Probabilistic Automated Language Learning for Configuration Files

Mark Santolucito Ennan Zhai Ruzica Piskac

Yale University

Abstract. Software failures resulting from configuration errors have become commonplace as modern software systems grow increasingly large and more complex. The lack of language constructs in configuration files, such as types and grammars, has directed the focus of a configuration file verification towards building post-failure error diagnosis tools. In addition, the existing tools are generally language specific, requiring the user to define at least a grammar for the language models and explicit rules to check. In this paper, we propose a framework which analyzes datasets of correct configuration files and derives rules for building a language model from the given dataset. The resulting language model can be used to verify new configuration files and detect errors in them. Our proposed framework is highly modular, does not rely on the system source code, and can be applied to any new configuration file type with minimal user input. Our tool, named ConfigC, relies on an abstract representation of language rules to allow for this modularity. ConfigC supports learning of various rules, such as orderings, value relations, type errors, or user defined rules by using a probabilistic type inference strategy and defining a small interface for the rule type.

1 Introduction

Configuration errors are one of the most important root causes of today's software system failures [40, 41]. In their empirical study Yin *et al.* [41] report that about 31% of system failures were caused by misconfiguration problems and only 20% were caused by bugs in program code. Misconfigurations, in practice, may result in various system-wide problems, such as security vulnerabilities, application crashes, severe disruptions in software functionality, and incorrect program executions [38, 39, 42, 44].

While many efforts have been proposed to check, troubleshoot, diagnose, and repair configuration errors [10,35,37], those tools mainly try to understand *what* caused the error – they are still not on a level of automatic verification tools used for regular program verification [17, 28, 31] that can detect errors without executing the code. Two main obstacles why we cannot simply apply the existing automatic tools and techniques to verification of configuration files are: 1) a lack of a specification which would describe properties of configuration files, and 2) a program structure of configuration files – they are mainly a sequence of entries assigning some value to system variables. The language in which configuration files are written does not adhere to a specific grammar or syntax. In particular, the entries in configuration files are untyped. Moreover, there are surprisingly few rules specifying constraints on entries and there is no explicit structure policy for the entries. Thus, automated verification of configuration files would be highly desirable [36, 40, 44].

To overcome these obstacles, researchers proposed approaches based on statistical analysis and learning [36, 42, 44] that try to infer rules and policies about how configuration files are constructed. Rather than explicitly specifying entries' types or rules, these efforts focused on learning policies from a sample dataset. The learned rules are usually constructed in the following way: for every entry in a configuration file, they check if it deviates from a "typical" value, *i.e.*, a value computed from a large training set consisting of configuration files. If the entry is significantly different from a typical value, they suspect that it could be a potential configuration error.

The downside of these is that their learning approaches are limited to simplistic configuration errors, such as type errors and syntax errors, or they heavily rely on template-based inference [44]. Many sophisticated configuration errors, which are difficult to template in practice, cannot be detected in this way. As an illustration of such an error, consider entry ordering errors, which happens when some entries are inserted in a wrong order. For example, if in a PHP configuration file an entry $\mathrm{extension} = \mathrm{mysql.so}$ appears before $\mathrm{extension} = \mathrm{recode.so}$, it would lead to a crash error, where the Apache server cannot start due to the segmentation fault error. The correct ordering should be $\mathrm{extension} = \mathrm{recode.so}$ before $\mathrm{extension} = \mathrm{mysql.so}$ [41]. These types of errors cannot be detected by existing learning efforts, since it is hard to build a corresponding template [40].

In this paper, we present ConfigC, the first automatic verification framework for general software configurations. ConfigC is based on a collection of powerful learning algorithms that do not necessarily depend on templates. The learning process takes as input a large sample of configuration files. The process is language-agnostic and works for any kind of configuration file, but all of the files in the sample need to be of the same kind (such as MySQL or HTTPD configuration files). From that sample, ConfigC learns an abundant set of rules specifying various properties that hold on the given sample. The files in the sample might contain errors – we are, therefore, using probabilistic learning to derive a set of accurate rules.

