Aiding Developer Understanding of Software Changes via Symbolic Execution-based Semantic Differencing

Johann Glock (johann.glock@aau.at)
Software Engineering Research Group, University of Klagenfurt

Motivation

While the accuracy and runtime performance of equivalence checking approaches have seen continuous improvements throughout the years, little attention has been paid to the way in which equivalence checking results are presented to developers.

```
1 | int abs(int x) {          | int abs(int x) {
2 |   if (x >= 0) {           |   if (x == 0) {
3 |     return x;             |     return x;
4 |   } else {                |   } else {
5 |     return -x;            |     return -x;
6 |   }                       |   }
7 | }                         | }
```

```
Output: NOT EQUIVALENT because the formula below is satisfiable.
((x >= 0 && abs == x) || ... ) != ((x == 0 && abs == x) || ... )
```

We hypothesize that developers' understanding of program changes and trust of corresponding tool outputs can be improved if these outputs are enriched with better visualizations and result explanations.

Research Goal

Develop an equivalence checking approach that:

- is at least as accurate as existing approaches,
- provides more information about behavioral differences,
- presents results in a way that is useful for developers,

thus aiding developer understanding of software changes.

Research Questions

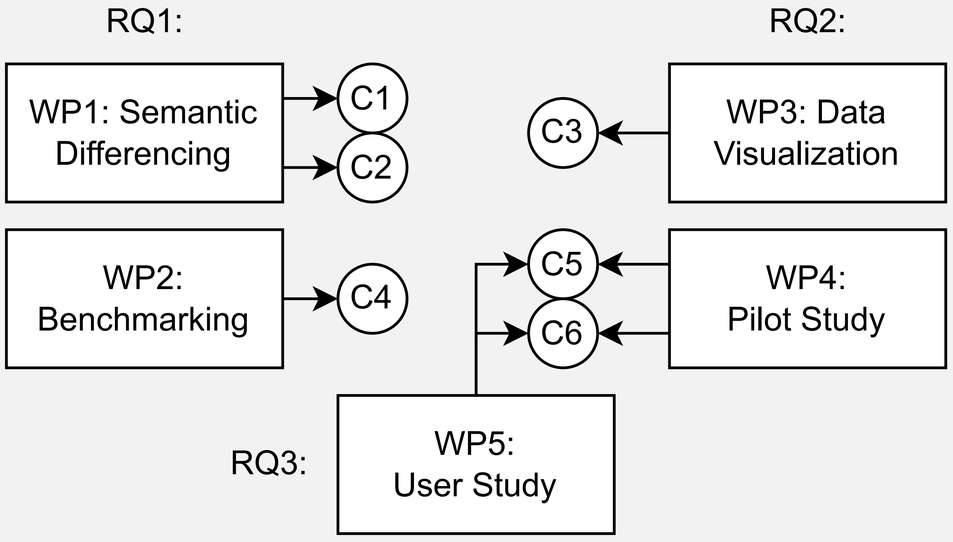
RQ1: How can we create an equivalence checking approach that is at least as accurate as existing approaches but provides information about behavioral / semantic differences?

RQ2: How should we present the collected information (i.e., equivalence checking results, behavioral differences) to developers in order to be most useful for them?

RQ3: To which degree does the provided information improve (or hinder) the speed, reliability, etc. with which developers are able to reason about software changes?

Research Approach

We split our research work into five work packages (WP1–WP5) that produce a total of six research contributions (C1-C6):



