# Static Analysis of File Manipulation Scripts

Rodney Rodriguez and Xiaoyin Wang
*Department of Computer Science, University of Texas at San Antonio*
{*rodney.rodriguez, xiaoyin.wang*}*@utsa.edu*

***Abstract*—Various scripts are often used to automatically build, configure, and deploy software, as well as to combine code base written in different programming languages. In this project, we develop novel analysis techniques for file manipulation scripts to statically detect path-related errors and estimate the necessary directory structure (path pre-conditions) before running a script. Due to the prevalence of path generation and file manipulation operations in various scripts, our proposed technique may enhance the reliability and transportability of these scripts.**

## 1. Introduction

Path generation and file manipulation account for a large portion of these scripts and path-related bugs [1] such as “file not found” and “permission denied” are common in the field. Furthermore, when transporting such scripts to other machines (including virtual machines and docker containers), it is necessary to extract their path pre-condition, which can be tedious and error-prone. While existing static string analyses [2], [3], [4] can estimate values of path variables (variables representing file paths) as automata and summarize string operations, they cannot check states of directory structures or summarize file operations (e.g., cp, rm). Unlike memory objects whose structure are often predefined in the code as class / struct definitions, directory structure can be manipulated at runtime and no existing techniques can statically check the correctness of directory structure states and operations on them.

In this paper, we describe the first technique to statically estimate the states of the file system at certain program points of a file-manipulation script. There are three major challenges. The first challenge is how to design proper abstract domains to represent the state of directory structures. We further need to represent file permissions and links to detect related errors. We overcome this challenge with the novel intuition: a state of the file system can be represented as a set of paths (strings), so an estimation of the file-system state can be represented as an estimation of its corresponding path set. The second challenge is how to represent file operations as transfer functions on the abstract domain. To overcome this challenge, we use FSTs (Finite State Transducers) to summarize the effect of file operations (e.g., cp, rm) on directory structures (path sets). The third challenge is the large variety of scripting languages and system / third-party commands that have effect on the file system. To make our technique general enough, we designed a new intermediate language (FIle Manipulation Intermediate Language, or FMIL) that abstracts control flows, string operations (e.g., concatenation, replace) and basic file operations (e.g., cp, mkdir) from the programming language to be analyzed. The benefit of FIMIL is to have a common input format for our analysis so that it can support any scripts that can be translated to FMIL.

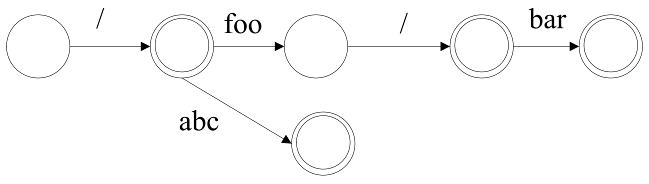

Figure 1. File-System-State Automaton

## 2. Background

There are research efforts on checking the correctness of build and configuration scripts such as Javascript [5] or Perl [6]. There are also research efforts on checking the correctness of build and configuration scripts, such as makefiles [7] and puppet scripts [8]. Some recent research [9], [10], [11] studied the automatic repair and generation of build scripts [12]. Adams et al. [13] proposed a framework to extract a dependency graph for build configuration files, and provide automatic tools to keep consistency during revision. However, these analyses mostly perform whole program analysis or bottom-up analysis on one type of code with conservative assumptions of entry states, while our approach is more of hybrid analyses that take advantage of concrete execution records. Furthermore, they focus on the internal logic or global variables, while our work further handles the program’s interaction with system environment and system environment states.

## 3. Analysis Framework

The basic idea behind our research is that file-system states can be presented as a set of path values (strings). To handle infinite string value sets, we use automata to summarize all possible file-system states. For example, Figure 1 shows the automaton presentation of a file system state with root folder “/”, folder “/foo”, and files “/foo/bar” and “/abc”. We have implemented File Manipulation Intermediate Language (FMIL) as the preliminary basis for the research. FMIL is currently supported with a parser, an abstraction interpretation engine (based on fixed point

2473-3105/19/$31.00 ©2019 IEEE
DOI 10.1109/PRDC47002.2019.00039

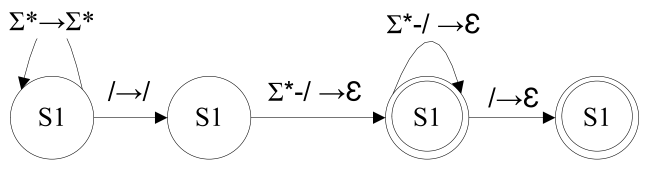

Figure 2. The FST for $parentDir$ Operation

algorithm), and a simple static path analysis which considers only file operations with constant paths as their arguments.

