Embedding Genetic Improvement into Programming Languages

Shin Yoo
Korea Advanced Institute of Science and Technology
291 Daehak Ro, Yuseong Gu
Daejeon, Republic of Korea 34141
shin.yoo@kaist.ac.kr

ABSTRACT

We present a vision of genetic improvement firmly embedded in, and supported by, programming languages. Genetic improvement has already been envisioned as the *next compiler*, which would take human written programs as input and return versions optimised with respect to various objectives. As an intermediate stage, or perhaps to complement the fully automated vision, we imagine genetic improvement processes that are hinted at and directed by humans but understood and undertaken by programming languages and their runtimes, via interactions through the source code. We examine existing similar ideas and examine the benefits of embedding them within programming languages.

CCS CONCEPTS

•Software and its engineering → Search-based software engineering;

KEYWORDS

Genetic Improvement, Self Adaptation, Programming Language

ACM Reference format:
Shin Yoo. 2017. Embedding Genetic Improvement into Programming Languages. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082516

1 INTRODUCTION

Genetic Improvement (GI) aims to improve existing software through the use of optimisation techniques, most notably genetic programming and other evolutionary computation techniques [5]. The nature of the *improvement* ranges from repairing functional faults [3, 6, 14] to optimising non-functional behavious such as execution speed [8, 15], and energy consumption [16]. The application of GI also includes manipulation of program source code that has a fundamentally impact on the way we write software: automated program specialisation [11] and software transplantation [1, 4].

While stories of interesting GI applications abound, we also note that the approach in general presents a steep learning curve. Apart from the conversion to the somewhat unorthodox belief that making stochastic modifications to a given software may result in functional or non-functional improvements, there are many other technical components and required knowledge involved in GI, such as evolutionary computation, Pareto optimality, and sensitivity analysis. Some applications of GI would also benefit from the use of other state-of-the-art Search Based Software Engineering (SBSE) techniques such as automated test data generation [2] and fault localisation [19], making the curver even steeper.

As the number of successful applications of GI increases, it would be perhaps wise to consider what its adoption should look like during the various stages of technical maturity. We posit that embedding various ideas from GI into programming languages would provide a productive form of adoption, either as an intermediate step before the full automated future, or as a form of "human-in-the-loop" GI [5]. This paper aims to promote this argument by outlining features of a potential GI-equipped programming language and discussing the benefits of having GI techniques embedded at the programming language level.

2 IMAGINING GI AT THE PROGRAMMING LANGUAGE LEVEL

Embedding some form of adaptivity into software system is not new. Existing work include systems that can adapt memory usage [18] or behaviours [12]. The latter, in particular, advocates a novel programming paradigm (Context-oriented Programming) as a way to effectively implement adaptive systems. We propose to push down the adaptivity (i.e. elements of GI) to the next lower level, which is the programming language itself.

Which features and language constructs would a GI-equipped programming language contain? While our current proposal is a future vision rather than an ongoing research, we would like to venture describing a few possible features.

- Annotation `@optimize`: variables decorated with this annotation would be submitted to on-line optimisation, using the actual execution of the host software system as opportunities for fitness evaluation. The programmer will be able to specify what the optimisation should be with respect to.
- Keyword `optional`: modules marked with this keyword will not be loaded into the memory, saving the memory footprint of the host software system, if not used by the user for the predetermined period of time.
- Annotation `@approximate`: functions decorated with this annotation will be approximated by a faster or more energy efficient alternative, which are learnt from observations of input/output pairs during normal executions by Genetic Programming or other machine learning techniques.
- Specialisation of Runtime: the GI-equipped Java Virtual Machine or Python runtime will be able to optimise their

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
GECCO '17 Companion, Berlin, Germany
© 2017 ACM. 978-1-4503-4939-0/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3067695.3082516

internal behaviours (such as their Just In Time compilation strategies) so that they can provide the best performance for the programs they are running. This would be particularly helpful in a setting where many instances of the same program are executed (e.g. data-centres).

The first three are designed to help human developers write more adaptive software without necessarily dealing with the full learning curve of GI. Note that each of these ideas have been implemented, but as frameworks and libraries, and not at the programming language level: optimisation [13, 17], program reduction [7], and approximation [10]. The last can be also considered to be GI applied to PL, but it would still improve the programs written and executed in the language, albeit indirectly. This idea has also been implemented in the form of a framework that can optimise JIT parameters for the pypy[1] Python runtime [20].

Why embed these features into programming languages instead of providing them as additional frameworks, tools, or libraries? In fact, there exist frameworks for runtime specialisation and in-situ variable optimisation [20], as well as decision support tool that helps developers to choose the most energy efficient design [9]. However, putting these features into the language level has clear benefits:

- Precise Measurement: to improve anything, we need to be able to quantitatively measure our objective [5]. Many objectives that GI cares about (such as memory usage or energy consumption) are more easily, and possibly more accurately, measured closer to the system, i.e. within runtime environment rather that in the user code.
- Consolidated and Smoother Learning Curve: developers only need to learn GI-related features for one language, in order to apply it to any program they write using that language. They only need to learn the semantic behaviour of each feature, and not how GI works internally.
- Finer and Pervasive Control: when embedded into programming languages, the GI logic has much more pervasive access to program states compared to external libraries or frameworks. This may eliminate the need to identify *deep* parameters [13, 17], allowing GI to optimise precisely what is needed.

