Inferring Automatic Test Oracles

William B. Langdon Shin Yoo Mark Harman
Computer Science, University College London, Gower Street, London, WC1E 6BT, UK.
School of Computing Korea Advanced Institute of Science and Technology, Daejeon 34141 Republic of Korea
shin.yoo@kaist.ac.kr w.langdon@cs.ucl.ac.uk mark.harman@ucl.ac.uk

***Abstract*—We propose the use of search based learning from existing open source test suites to automatically generate partially correct test oracles. We argue that mutation testing and n-version computing (augmented by deep learning and other soft computing techniques), will be able to predict whether a program's output is correct sufficiently accurately to be useful.**
Keywords: SBSE, Multiplicity computing, deep testing, Search Based Automatic Oracles.

I. Introduction

WE envisage a future in which most programming is largely automatic [1]. Will there still be a need to test such programs? While one might hope for a world of fully formal (and correct) specifications, we might have to wait sometime for this world to be realised. Consequently, lack of automatic testing progress will reduce productivity gains from automated programming.

Recently we [2] and others [3],[4] have surveyed the Oracle Problem in testing software. It is clear that although great strides are being made in the automatic generation of test suites, determining whether a program has passed a given test (i.e. the test oracle problem) remains largely manual.

The next section recaps the four existing approaches [3], while Section III gives our position on using search, modern optimisation techniques, data mining and Artificial Intelligence and to improve Automatic Programming via deep testing.

II. Existing Test Oracles

Nardi et al. [3, page 50] divide existing research papers into four categories: formal specification based, Metamorphic, Artificial Intelligence and N-Versions. However, we argue that none of these techniques is capable of scaling up to meet the challenge of fully automated software testing.

Formal Specification: In future, it may become easier to generate both test cases and test oracles to confirm that the program has indeed passed each test from a formal specification, than to automatically generate correct software from the specification. However, the creation of correct and complete specifications is hard and may remain so.

Metamorphic Testing: Test oracles based on metamorphic relationships [5] exploit the fact that some correct implementations maintain specific relationships between an original and a follow-up test case. For example, we know that $\sin x$ is symmetric about $x = \frac{\pi}{2}$. Suppose an implementation `my_sin(a)` returned `b`. Although it may be difficult to check whether $\sin a$ is indeed b, we can easily check whether `my_sin(a)` = `my_sin(π - a)` = `b`. Despite the elegance of the concept, often it is difficult to find metamorphic relationships in real world applications. Nardi et al. [3] report they were unable to find practical applications of metamorphic oracles.

AI based Test Oracles: AI techniques such as Artificial Neural Networks, Support Vector Machines, and Decision Trees [4] have been applied to the test oracle problem in order to learn from examples where the software is run on a test suite with known correct answers. In most cases the programs are limited to giving only one bit (binary classification: true or false) answers. However, some looked at the Triangle Program [6; 7] (which has four legal answers). On more complex problems, it is claimed that ANN could predict the correct answer more than 90% of the time [3, page 55].

N-Version Testing: At present N-Version testing is considered very expensive, since it relies on multiple (typically 2 or 3) instances of the software being independently implemented. As Bishop et al. [8] found, ensuring independence between manual implementations is very hard, even the specification may be a source of common errors. The widespread use of code search engines and reuse of open source software [9; 10] will increase the difficulty of keeping versions independent.

III. Current Weaknesses and Ways Forward

A. Artificial Intelligence Deep Test Oracles

Although there has been some work on using AI techniques as automatic test oracles only a few AI approaches have been reported and mostly on small programs. Search Based Software Engineering [11] has recently led to the use of AI to improve existing human written code [12]. There is great scope for using SBSE more widely, e.g. for fixing software bugs [13]. We propose the use of SBSE to automatically generate test oracles, i.e. Search Based Automatic Oracles. The goal of a 100% correct oracle may not be obtainable (and thus "oracle" is perhaps a misnomer). Nevertheless in future we may see the automatic generation and running of tens of thousands of test cases. And so, particularly where much of the software life cycle has been mechanised, an automated "oracle" which can deal with this volume of data and just bring to the attention of a human tester a few suspicious answers, could be a great boon.

1) Deep Learning from the Internet: Given the huge volume of open source software, can SBSE be used with deep learning optimisation techniques to infer some common sense rules for correctly behaving programs? Although unsupervised (one-class) learning can be successful, supervised learning, in which data are labelled, is often regarded as easier. Perhaps

mutation testing [7] could be used to inject (non-equivalent) faults. Thus potentially giving labelled datasets (test cases with correct outputs and test cases where broken code gave faulty answers).

2) IDE with ANN for Natural Language and Vision: Future integrated development environments may use Artificial Neural Nets to support testing. Two types of ANN have been particularly successful in natural language processing and computer vision. Recurrent Neural Nets (RNNs), such as Long Short Term Memory (LSTM) [14], have many interesting applications [15], including automated translation. Similarly, the use of Convolutional Neural Networks (CNNs) [16] has improved image recognition.