In the scripts, path arguments are often generated from string constants with string operations. We can use string analysis to estimate path argument's possible values as an automaton, but then we need to feed this automaton to the file-state transfer functions, which makes them much more complicated. As an example, for the operation "cp x y", and automaton aut(x) representing possible values of x, automaton aut(y) representing possible values of y, the transfer function from old file-system abstraction D to new abstraction D' will be:

$$D' = D \cup (aut(y)./.replace(D \cap aut(x).*, \\ parentdir(aut(x)),''))\quad (1)$$

, in which "." represents string concatenations, replace(a, b, c) is the standard string operation to replace appearances of regular expression b in automaton a to string c, which is supported by a Finite State Transducer (FST). And parentDir(a) is the string operation to get parent directory of a path a (substring until last "/" in a or second last "/" in a if a ends with "/"). This operation is straightforward for a constant string input, but for an automaton input I, we need to carefully design an FST (as shown in Figure 2) to transform I to a correct output automaton O, so that . For rm operations it is more complicated. As string analysis always over-estimates the paths to be deleted, if we simply do a difference between the automata, the resulted file-system state will no longer be an over-estimation.

Although our research is fully based on the above intuitive ideas. Designing a full-fledged analysis for file-system states has a lot of complications. We specifically identified risks and challenges from three aspects, and we list our mitigation plans as follows.

- Modeling of File Links. File system may contain link files that link to a target file or directory at a different path. This will cause some operations on the link file to be actually applied to the target file or directory. So, we need to model links between paths in a file-system state. Since the arguments of link-generation operations can be string variables, we plan to use a domain of automaton pairs to present the links between possible values of the arguments.
- Modeling File Permissions. Existence is the basic property of a path value. Permission is another important property of a path value. We plan to handle permissions by enriching each acceptance state in the automata with tags indicating the upper and lower permission bound of path values accepted at the state. When a permission changing (e.g. chmod) or checking operation is performed, we will intersect the argument automata with the file-system-sate automata and trace matching states to find out the tags of which acceptance states should be changed.
- Modeling File Co-Existence. Paths that may exist in a file-system state may not exist together. For more precise analysis, we need to summarize co-existence relationships between paths. We plan to maintain two state-pair domains for the automatons to model the file pairs that may co-exist or may not co-exist.

## References

[1] Y. Song, X. Wang, T. Xie, L. Zhang, and H. Mei, "JDF: detecting duplicate bug reports in jazz," in *Proc. ICSE, Volume 2*, 2010, pp. 315–316.

[2] H. Zhang, H. B. K. Tan, L. Zhang, X. Lin, X. Wang, C. Zhang, and H. Mei, "Checking enforcement of integrity constraints in database applications based on code patterns," *Journal of Systems and Software*, vol. 84, no. 12, pp. 2253–2264, 2011.

[3] H. Zhong and X. Wang, "Boosting complete-code tool for partial program," in *Proc. ASE*, 2017, pp. 671–681.

[4] X. Wang, L. Zhang, T. Xie, H. Mei, and J. Sun, "Locating need-to-externalize constant strings for software internationalization with generalized string-taint analysis," *Software Engineering, IEEE Transactions on*, vol. 39, no. 4, pp. 516–536, 2013.

[5] M. Madsen, B. Livshits, and M. Fanning, "Practical static analysis of javascript applications in the presence of frameworks and libraries," in *Proc. FSE*, 2013, pp. 499–509.

[6] D. Evans and D. Larochelle, "Improving security using extensible lightweight static analysis," *IEEE software*, no. 1, pp. 42–51, 2002.

[7] A. Tamrawi, H. A. Nguyen, H. V. Nguyen, and T. N. Nguyen, "Build code analysis with symbolic evaluation," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 650–660.

[8] R. Shambaugh, A. Weiss, and A. Guha, "Rehearsal: a configuration verification tool for puppet," in *ACM SIGPLAN Notices*, vol. 51, no. 6. ACM, 2016, pp. 416–430.

[9] F. Hassan and X. Wang, "Hirebuild: an automatic approach to history-driven repair of build scripts," in *Proceedings of the 40th International Conference on Software Engineering*. ACM, 2018, pp. 1078–1089.

[10] X. W. Foyzul Hassan, Rodney Rodgriguez, "Rudsea: Recommending updates of dockerfiles via software environment analysis," in *International Conference on Automated Software Engineering (ASE), New Idea Paper, To Appear*, 2018.

[11] F. Hassan, S. Mostafa, E. S. Lam, and X. Wang, "Automatic building of java projects in software repositories: A study on feasibility and challenges," in *Empirical Software Engineering and Measurement (ESEM), 2017 ACM/IEEE International Symposium on*. IEEE, 2017, pp. 38–47.

[12] F. Hassan and X. Wang, "Mining readme files to support automatic building of java projects in software repositories: Poster," in *Proceedings of the 39th International Conference on Software Engineering Companion*, 2017, pp. 277–279.

[13] B. Adams, H. Tromp, K. De Schutter, and W. De Meuter, "Design recovery and maintenance of build systems," in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, 2007, pp. 114–123.