In summary, GI embedded in programming languages would provide more powerful control over the program being improved to the optimisation of GI while presenting less challenging learning curve to its users.

3 CONCLUSION

Many of existing GI work can be recast as programming language constructs and features. If the grand future vision for GI is the next generation compiler, then embedding the existing GI work into programming languages appears to be the first step in the right direction.

[1] A Python runtime that supports JIT compilation: https://pypy.org

REFERENCES

[1] Earl T. Barr, Mark Harman, Yue Jia, Alexandru Marginean, and Justyna Petke. 2015. Automated Software Transplantation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA 2015)*. ACM, New York, NY, USA, 257–269.

[2] Gordon Fraser and Andrea Arcuri. 2013. Whole Test Suite Generation. *IEEE Trans. Softw. Eng.* 39, 2 (Feb. 2013), 276–291.

[3] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. 2012. A Systematic Study of Automated Program Repair: Fixing 55 out of 105 bugs for $8 Each. In *Proceedings of the 34th International Conference on Software Engineering*. 3–13.

[4] Mark Harman, Yue Jia, and William B. Langdon. 2014. *Babel Pidgin: SBSE Can Grow and Graft Entirely New Functionality into a Real World System.* Springer International Publishing, Cham, 247–252.

[5] Mark Harman, William B. Langdon, Yue Jia, David R. White, Andrea Arcuri, and John A. Clark. 2012. The GISMOE Challenge: Constructing the Pareto Program Surface Using Genetic Programming to Find Better Programs (Keynote Paper). In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*. 1–14.

[6] Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim. 2013. Automatic Patch Generation Learned from Human-written Patches. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 802–811.

[7] Jason Landsborough, Stephen Harding, and Sunny Fugate. 2015. Removing the Kitchen Sink from Software. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15)*. ACM, New York, NY, USA, 833–838.

[8] W.B. Langdon and M. Harman. 2015. Optimizing Existing Software With Genetic Programming. *Transactions on Evolutionary Computation* 19, 1 (2015), 118–135.

[9] Irene Manotas, Lori Pollock, and James Clause. 2014. SEEDS: A Software Engineer's Energy-optimization Decision Support Framework. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, New York, NY, USA, 503–514.

[10] Vojtech Mrazek, Zdenek Vasicek, and Lukas Sekanina. 2015. Evolutionary Approximation of Software for Embedded Systems: Median Function. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15)*. ACM, 795–801.

[11] Justyna Petke, Mark Harman, William B. Langdon, and Westley Weimer. 2014. Using Genetic Improvement and Code Transplants to Specialise a C++ Program to a Problem Class. In *17th European Conference on Genetic Programming (LNCS)*, Miguel Nicolau, Krzysztof Krawiec, Malcolm I. Heywood, Mauro Castelli, Pablo Garcia-Sanchez, Juan J. Merelo, Victor M. Rivas Santos, and Kevin Sim (Eds.), Vol. 8599. Springer, 137–149.

[12] Guido Salvaneschi, Carlo Ghezzi, and Matteo Pradella. 2012. Context-oriented programming: A software engineering perspective. *Journal of Systems and Software* 85, 8 (2012), 1801 – 1817.

[13] Jeongju Sohn, Seongmin Lee, and Shin Yoo. 2016. Amortised Deep Parameter Optimisation of GPGPU Work Group Size for OpenCV. In *Proceedings of the 8th International Symposium on Search Based Software Engineering*, Federica Sarro and Kalyanmoy Deb (Eds.). Lecture Notes in Computer Science, Vol. 9962. Springer International Publishing, 211–217.

[14] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. 2009. Automatically Finding Patches Using Genetic Programming. In *Proceedings of the 31st IEEE International Conference on Software Engineering (ICSE '09)*. IEEE, Vancouver, Canada, 364–374.

[15] David White, Andrea Arcuri, and John Clark. 2011. Evolutionary Improvement of Programs. *IEEE Transactions on Evolutionary Computation* 15, 4 (August 2011), 515 –538.

[16] David R. White, John Clark, Jeremy Jacob, and Simon M. Poulding. 2008. Searching for Resource-efficient Programs: Low-power Pseudorandom Number Generators. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO '08)*. ACM, New York, NY, USA, 1775–1782.

[17] Fan Wu, Westley Weimer, Mark Harman, Yue Jia, and Jens Krinke. 2015. Deep Parameter Optimisation. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO 2015)*. 1375–1382.

[18] Kwaku Yeboah-Antwi and Benoit Baudry. 2015. Embedding Adaptivity in Software Systems Using the ECSELR Framework. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15)*. ACM, New York, NY, USA, 839–844.

[19] Shin Yoo. 2012. Evolving Human Competitive Spectra-Based Fault Localisation Techniques. In *Search Based Software Engineering*, Gordon Fraser and Jerffeson Teixeira de Souza (Eds.). Lecture Notes in Computer Science, Vol. 7515. Springer Berlin Heidelberg, 244–258.

[20] Shin Yoo. 2015. Amortised Optimisation of Non-functional Properties in Production Environments. In *Search-Based Software Engineering*, Márcio Barros and Yvan Labiche (Eds.). Lecture Notes in Computer Science, Vol. 9275. Springer International Publishing, 31–46.