For practical purposes we use a real-world dataset [2]. Every file in our dataset contains several errors, but they are typically different errors and only appear in a small percentage of files. Using that insight we were able, with the help of the probabilistic cutoff, to learn an accurate set of rules. The rules, in general, specify which properties variables need to satisfy. One can see this learning process as a way of deriving a specification for configuration files. Once there is a specification, we can do formal verification. With these rules we can efficiently check the correctness of the configuration files of interest and detect potential errors.

The learning process has two phases. In the first phase, ConfigC analyzes the sample dataset and generates a well-structured and probabilistically-typed intermediate representation. In the second phase, ConfigC derives rules and constraints by analyzing the intermediate representations. Every learned rule is annotated with a probability depicting which percentage of the files the rule was seen as correct. We only accept those rules whose correctness probability is above the given threshold.

Finally, while the learned rules establish correlations between two or more variables, using ConfigC, one can also detect whether a single variable has an appropriate value

assigned to it. We do that by measuring the values that are typically assigned to that variable and reporting if the current value deviates too much from the recorded values.

Building such an automatic verification framework for configuration files, nevertheless, requires addressing several challenges. First, we need to assign a type to every variable in a configuration file during the transformation phase. However, the type of a variable cannot always be fully determined from a single value. For example, an entry $\mathrm{temp\_dir} = 300$ assigns to variable $\mathrm{temp\_dir}$ either a directory named "300" or sets the size of that directory to 300 (an integer). Some existing type inference work would report this is an error, because $\mathrm{temp\_dir}$ should be assigned an integer [44]. We address this problem by introducing the concept of *probabilistic types*. Rather than assigning only one variable to a single type, we assign several types over a probability distribution. The entry in the above example might be assigned the following probabilistic type $\{\mathrm{temp\_dir},\ 300,\ [(\mathrm{File},\ 60\%),\ (\mathrm{Int},\ 40\%)]\}$. Using probabilistic types, we can generate a more accurate language model, thus significantly improving our verification capabilities.

Second, when learning rules we use very general templates to infer them. The user does not need to provide any templates – they are internally associated to the types. Nevertheless, in addition to those general templates, we still need specific algorithms to learn rules that cannot be easily templated.

From a practical perspective, ConfigC introduces no additional burden to the user: they can simply use ConfigC to check for errors in their configuration files. However, they can also easily extend the framework themselves. The system is designed to be highly modular. If there is a class of rules that ConfigC is not currently learning, the user can develop her own templates and learners for that class. The new learner can be added to ConfigC and this way it can check an additional new set of errors.

Our ConfigC prototype still has many limitations: for example, we cannot handle configuration errors that can be triggered during system execution time. Nevertheless, we believe ConfigC may suggest a practical path toward automatic and modular language-based configuration verification. To summarize, this tool paper makes the following contributions:

1. We propose the first automatic configuration verification framework, ConfigC, that can learn a language model from a sample dataset, and then use this language model to verify configuration files of interest.
2. ConfigC proposes probabilistic types to assign a confidence distribution over a set of types to each entry, while generating the intermediate representation.
3. ConfigC employs a collection of machine learning algorithms to enable powerful rule and constraint inference.
4. ConfigC is capable of detecting various tricky errors that cannot be detected by previous efforts, including entry ordering errors, fine-grained value correlation errors, missing entry errors, and environment-related errors.
5. We implement a ConfigC prototype and evaluate it by conducting comprehensive experiments on real-world dataset.

Acknowledgements

We thank the anonymous reviewers for their insightful comments. We also thank Tianyin Xu for his valuable feedback on earlier version of this work. This research was supported by the NSF under grant CCF-1302327.

