those functions from which error-handling specifications can be inferred. Furthermore, the free functions themselves are semantically and syntactically dissimilar.

```
1  // Adapted, simplified from actual sound drivers
2  pci_request_regions(...);
3  pci_enable_device(...);
4  if (request_irq(...)) {
5    // error-only (need not be in same handler)
6    dev_err(msg);
7    // A call a driver-specific free function
8    snd_intel8x0_free OR snd_sonicvibes_free OR ...
9  }
```

Figure 1: Running example from PCI sound drivers. The code has been simplified; the contexts that the free functions are called from are not identical.

In a large code base there often exist functions that are not clones, but which serve the same purpose. We call these "function synonyms." Multiple implementations of similar drivers are rich sources of synonyms, of which the free functions in Figure 1 are an example. Function synonyms can be used to enhance the support of error-handling specifications by merging together multiple specifications which each have only a few supporting examples.

The contributions of this work are:

(1) A static analysis that efficiently infers function error specifications and detects error handlers.
(2) The Func2vec tool for producing source code embeddings and a dataset of function synonyms.
(3) An approach and tool for mining error-handling specifications using synonyms.

2 BACKGROUND AND RELATED WORK

Frequent Itemset Mining. Given a list of items, a database of itemset transactions, and a support threshold s, the goal of frequent itemset mining [3] is to return itemsets where the items in the itemset co-occur in at least s transactions. Of the wide range of specification inference techniques [7, 11], frequent itemset mining was chosen instead of sequence or automata mining because it imposes less stringent requirements on the form of the specification, better matching the error-handling specification mining problem.

Embeddings. Previous work [10] has used software embeddings for other applications, but Func2vec is the first to embed static program paths.

Error-Handling Specifications. A number of papers use normal paths to mine specifications for error-handling paths [2, 5, 15]. Observing that there are specifications which are *only* supported

ESEC/FSE '18, November 4–9, 2018, Lake Buena Vista, FL, USA

ACM ISBN 978-1-4503-5573-5/18/11.
https://doi.org/10.1145/3236024.3275440

by error paths, we focus our efforts on the problem of inferring error specifications for functions with few supporting examples.

Clone Detection. Clone detection is a different problem from synonym identification. Clone detection tools report sections of similar code, which in some cases contain calls to synonyms, but the synonyms within these context clones are not identified. In practice, we have found that tools such as Deckard [6] or MOSS [14] cannot be used to find synonyms.

3 APPROACH

3.1 Error Handler Detection

The error-handler detection analysis is bootstrapped with a small seed set $e_1, e_2, ..., e_n$ of error handlers. The predicate associated with the immediate parent of each handler in the control dependence graph is abstracted to a value $\alpha(e_i)$ in an interval lattice. The error specification of a function F, denoted $E(F)$ is the solution to the following set constraint problem.

(1) $E(F) \subseteq \alpha(e_1) \cup \alpha(e_2) \cup ... \cup \alpha(e_n)$
(2) $E(F) \subseteq \alpha(\text{ret}_F)$ where ret_F denotes the set of values a function can return, as determined statically.

Error handlers are identified by labeling each basic block with the constraints under which that block executes. Any block that executes under constraints that satisfy the error specification of any function is labeled as an error handler. Using just the knowledge that a single function (`dev_err`) is called exclusively on error paths, this approach detects 1,406 handlers in the Linux PCI sound drivers. These are the handlers which were mined to create the results in section 4. Previous work [4] relied on an existing dataflow analysis [12] to detect error handlers.

3.2 Function Synonym Identification

The technique used by Func2vec computes a map from each function to a vector in a continuous vector space such that vectors for function synonyms ar in close proximity, without any previous knowledge about naming conventions. For a given vocabulary L of program functions, Func2vec computes an embedding $\Phi : L \to \mathbb{R}^d$ that maps each program function $\ell \in L$ to a d-dimensional vector in $\mathbb{R}^d$. This embedding is computed by generating sentences from random walks over a pushdown system, modeling the set of valid interprocedural paths in the program. These walks comprise the training corpus.

