New Applications of Software Synthesis: Verification of Configuration Files and Firewall Repair

Ruzica Piskac*
Yale University

Abstract. The main goal of software synthesis is to automatically derive code from a given specification. The specification can be either explicitly written, or specified through a couple of representative examples illustrating the user's intent. However, sometimes there is no specification and we need to infer the specification from a given environment. This paper present two such efforts.
We first show, using verification for configuration files, how to learn specification when the given examples is actually a set of configuration files. 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. We describe a framework which analyzes data sets of correct configuration files and derives rules for building a language model from the given data set. The resulting language model can be used to verify new configuration files and detect errors in them.
We next describe a systematic effort that can automatically repair firewalls, using the programming by example approach. Firewalls are widely employed to manage and control enterprise networks. Because enterprise-scale firewalls contain hundreds or thousands of policies, ensuring the correctness of firewalls – whether the policies in the firewalls meet the specifications of their administrators – is an important but challenging problem. In our approach, after an administrator observes undesired behavior in a firewall, she may provide input/output examples that comply with the intended behavior. Based on the given examples, we automatically synthesize new firewall rules for the existing firewall. This new firewall correctly handles packets specified by the examples, while maintaining the rest of the behavior of the original firewall.

Keywords: Software synthesis · program repair · verification · configuration files · firewalls

*This research was sponsored by the NSF under grants CCF-1302327, CFF-1553168, and CCF-1715387

1 Introduction

Software synthesis has the potential to transform the software development process, by eliminating software errors before they even occur. The essence of software synthesis is that a programmer must only state *what* should be done, and not *how* it should be done. Instead of writing code manually, a programmer provides a specification and the synthesis tool automatically generates code that satisfies the specification. Consequently, the generated code is correct by construction, thereby avoiding many of the potential errors that could creep into manually written code.

Recent work in this area has focused on manipulating fundamental data types such as strings [5,9,14], lists [4,10] and numbers [15]. The success and impact of this line of work can be estimated from the fact that some of this technology [5] ships as part of the popular Flash Fill feature in Excel 2013 [16].

A common thread to all those tools that they take a given specification and automatically generate code corresponding to that specification. The specification can be explicitly written, in which case we are talking about complete specification, or it can be given in the form of input/output examples. The user provides those examples and they should be chosen so that they illustrate the user's intentions. Nevertheless, this is still an incomplete specification, which means that there might be many programs satisfying the given examples. This type of software synthesis is well known under the name *programming by example* [3,6,8].

In this paper we describe two application domains for software synthesis that have not been not previously studied by the community working in formal methods and verification. One topic is synthesis of a specification for configuration files (Sec. 2, previously published in [12,13]). The second project introduces so-called *repair by example*, which is used for verification and repair of firewall programs (Sec. 3, previously published in [7]).

2 Verification of Configuration Files

Configuration errors (also known as misconfigurations) have become one of the major causes of system failures, resulting in security vulnerabilities, application outages, and incorrect program executions [17, 18]. In 2015 Facebook, Tinder, and Instagram all became inaccessible for approximately 52 minutes. A Facebook spokeswoman reported that this was caused by a change to the site's configuration system [11]. These critical system failures are not rare – a software system failures study [19] reports that about 31% of system failures were caused by configuration errors. This is even higher than the percentage of failures resulting from program bugs (20%).

A recent study [2] showed that in 2016 software errors cost the United States economy approximately $1.1 trillion. Detecting and preventing software errors plays a major role in a development process, with programmers using techniques like testing, debugging, and verification. However, none of these techniques can

be applied to finding errors in configuration files. Effective testing of configuration files is difficult because errors may arise only under certain, hard to simulate conditions, such as heavy traffic loads. Another approach to finding errors in program code is software verification, which has been applied to many complex systems (for example, operating systems, compilers). However, traditional verification techniques cannot be applied to configuration files, because these techniques rely on formal specifications describing the expected behavior of the program. The difficulty with configuration files is that they are mostly simple text files of keywords and values, and there is no traditional sense of a specification. With no formal specification of correctness or semantic program information, verifying configuration files is far outside the scope of existing technologies.

