Project proposal
15-745 Spring 2019

1 Project info

Title	Improving the precision of incremental algorithms for pointer analysis
Team members	Nick Roberts (`nroberts@andrew.cmu.edu`)
	Vijay Ramamurthy (`vijayram@andrew.cmu.edu`)
Website	`https://ncik-roberts.github.io/15745`

2 Project motivation

Pointer analysis is a well-established technique for estimating aliasing among variables and heap objects in a program. Its results can be used in race detection, among other compilation and debugging scenarios. Incremental algorithms (or "differential algorithms") for pointer analysis allow for greater efficiency in the presence of change: As the programmer modifies source code for which pointer analysis facts have already been derived, an incremental algorithm will update the derived facts accordingly, with the same result as if a non-incremental algorithm had been run on the entirety of the modified program. The benefit of an incremental algorithm is that updating a database of facts can be orders of magnitude more efficient than rederiving the database from scratch.

Recent work by Liu and Huang on their D4 framework [1] puts this into practice, combining the insight about incremental algorithms with two other major insights: (1) that incremental algorithms can be designed for parallelism, and (2) that decoupling the client (the IDE where the code is developed) from the server (the location where the pointer analysis data structures are maintained) makes it feasible for the server to be a much higher-performance machine than would be reasonable for a client. Applying these insights together, they see significant speedup in race detection: 0.12s, on average, compared to 25s using previous approaches. But their improved incremental algorithm alone provides speedup compared to previous incremental approaches: on average, 0.72ms (versus 6.8ms) for insertions and 73ms (versus 26s) for deletions.

In our project, we intend to take advantage of the performance gains found by Liu and Huang by recovering precision. They use Andersen's flow-insensitive, context-insensitive algorithm; we will develop a context-sensitive algorithm (using a variety of context heuristics), and, as time permits, flow-sensitive and other more precise algorithms. Since Liu and Huang's analysis improved on the state-of-the-art in an incremental setting, our analysis is guaranteed to be novel as long as we are sensitive to the same constraints they were: the algorithm should be incremental and amenable to massive parallelism; the data structures should be of a feasible size to construct on real machines. We will implement our algorithms in the D4 framework and compare the performance and precision to existing approaches.

3 Project approach

To operate under the framework used in D4, a pointer analysis must efficiently support two operations: insert and delete. Adapting these operations to more precise pointer analysis will require some amount of novel algorithmic work on our part. Once we have developed algorithms, we will extend the D4 framework, which operates on Java bytecode, to use our analysis. To

adapt existing analyses, we will have to likewise adapt the typical rules used for constructing a pointer analysis from Java bytecode [2].

To evaluate the performance of our implemented algorithm, we will run on the same subset of the DaCapo Benchmarks[1] used in [1]; as necessary, we will add additional benchmarks from this suite. We will assess both the performance of our implementation and the relative precision of our analyses.

Though more general frameworks for context sensitivity exist, in the process of working on our project we will begin by implementing a dominant flavor of context sensitivity, likely object sensitivity [3].

4 Goals

75% goal	Implement object-sensitive (or call-site sensitive) pointer analysis in D4 framework; analyze and evaluate
100% goal	Implement more general context-sensitive pointer analysis in D4 framework; analyze and evaluate
125% goal	100% goal, but additionally consider other ways to enhance pointer analysis precision (like flow sensitivity and shape analysis); implement these where possible in D4 framework, and discuss where not possible.

5 Schedule

Week 1	3/25–4/01	Read papers and successfully existing D4 framework.
Week 2	4/02–4/08	Develop algorithms by hand (at least context-sensitive).
Week 3	4/09–4/15	Implement algorithms in D4.
Week 4	4/16–4/22	Finish implementation of algorithms in D4.
Week 5	4/23–4/29	Run experiments and evaluations of performance.
Week 6	4/30–5/08	Write report and make poster.

6 Resources needed

The D4 framework is freely available on GitHub.[2] We will use AWS if we need to test on a system with more than the 2 cores that our personal laptops have. This is all the software and hardware we believe we will need.

7 Outline of final paper

We expect that the outline linked from the course website will be appropriate for our project.[3] The "details regarding our approach" will relate to the design and implementation of our incremental pointer analysis algorithms; and we will most likely run our code on the previously-described DaCapo Benchmarks.

[1] `http://dacapobench.org/`

[2] `https://github.com/parasol-aser/D4`

[3] `http://www.cs.cmu.edu/~15745/final_report.pdf`

8 Getting started

We have read the D4 paper and are exploring related literature. Nick has some ideas for the context-sensitive algorithms, and in the meantime we can both get started looking at the D4 codebase.

References

[1] B. Liu and J. Huang, "D4: Fast concurrency debugging with parallel differential analysis," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2018. New York, NY, USA: ACM, 2018, pp. 359–373. [Online]. Available: http://doi.acm.org.proxy.library.cmu.edu/10.1145/3192366.3192390

[2] S. Zhan and J. Huang, "Echo: Instantaneous in situ race detection in the ide," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 775–786. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950332

[3] A. Milanova, A. Rountev, and B. G. Ryder, "Parameterized object sensitivity for points-to analysis for java," *ACM Trans. Softw. Eng. Methodol.*, vol. 14, no. 1, pp. 1–41, Jan. 2005. [Online]. Available: http://doi.acm.org/10.1145/1044834.1044835