Given the set of walks over the ℓ-PDS, Func2vec uses a neural network [9] to learn a vector representation for labels $\Phi : L \to \mathbb{R}^d$. Traditional language models try to estimate the probability of seeing a label ℓ_i given the context of the previous labels in the random walk; viz. $\Pr\left(\ell_i | \ell_1, \ell_2, \ldots, \ell_{i-1}\right)$. However, we also want to learn the distributed representation in the form of an embedding: $\Phi : L \to \mathbb{R}^d$. Thus, our problem is to estimate the likelihood: $\Pr\left(\ell_i | \Phi(\ell_1), \Phi(\ell_2), \ldots, \Phi(\ell_{i-1})\right)$.

3.3 Mining with Synonyms

An *error-handling specification* is defined as an association rule whose antecedent is the *specification context* and consequent is the *specification response.*

Definition 3.1. An *error-handling specification* S is defined as $C_S \to R_S$, where $C_S = \{c_1, c_2, \ldots, c_m\}$ is the context set of function calls for the specification S, and $R_S = \{r_1, r_2, \ldots, r_m\}$ is the response set of function calls for the specification S.

Function synonyms can be leveraged to increase the support of specifications that would otherwise be filtered out by any reasonable minimum support threshold. The support for an error-handling specification S is the size of the set of error handlers that support that specification, $|\text{Supp}(S)|$. For any two specifications $(S1, S2)$ where (F, F') is a pair of function synonyms such that $F \in C_{S1}$ and $F' \in C_{S2}$, or $F \in R_{S1}$ and $F' \in R_{S2}$, the specifications can be *merged* to yield a support value of $|\text{Supp}(S1) \cup \text{Supp}(S2)|$. Note that the supporting error handlers for two specifications can overlap, so $|\text{Supp}(S1) \cup \text{Supp}(S2)| \leq \text{Supp}(S1) + \text{Supp}(S2)$.

4 RESULTS

Function Synonyms. The Func2vec tool was evaluated [4] on how well the function synonyms it identifies comport with two gold standards. The first gold standard (manually reviewed) consists of 2,652 synonyms in 265 equivalence classes, created with the assistance of a Linux kernel developer. The second (underscores) consists of 2,017 function pairs that follow a strong naming convention. The Func2vec synonym equivalence classes in the Func2vec embedding were compared with these gold standards using two different metrics: AUROC [8] when the pairwise distance of vectors in the embedding are sorted, and the F1 score comparing KMeans clusters with the gold standards. On the manual gold standard the AUROC score was 0.73, and the cluster F1 score was also 0.73. On the underscores gold standard the AUROC score was 0.75, and the cluster F1 score was 0.58. These indicate that the Func2vec synonym equivalence classes significantly overlap the gold standards.

Error-Handling Specifications. Consider the two specifications from the motivating example in Figure 1. The support for the specification {`pci_request_regions`}, {`pci_enable_device`} → {`snd_intel8x0_free`} in Figure 1 increased from 3 to 42. The support for the specification {`pci_request_regions`, `pci_enable_device`} → {`snd_sonicvibes_free`} increased from 4 to 42. After sorting the specifications by support, the `intel8x0` specification is at position 2,134 without merging, and at position 47 with merging. The `sonicvibes` specification is at 1,098 without merging, and 43 after merging. This demonstrates that the error handling detection technique combined with function synonyms enables mining error-handling specifications, even when the specifications have only a few supporting examples.

The error-handling specifications mined using this technique have been useful in finding real bugs in the Linux kernel [1]. We reported bugs that were automatically found to be violations of the specification gfs2_holder_init → gfs2_holder_uninit. We supplied patches for these bugs that were accepted and merged into Linux 4.7.

ACKNOWLEDGMENTS

This work was supported by NSF grant CCF-1464439, and AWS Cloud Credits for Research.