Modern verification technologies inherently depend on the availability of formal specifications, yet they are extremely labor intensive to create and maintain. This is especially the case for configuration files, which rarely have any documentation, even in written English form. We proposed and developed the first tool that can synthesize complex configuration specifications. Our tool combines knowledge discovery techniques with automated reasoning to synthesize constraint models of configuration files.

In the first prototype of our tool, ConfigC [13], we analyzed existing correct configuration files and learned properties that always hold in those configuration files. Some examples of the properties are ordering constraints (e.g., one library should be loaded before another), type constraints (e.g., which keywords act as Boolean flags), and size constraints (e.g., that some memory size always needs to be bounded by some other). Once we have learned such constraints/specifications, ConfigC can verify users configuration files, and report any violations to the user.

We extended this work to a more advanced tool ConfigV [12], which is the first tool that can automatically detect complex errors involving multiple variables, and learn over a training set of partially incorrect configuration files. ConfigV required two core theoretical advances; the first was the introduction of probabilistic types, and the second was an extension to association rule learning [1]. Since configuration files lack helpful semantic information to infer types, we use a probabilistic inference method to learn likely types for keywords based on their values from the training set. We combined this new type information with a generalization of association rule learning that handles not just association rules, but arbitrary, typed predicates.

We evaluated ConfigV by verifying public configuration files on GitHub, and we showed that ConfigV can successfully detect configuration errors in these files.

3 Verification and Repair of Firewalls

Firewalls play an important role in today's individual and enterprise-scale networks. A typical firewall is responsible for managing all incoming and outgoing

traffic between an internal network and the rest of the Internet. The firewall accepts, forwards, or drops packets based on a set of rules specified by its administrators. Because of the central role firewalls play in networks, small changes can propagate unintended consequences throughout the network.

We proposed and developed the first framework, called FireMason [7], `https://github.com/BillHallahan/FireMason`, that not only detects errors in firewall behaviors, but also automatically repairs the firewall. Broadly speaking, a firewall is correct if the rules of that firewall meet the specification of its administrator. While existing tools can identify the cause of an error, administrators still have to manually find an effective repair to the firewall so that it meets the specification. We introduced the concept called *repair by example*. Specifically, a user provides a list of examples of packets and desired routing (e.g., all packets with a certain source IP address should be dropped) to describe the desired behavior of the firewall. The current firewall might or might not route the packets as specified in the examples, but FireMason automatically synthesizes a new firewall that is guaranteed to satisfy the examples. Given the complexity of enterprise-scale networks, finding such a repair requires considerable expertise on the part of the administrator. To the best of our knowledge, there is no other existing effort that automates firewall repairs by examples. The concept of "repair by example" was motivated by the standard practice of how users ask for help with repairing their firewalls. On user forums, users would provide their firewalls and then list a couple of illustrative examples to show how the behavior should change.

The main challenges in firewall repair and verification is that adding a new rule might fix the current problem, but entirely break the behavior on some packages that the user might not have considered. To ensure the correctness of the repair we use techniques from formal methods. We translated a firewall into the formal mathematical language of first-order logic. This allows us to use existing SMT solvers which can automatically reason about these logics. As an illustration, checking if the repair broke some of previously correct behavior reduces to checking a formulas entailment.

By using SMT solvers, FireMason can provide formal guarantees that the repaired firewalls satisfy two important properties:

- Those packets described in the examples will be routed in the repaired firewall, as specified in provided examples.
- All other packets will be routed by the repaired firewall exactly as they were in the original firewall.

Taken together, these two properties allow administrators confidence that the repairs had the intended effect.

By using our formalism we are able to check some important and widely used, but previously out-of-scope, properties, including rate limits. Rate limits, which are frequently used in modern firewalls, put a restriction on the number of packets matched in a given amount of time. Such rules say, for example, that we can only accept 6 packets per minute from a certain IP address. As before, the user provides a list of examples, but with relative times. This requires reasoning

about the priorities and permissions of each firewall entry, as well as the temporal patterns of the incoming packets.