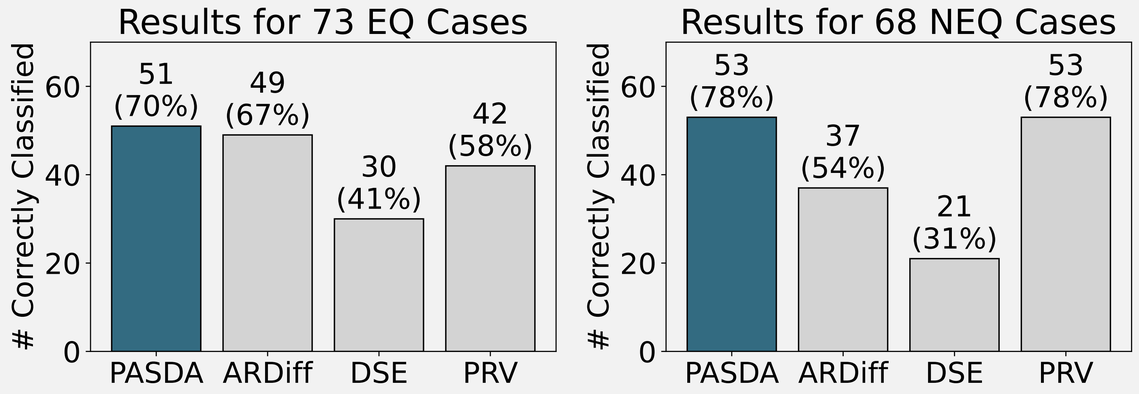
WP1 concerns the development of the semantic differencing approach and its prototypical implementation. WP2 evaluates the prototype's accuracy and runtime performance. In WP3, an IDE plugin is created that visualizes the semantic differencing data via source code editor augmentations. WP4 and W5 evaluate the usefulness of the IDE plugin via experiments with developers.

Expected Contributions

- **C1** An **approach** for equivalence checking of software programs that provides more information about behavioral / semantic differences than existing approaches.
- **C2** An open source **prototype** that implements the computation of the raw semantic differencing data as a Java application.
- **C3** An open source **prototype** that implements the processing and visualization of the semantic differencing data as an IDE plugin.
- **C4** **Benchmarking results** that compare the equivalence checking accuracy and runtime requirements of our approach to state-of-the-art equivalence checking approaches.
- **C5** **Experimental results** that compare how quickly and accurately developers are able to complete change understanding tasks when using our IDE plugin prototype vs. state-of-the-art tools.
- **C6** **Developer feedback** that compares the perceived usefulness of our IDE plugin prototype to state-of-the-art tools.

Preliminary Results

For WP1 and WP2, we developed an equivalence checking / semantic differencing tool called PASDA [1] and compared it to the three existing tools ARDiff [2], DSE [3], and PRV [4] using the ARDiff benchmark. PASDA **correctly classifies** 104 of 141 (**74%**) cases, i.e., 9–51 more cases than the three existing tools.



In addition, for each input partition that PASDA identifies, it aims to provide (i) a **partition-level non-/equivalence proof**, (ii) concrete and symbolic input and output values, and (iii) the lines of code that are covered by the corresponding execution path. For the two versions of the `abs` function shown before, PASDA reports:

Input Partition	Output v1	Output v2	Example v1	Example v2	Covered Lines v1	Covered Lines v2	Equiv. Class.
x = 0	x	x	1 → 1	1 → 1	1, 2, 3	1, 2, 3	EQ
x > 0	x	-x	1 → 1	1 → -1	1, 2, 3	1, 2, 5	NEQ
x < 0	-x	-x	-1 → 1	-1 → 1	1, 2, 5	1, 2, 5	EQ

For programs and partitions for which non-/equivalence cannot be proven, PASDA reports a **best-effort equivalence classification** of MAYBE_EQ / MAYBE_NEQ if at least a partial non-/equivalence proof can be provided. Existing tools simply classify such cases as UNKNOWN and provide no further information for them.

References

[1] Johann Glock, Josef Pichler, and Martin Pinzger. PASDA: A partition-based semantic differencing approach with best effort classification of undecided cases. *Journal of Systems and Software*, 2024.

[2] Sahar Badihi, Faridah Akinotcho, Yi Li, and Julia Rubin. ARDiff: Scaling program equivalence checking via iterative abstraction and refinement of common code. In *Proceedings of the 28th Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 13–24. ACM, 2020.

[3] Suzette Person, Matthew B. Dwyer, Sebastian G. Elbaum, and Corina S. Păsăreanu. Differential symbolic execution. In *Proceedings of the 16th International Symposium on Foundations of Software Engineering*, pages 226–237. ACM, 2008.

[4] Marcel Böhme, Bruno C. d. S. Oliveira, and Abhik Roychoudhury. Partition-based regression verification. In *Proceedings of the 35th International Conference on Software Engineering*, pages 302–311. IEEE, 2013.