Process Fuzzing as an Approach for Genetic Programming

Tom Wallis & Tim Storer

w.wallis.1@research.gla.ac.uk & timothy.storer@glasgow.ac.uk
School of Computing Science, University of Glasgow

Abstract. Genetic Programming (GP) has recently seen a growing application in the area of writing and improving computer programs. Generally, for experiments in this area, bespoke tools are constructed to perform research. In this paper, it is demonstrated that GP behaviour can be achieved via *process fuzzing*, and an implementation of the adaptation of ASTs for GP behaviour in the process fuzzing tool PyDySoFu is described.

1 Evolving Programs

Genetic Programming (GP) is a very well-established and promising field which has returned impressive results in a number of areas. The field has spawned a number of similar approaches which, while based on the same underlying concept, attempt an evolution-based solution in novel ways. Cartesian GP[6] uses a directed graph to represent a solution to a problem, and has seen great success, for its ability to converge on an acceptable solution in a relatively short number of generations. Stack-based GP[7] employs the use of multiple different stacks so as to work with state and keep track of multiple values, which can be difficult for the traditional tree-based genetic programming (TGP) approach.

1.1 Genetic Improvement

Generally, variants of GP present methods for improving mathematical-looking functions against some fitness function. However, variants have begun to arise which, rather than mutating some abstract representation of a program, mutate the program itself. Stack-based GP in the Push family of languages[8], for example, features an approach

where values on stacks can be code, which can be subsequently executed; in this way, Stack-based GP can be used to achieve a kind of metaprogramming. Similarly, Linear Genetic Programming[2] (LGP) is a method which evolves a series of instructions, rather than a tree of operations, to achieve a solution.

Indeed, approaches involving the alteration of source code have garnered a growing amount of attention: in the improvement of Java programs alone, several tools for the improvement of a codebase have arisen[4,1,3]. As well as improving codebases, genetic improvement-style metaprogramming could be used to implement solutions to problems in GP, by constructing imperative processes that fit a curve, rather than a functional-style tree representation as in TGP[5].

2 Approaches with Process Fuzzing

2.1 A Brief Note on Process Fuzzing

Ultimately, Genetic Programming relies on the mutation and evaluation of a representation of a problem's solution. Process fuzzing allows for this to be achieved for imperative code, by modifying and rewriting it prior to execution. A tool implementing this is PyDySoFu[11]. PyDySoFu catches function calls and — every time a function is executed — modifies the function's AST[1] and runs the resulting code, rather than the original. This modification is performed by passing the AST through a particular function, called a "process fuzzer" (or "fuzzer").

There is a clear link between the requirement for mutation in GP and the functionality provided by a fuzzer. Some work is required, however, to represent GP-like interactions within the tool. Specifically:

1. Multiple variants must be generated and their outputs tracked, so they can be compared to each other, and ranked.
2. This ranking must be done by some function appropriate for the problem domain at hand — GP's "fitness functions".
3. Once variants are ranked, it must be possible to recombine successful ones and use these in future generations.

[1] An Abstract Syntax Tree is a tree representation of an expression in a language with a formal grammar, such as programming languages.

2.2 Improving PyDySoFu

PyDySoFu was originally unable to keep track of multiple variants, nor record the return values of the variants it produced. The solution was to extend PyDySoFu's underlying mechanism into a more fully featured aspect orientation framework, capable of more sophisticated code weaving.

This extension became an aspect orientation framework, ASP[9]. ASP's pertinent feature is its abilty to weave advice around a method of a class, such that functions can be executed before and after the method is called, without being coupled to the original codebase.

2.3 Implementing GP-Like Behaviour

Critically, aspects in ASP can be objects. When fuzzers are written as instances of aspect classes, they can use instance variables to keep track of the variants they generate between invocations. Also, because ASP is capable not only of including behaviour before method invocation, but also after, PyDySoFu can utilise this to catch the output from the method call, and use this to rank variants. This satisfies the first of the three earlier criteria.

When instances of fuzzers are created, a success metric[2] can be passed to its constructor, and this is kept within the object's state — this can then be used in the ranking of variants in a round, fulfilling the second of the above criteria.