In addition to repairing, FireMason is also a stand-alone verification tool. For a given specification, such a checking if a certain packet will be rejected, FireMason can either prove that it holds, or it produce counterexamples.

We evaluated our tool using real-world firewall issues from user forums. We observed that FireMason is able to efficiently generate correct firewalls meeting administrators' examples, without introducing any new problems. In addition, our evaluation shows that FireMason scales well to enterprise-scale networks.

4 Conclusions

More details about the presented projects can be found in [7, 12, 13].

Two presented projects demonstrated that software synthesis can be successfully applied to problems, such as repair of firewalls and verification of configuration files, which are usually tackled by a system research community. One of the main obstacles was that often the specification does not exist and needs to inferred from the given context or provided examples. In our experience finding a suitable formalism to model the problem and efficiently solve real world instances is crucial. Both presented tools were successfully tested on the real world examples, which motivates us to further pursue addressing non-traditional synthesis problems.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. SIGMOD Rec. **22**(2), 207–216 (Jun 1993), `http://doi.acm.org/10.1145/170036.170072`
2. Cohane, R.: Financial cost of software bugs. `https://medium.com/@ryancohane/financial-cost-of-software-bugs-51b4d193f107` (2017)
3. Cypher, A., Halbert, D.: Watch what I Do: Programming by Demonstration. MIT Press (1993)
4. Feser, J.K., Chaudhuri, S., Dillig, I.: Synthesizing data structure transformations from input-output examples. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015. pp. 229–239 (2015)
5. Gulwani, S.: Automating string processing in spreadsheets using input-output examples. In: POPL. pp. 317–330 (2011)
6. Gulwani, S.: Synthesis from examples: Interaction models and algorithms. 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (2012), Invited talk paper
7. Hallahan, W.T., Zhai, E., Piskac, R.: Automated repair by example for firewalls. In: 2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017. pp. 220–229 (2017), `https://doi.org/10.23919/FMCAD.2017.8102263`
8. Lieberman, H.: Your Wish Is My Command: Programming by Example. Morgan Kaufmann (2001)

9. Menon, A.K., Tamuz, O., Gulwani, S., Lampson, B.W., Kalai, A.: A machine learning framework for programming by example. In: ICML (1). pp. 187–195 (2013)
10. Osera, P., Zdancewic, S.: Type-and-example-directed program synthesis. In: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015. pp. 619–630 (2015)
11. Ryall, J.: Facebook, Tinder, Instagram suffer widespread issues. http://mashable.com/2015/01/27/facebook-tinder-instagram-issues/ (2015)
12. Santolucito, M., Zhai, E., Dhodapkar, R., Shim, A., Piskac, R.: Synthesizing configuration file specifications with association rule learning. PACMPL **1**(OOPSLA), 64:1–64:20 (2017), http://doi.acm.org/10.1145/3133888
13. Santolucito, M., Zhai, E., Piskac, R.: Probabilistic automated language learning for configuration files. In: Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II. pp. 80–87 (2016), https://doi.org/10.1007/978-3-319-41540-6_5
14. Singh, R., Gulwani, S.: Learning semantic string transformations from examples. PVLDB **5** (2012)
15. Singh, R., Gulwani, S.: Synthesizing number transformations from input-output examples. In: CAV. pp. 634–651 (2012)
16. Flash Fill (Microsoft Excel 2013 feature), http://research.microsoft.com/users/sumitg/flashfill.html
17. Xu, T., Jin, L., Fan, X., Zhou, Y., Pasupathy, S., Talwadker, R.: Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software. In: The 10th ESEC/FSEJoint Meeting on Foundations of Software Engineering (Aug 2015)
18. Xu, T., Zhang, J., Huang, P., Zheng, J., Sheng, T., Yuan, D., Zhou, Y., Pasupathy, S.: Do not blame users for misconfigurations. In: The 24th SOSPACM Symposium on Operating Systems Principles (Nov 2013)
19. 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: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. pp. 159–172. SOSP '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2043556.2043572