References

1. Fine-grained value correlation error, http://serverfault.com/questions/628414/my-cnf-configuration-in-mysql-5-6-x
2. Misconfiguration dataset, https://github.com/tianyin/configuration_datasets
3. OpenStack, http://www.openstack.org/
4. parallel-3.2.1.0: Parallel programming library, https://hackage.haskell.org/package/parallel-3.2.1.0/docs/Control-Parallel-Strategies.html
5. Problem moving a MySQL data directory, http://serverfault.com/questions/281217/problem-moving-a-mysql-data-directory-to-a-new-drive
6. Singular value error, http://stackoverflow.com/questions/1980004/2006-mysql-server-has-gone-away-error-in-wamp
7. Agrawal, R., Evfimievski, A.V., Srikant, R.: Information sharing across private databases. In: ACM SIGMOD (Jun 2003)
8. Aguilera, M.K., Mogul, J.C., Wiener, J.L., Reynolds, P., Muthitacharoen, A.: Performance debugging for distributed systems of black boxes. In: 19th ACM Symposium on Operating Systems Principles (SOSP) (Oct 2003)
9. Attariyan, M., Chow, M., Flinn, J.: X-ray: Automating root-cause diagnosis of performance anomalies in production software. In: 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (Oct 2012)
10. Attariyan, M., Flinn, J.: Automating configuration troubleshooting with dynamic information flow analysis. In: 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (Oct 2010)
11. Bahl, P., Chandra, R., Greenberg, A.G., Kandula, S., Maltz, D.A., Zhang, M.: Towards highly reliable enterprise network services via inference of multi-level dependencies. In: ACM SIGCOMM (SIGCOMM) (Aug 2007)
12. Barham, P., Donnelly, A., Isaacs, R., Mortier, R.: Using Magpie for request extraction and workload modelling. In: 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (Dec 2004)
13. Basescu, C., Cachin, C., Eyal, I., Haas, R., Sorniotti, A., Vukolic, M., Zachevsky, I.: Robust data sharing with key-value stores. In: 42nd IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (Jun 2012)
14. Benson, T., Akella, A., Maltz, D.A.: Network traffic characteristics of data centers in the wild. In: Internet Measurement Conference (IMC) (Nov 2010)
15. Bessani, A.N., Correia, M.P., Quaresma, B., André, F., Sousa, P.: DepSky: Dependable and secure storage in a cloud-of-clouds. In: 6th ACM European Conference on Computer Systems (EuroSys) (Apr 2011)
16. Blundo, C., de Cristofaro, E., Gasti, P.: EsPRESSo: Efficient privacy-preserving evaluation of sample set similarity. vol. 22, pp. 355–381 (2014)
17. Bobot, F., Filliâtre, J., Marché, C., Paskevich, A.: Let's verify this with why3. STTT 17(6), 709–727 (2015)
18. Bonvin, N., Papaioannou, T.G., Aberer, K.: A self-organized, fault-tolerant and scalable replication scheme for cloud storage. In: ACM Symposium on Cloud Computing (SoCC) (Jun 2010)