Recombination of variants can be done by combining modified ASTs from variants in the previous round when constructing a new one — this fulfils the third of the above criteria, and is implemented in such a way as to make recombination easily tailored to individual problem domains via subclasses. Source for the `GeneticImprover` implementation of these improvements can be found in the project's repository[11].

2.4 Benefits of the Approach

Use of PyDySoFu as a GP solution has a number of advantages. First, it is under active development, meaning that the tool can

[2] Improvements to PyDySoFu were not originally developed with GP specifically in mind. Therefore, generations are referred to as "rounds", and fitness functions as "success metrics".

be expected to improve. Users can anticipate PyDySoFu to be a fertile ground for new research, where process fuzzing can be used to separate concerns in a variety of fields.

Importantly, PyDySoFu is not just a tool for implementing solutions to GP problems. Its versatility is a second benefit: its most active area of study, socio-technical variance, provides a plethora of problem domains where GP might find applications. Performing this research without a cross-disciplinary tool would require lots of ancillary work, but PyDySoFu bridges this gap.

Further, PyDySoFu is able to fuzz code *as it is run* ("dynamic fuzzing"). This functionality can be used to perform experiments with GP solutions which might — for example — use dynamic fuzzing to represent solutions which operate in an unreliable real world, such as an unreliable network or anomalies in animal populations.

3 Future Work

PyDySoFu is a new entrant into tools for running experiments within GP, with the unusual trait that its suitability for evaluating GP problems comes from its versatility, meaning that PyDySoFu is positioned to be an unusually effective tool in a variety of fields. Many things can be done to increase PyDySoFu's effectiveness as a GP tool, and to exploit it's versitility to explore new research possibilities, including:

- A wider array of GP-style fuzzers can be implemented, building on the broad array of code-improving GP approaches surfacing in the literature. These could also be used to replicate previous work in the field.
- Further exploration of GP using AST-style program mutation for codebase improvement can also be explored in Python using PyDySoFu, which, combined with the other research opportunities, makes it an exciting alternative to existing solutions.
- Given PyDySoFu naturally links socio-technical modelling and GP, experiments involving GP solutions to socio-technical problems are now feasible. A major contribution of GP interactions in PyDySoFu is that the tool's versitility allows GP to be explored within socio-technical problem domains[10].

4 Conclusion

This paper has given a brief overview of recent development of PyDySoFu, a process fuzzing tool which is now capable of GP-style interactions. While GP-style interactions arising from process fuzzing is an interesting result of its own, the availability of the tool should inspire further genetic metaprogramming work, and encourage researchers to make use of its potential across a variety of domains.

Acknowledgements

The authors would like to thank Obashi Technology for helping to fund this research, and Rob Dekkers for his help reviewing this work.

References

1. Arcuri, A., White, D.R., Clark, J., Yao, X.: Multi-objective improvement of software using co-evolution and smart seeding. In: Asia-Pacific Conference on Simulated Evolution and Learning. pp. 61–70. Springer (2008)
2. Brameier, M.F., Banzhaf, W.: Linear genetic programming. Springer Science & Business Media (2007)
3. Castle, T., Johnson, C.G.: Evolving high-level imperative program trees with strongly formed genetic programming. In: European Conference on Genetic Programming. pp. 1–12. Springer (2012)
4. Cody-Kenny, B., Galván-López, E., Barrett, S.: locogp: improving performance by genetic programming java source code. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 811–818. ACM (2015)
5. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. Statistics and computing **4**(2), 87–112 (1994)
6. Miller, J.F.: Cartesian genetic programming. In: Cartesian Genetic Programming, pp. 17–34. Springer (2011)
7. Perkis, T.: Stack-based genetic programming. In: Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on. pp. 148–153. IEEE (1994)
8. Spector, L.: Autoconstructive evolution: Push, pushgp, and pushpop. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001). vol. 137 (2001)
9. Wallis, T., Storer, T.: Asp github repository. http://www.github.com/probablytom/asp (June 2018)
10. Wallis, T., Storer, T.: Modelling realistic user behaviour in information systems simulations as fuzzing aspects. CAiSE Forum (2018)
11. Wallis, T., Storer, T.: Pydysofu github repository. http://www.github.com/twsswt/pydysofu (June 2018)