REFERENCES

[1] 2016. [Cluster-devel] [PATCH] GFS2: Add calls to gfs2_holder_uninit in two error handlers. https://www.redhat.com/archives/cluster-devel/2016-April/msg00082.html. (2016). Accessed: 2018-06-29.

[2] Mithun Acharya and Tao Xie. 2009. Mining API Error-Handling Specifications from Source Code. In *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings (Lecture Notes in Computer Science)*, Marsha Chechik and Martin Wirsing (Eds.), Vol. 5503. Springer, 370–384. DOI: http://dx.doi.org/10.1007/978-3-642-00593-0_25

[3] Rakesh Agrawal, Ramakrishnan Srikant, and others. 1994. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, Vol. 1215. 487–499.

[4] Daniel DeFreez, Aditya V Thakur, and Cindy Rubio-González. 2018. Path-Based Function Embedding and its Application to Error-Handling Specification Mining. In *Proceedings of ESEC/FSE 2018*. To appear.

[5] Claire Le Goues and Westley Weimer. 2012. Measuring Code Quality to Improve Specification Mining. *IEEE Trans. Software Eng.* 38, 1 (2012), 175–190. DOI: http://dx.doi.org/10.1109/TSE.2011.5

[6] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stéphane Glondu. 2007. DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones. In *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007*. IEEE Computer Society, 96–105. DOI: http://dx.doi.org/10.1109/ICSE.2007.30

[7] Zhenmin Li and Yuanyuan Zhou. 2005. PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2005, Lisbon, Portugal, September 5-9, 2005*, Michel Wermelinger and Harald C. Gall (Eds.). ACM, 306–315. DOI: http://dx.doi.org/10.1145/1081706.1081755

[8] Roy A Maxion and Rachel R Roberts. 2004. *Proper use of ROC curves in Intrusion/Anomaly Detection*.

[9] Tomas Mikolov, Kai Chen, Greg S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. (2013). http://arxiv.org/abs/1301.3781

[10] Trong Duc Nguyen, Anh Tuan Nguyen, Hung Dang Phan, and Tien N. Nguyen. 2017. Exploring API embedding for API usages and applications. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, Sebastián Uchitel, Alessandro Orso, and Martin P. Robillard (Eds.). IEEE / ACM, 438–449. DOI: http://dx.doi.org/10.1109/ICSE.2017.47

[11] Martin P. Robillard, Eric Bodden, David Kawrykow, Mira Mezini, and Tristan Ratchford. 2013. Automated API Property Inference Techniques. *IEEE Trans. Software Eng.* 39, 5 (2013), 613–637. DOI: http://dx.doi.org/10.1109/TSE.2012.63

[12] Cindy Rubio-González, Haryadi S. Gunawi, Ben Liblit, Remzi H. Arpaci-Dusseau, and Andrea C. Arpaci-Dusseau. 2009. Error propagation analysis for file systems. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2009, Dublin, Ireland, June 15-21, 2009*. 270–280. DOI: http://dx.doi.org/10.1145/1542476.1542506

[13] Suman Saha, Jean-Pierre Lozi, Gaël Thomas, Julia L. Lawall, and Gilles Muller. 2013. Hector: Detecting Resource-Release Omission Faults in error-handling code for systems software. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Budapest, Hungary, June 24-27, 2013*. IEEE Computer Society, 1–12. DOI: http://dx.doi.org/10.1109/DSN.2013.6575307

[14] Saul Schleimer, Daniel Shawcross Wilkerson, and Alexander Aiken. 2003. Winnowing: Local Algorithms for Document Fingerprinting. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, June 9-12, 2003*, Alon Y. Halevy, Zachary G. Ives, and AnHai Doan (Eds.). ACM, 76–85. DOI: http://dx.doi.org/10.1145/872757.872770

[15] Westley Weimer and George C. Necula. 2004. Finding and preventing run-time error handling mistakes. In *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2004, October 24-28, 2004, Vancouver, BC, Canada*. 419–431. DOI: http://dx.doi.org/10.1145/1028976.1029011