19. Broder, A.Z.: On the resemblance and containment of documents. In: Compression and Complexity of Sequences (SEQUENCES) (Jun 1997)
20. Chen, M.Y., Accardi, A., Kiciman, E., Patterson, D.A., Fox, A., Brewer, E.A.: Path-based failure and evolution management. In: 1st USENIX Symposium on Networked System Design and Implementation (NSDI) (Mar 2004)
21. Chen, X., Mao, Y., Mao, Z.M., van der Merwe, J.E.: Declarative configuration management for complex and dynamic networks. In: ACM CoNEXT (CoNEXT) (Nov 2010)
22. Chen, X., Zhang, M., Mao, Z.M., Bahl, P.: Automating network application dependency discovery: Experiences, limitations, and new solutions. In: 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (Dec 2008)
23. Dobrescu, M., Argyraki, K.J.: Software dataplane verification. In: 11th USENIX Symposium on Networked System Design and Implementation (NSDI) (Apr 2014)
24. Enck, W., McDaniel, P.D., Sen, S., Sebos, P., Spoerel, S., Greenberg, A.G., Rao, S.G., Aiello, W.: Configuration management at massive scale: System design and experience. In: USENIX Annual Technical Conference (USENIX ATC) (Jun 2007)
25. Huang, P., Bolosky, W.J., Singh, A., Zhou, Y.: Confvalley: A systematic configuration validation framework for cloud services. In: 10th European Conference on Computer Systems (EuroSys) (Apr 2015)
26. Kissner, L., Song, D.X.: Privacy-preserving set operations. In: 25th Annual International Cryptology Conference (CRYPTO). Springer (Aug 2005)
27. Launchbury, J., Jones, S.L.P.: Lazy functional state threads. In: Programming Language Design and Implementation (PLDI). pp. 24–35. ACM Press (1993)
28. Leino, K.R.M.: Dafny: An automatic program verifier for functional correctness. In: Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16. pp. 348–370 (2010)
29. Loo, B.T., Hellerstein, J.M., Stoica, I., Ramakrishnan, R.: Declarative routing: Extensible routing with declarative queries. In: ACM SIGCOMM (SIGCOMM) (Aug 2005)
30. Lopes, N.P., Bjørner, N., Godefroid, P., Jayaraman, K., Varghese, G.: Checking beliefs in dynamic networks. In: 12th USENIX Symposium on Networked System Design and Implementation (NSDI) (May 2015)
31. Piskac, R., Wies, T., Zufferey, D.: Grasshopper - complete heap verification with mixed specifications. In: Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014. pp. 124–139 (2014)
32. Raychev, V., Bielik, P., Vechev, M.T., Krause, A.: Learning programs from noisy data. In: 43rd ACM SIGPLAN-SIGACT (POPL) Symposium on Principles of Programming Languages (Jan 2016)
33. Raychev, V., Vechev, M.T., Krause, A.: Predicting program properties from "big code". In: 42nd ACM SIGPLAN-SIGACT (POPL) Symposium on Principles of Programming Languages (Jan 2015)
34. Santolucito, M., Zhai, E., Piskac, R.: Probabilistic automated language learning for configuration files. In: 28th Computer Aided Verification (CAV) (Jul 2016)
35. Su, Y., Attariyan, M., Flinn, J.: AutoBash: Improving configuration management with operating systems. In: 21st ACM Symposium on Operating Systems Principles (SOSP) (Oct 2007)
36. Wang, H.J., Platt, J.C., Chen, Y., Zhang, R., Wang, Y.: Automatic misconfiguration troubleshooting with PeerPressure. In: 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (Dec 2004)
37. Whitaker, A., Cox, R.S., Gribble, S.D.: Configuration debugging as search: Finding the needle in the haystack. In: 6th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (Dec 2004)

38. Xu, T., Jin, L., Fan, X., Zhou, Y., Pasupathy, S., Talwadker, R.: Key, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In: 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE) (Aug 2015)
39. Xu, T., Zhang, J., Huang, P., Zheng, J., Sheng, T., Yuan, D., Zhou, Y., Pasupathy, S.: Do not blame users for misconfigurations. In: 24th ACM Symposium on Operating Systems Principles (SOSP) (Nov 2013)
40. Xu, T., Zhou, Y.: Systems approaches to tackling configuration errors: A survey. ACM Comput. Surv. 47(4), 70 (2015)
41. Yin, Z., Ma, X., Zheng, J., Zhou, Y., Bairavasundaram, L.N., Pasupathy, S.: An empirical study on configuration errors in commercial and open source systems. In: 23rd ACM Symposium on Operating Systems Principles (SOSP) (Oct 2011)
42. Yuan, D., Xie, Y., Panigrahy, R., Yang, J., Verbowski, C., Kumar, A.: Context-based online configuration-error detection. In: USENIX Annual Technical Conference (USENIX ATC) (Jun 2011)
43. Zhai, E., Chen, R., Wolinsky, D.I., Ford, B.: Heading off correlated failures through Independence-as-a-service. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI) (Oct 2014)
44. Zhang, J., Renganarayana, L., Zhang, X., Ge, N., Bala, V., Xu, T., Zhou, Y.: Encore: Exploiting system environment and correlation information for misconfiguration detection. In: Architectural Support for Programming Languages and Operating Systems (ASPLOS) (Mar 2014)