We suggest the combined use of both types of neural networks can have a strong impact on test oracles for system level testing. Current industrial practice for system and integration level testing is either still largely manual or dependent on managing a separate set of test cases written in Domain Specific Languages (DSLs). RNNs may be used to translate natural language requirements into DSLs. Using CNNs automatic test oracles could be based on treating program output screens as images rather than needing special interaction event hooks.

B. Automatic N-Version Testing

In a world of automated programming, it would seem straightforward to generate two, three or even many versions [17] of the target software. Indeed if they are created automatically [18], this would remove the possibility of human programmers assigned to different implementations of the same task talking to each other and thus leading to common failure modes in programs which are intended to be independent of each other. However, the scope for common errors via a common specification remains and explicit control (e.g. via randomisation) might be needed to ensure each version of the software is indeed different.

C. Research Questions in Automatic Test Oracle Inference

Legibility and Maintainability: Will machine generated test oracles be legible to human engineers? Will they be as maintainable as human generated ones? Existing work on automatically generated patches [19] suggests it is possible.

Partial Correctness: Machine generated test oracles are likely to be only partially correct. Instead of throwing them away, we would like to focus on their use as a warning system. Can a group of inferred test oracles be still useful, if violating a sufficient number of them activates warning for human intervention?

User Acceptability: Replacing human contribution entirely is not the goal. Instead the interaction between the automated components (test generation, execution, and oracle inference) and human developers should be carefully designed, to make the most of the human input.

Benchmarks: Research on automated test data generation had the challenge of establishing an adequate benchmark suite. We suggest the community carefully plan ahead and invest in building a reliable benchmark for the test oracle inference problem.

IV. Conclusions

Automated generation of test cases continues to make great strides yet automated oracle generation lags behind. We have argued that mutation testing, n-version computing and machine learning could combine to allow automated output checking to catch up with progress on automated input generation.

Acknowledgement: We thank our anonymous reviewers.

References

[1] M. Harman, W. B. Langdon, Y. Jia, D. R. White, A. Arcuri, and J. A. Clark, "The GISMOE challenge: Constructing the Pareto program surface using genetic programming to find better programs," in *The 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 12)*. ACM, Sept. 3-7 2012, pp. 1–14.

[2] E. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, May 2015.

[3] P. A. Nardi and E. F. Damasceno, "A survey on test oracles," *Journal on* Advances in Theoretical and Applied Informatics, vol. 1, no. 2, pp. 50–59, 2015.

[4] M. Pezze and C. Zhang, "Automated test oracles: A survey," in *Advances* in Computers. Elsevier, 2015, vol. 95, pp. 1–48.

[5] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Department of Computer Science, Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS98-01.

[6] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *Computer*, vol. 11, no. 4, pp. 34–41, April 1978.

[7] W. B. Langdon, M. Harman, and Y. Jia, "Efficient multi-objective higher order mutation testing with genetic programming," *Journal of Systems and Software*, vol. 83, no. 12, pp. 2416–2430, Dec. 2010.

[8] P. G. Bishop, D. G. Esp, M. Barnes, P. Humphreys, G. Dahll, and J. Lahti, "PODS a project on diverse software," *IEEE Transactions on Software Engineering*, vol. SE-12, no. 9, pp. 929–940, Sept 1986.

[9] N. Sahavechaphan and K. Claypool, "XSnippet: Mining for sample code," in OOPSLA '06. ACM, 2006, pp. 413–430.

[10] S. Bajracharya and C. Lopes, "Mining search topics from a code search engine usage log," in MSR '09. IEEE, 2009, pp. 111–120.

[11] M. Harman and B. F. Jones, "Search based software engineering," *Information and Software Technology*, vol. 43, no. 14, 833–839, 2001.

[12] W. B. Langdon, "Genetically improved software," in *Handbook of Genetic* Programming Applications, A. H. Gandomi, A. H. Alavi, and C. Ryan, Eds. Springer, 2015, ch. 8, pp. 181–220.

[13] W. Weimer, "Advances in automated program repair and a call to arms," in *Symposium on Search-Based Software Engineering*, ser. Lecture Notes in Computer Science, G. Ruhe and Y. Zhang, Eds., vol. 8084. Springer, Aug. 24-26 2013, pp. 1–3, invited keynote.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural* Comput., vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[15] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, ser. NIPS'14. MIT Press, 2014, pp. 3104–3112.

[16] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *International Conference on Learning Representations (ICLR2014)*. CBLS, April 2014.

[17] C. Cadar, P. Pietzuch, and A. L. Wolf, "Multiplicity computing: a vision of software engineering for next-generation computing platform applications," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, ser. FoSER '10, K. Sullivan, Ed. ACM, 7-11 Nov. 2010, pp. 81–86.

[18] W. B. Langdon and M. Harman, "Optimising existing software with genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 1, pp. 118–135, Feb. 2015.

[19] Z. P. Fry, B. Landau, and W. Weimer, "A human study of patch maintainability," in *Proceedings of the 2012 International Symposium on Software Testing and Analysis, ISSTA 2012*, Z. Su, Ed. ACM, 15-20 July 2012, pp. 177–187.