Checkpoint report
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 Summary of checkpoint report

Our motivation remains the same: we want to improve the precision of efficient incremental pointer analysis algorithms. Our goals have been reduced, in part owed to the rather raw code of the framework we are basing our implementations on.

3 Motivation

The motivation remains the same as in our project proposal.[1] To reiterate: in light of the performance gains of the D4 framework [1] over existing incremental pointer analysis algorithms, we see the opportunity to recoup precision. In particular, we are performing a survey of which pointer analysis algorithms (especially those more precise than the traditional Andersen's algorithm currently implemented in D4) can be adapted to this incremental setting, with us furthermore providing implementations of some representative algorithms.

Based on our research, we have narrowed our focus of possible precision improvements. We are primarily focusing on call-site sensitive analysis [2] rather than object sensitive analysis.

4 Goals

The principal change from the proposal is in the extent of our goals: we believe that the previous 75% goal should be promoted to replace the previous 100% goal. The reason for this relaxation of goals is because of the somewhat tangled and roughly-engineered state of the D4 codebase. We sunk some more time in setup than anticipated because we had to comb through Git commits to find the right version of several repositories, and all the typical “features” of a large Java codebase are manifest (multiple layers of indirection to the point that it's difficult to tell what code is being run where; questionable adherence to inheritance instead of composition; runtime inspection of types). We identify this not to criticize D4 but to explain our setup difficulties: the codebase was engineered to work for a small group of people whose offices were in walking distance of each other, not for outside extension.

[1] `https://ncik-roberts.github.io/15745/files/proposal.pdf`

Here are what we believe to be reasonable updated goals:

75% goal	Partially implement call-site–sensitive pointer analysis in D4 framework; analyze and evaluate
100% goal	Efficiently implement call-site–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.

A partial implementation might be an implementation that doesn't achieve the time- and space-efficiency of the original D4 paper. An efficient implementation is one that adheres to the same efficiency goals of the D4 paper. For example, the constructed pointer analysis graph should not be "too large" (i.e. it should fit in memory for most Java programs), and the incremental analysis should run faster than existing incremental algorithms.

We removed the previous 100% goal, which was to implement a general framework for context-sensitive pointer analysis in an incremental setting (such as an analysis that considers call sites up to depth n rather than depth 1). Based on our research, this seems to be significantly more complicated than an analysis with depth 1 in an incremental setting, and it would likely be more interesting to discuss algorithms for other types of pointer analysis rather than focusing solely on context sensitivity.

5 Evaluating our progress so far

Looking at our original proposal, we have met our milestones so far (successfully import and run D4 framework; develop algorithm for call-site–sensitive analysis by hand), and we are working to meet the third milestone (implement algorithm in D4). We anticipate that this third milestone will be more difficult than the previous one, but we left two weeks for it in the proposal, so we remain on track.

For the remaining weeks, we expect to adhere to the timeline outlined in the proposal:

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 Roadblocks

A potential roadblock continues to be specters of bugs in the existing D4 implementation. Some incremental updates to the analyzed program will cause the D4 implementation to raise errors, and it is unclear if this is because the framework does not support these changes (e.g. deletion of entire methods) or if we are using an incorrect version of the code. Such bugs might complicate our implementation of our context-sensitive algorithm, which could delay the delivery of our fourth milestone.

There are some roadblocks regarding the development of algorithms, but this affects only our 125% goal. It was not too hard to develop a call-site–sensitive algorithm, but we remain unconvinced that flow- or path-sensitive pointer analysis algorithms are feasible. We are not concerned about this roadblock because we have developed an algorithm for our 100% goal.

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] R. P. Wilson and M. S. Lam, “Efficient context-sensitive pointer analysis for c programs,” in *Proceedings of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation*, ser. PLDI ’95. New York, NY, USA: ACM, 1995, pp. 1–12. [Online]. Available: http://doi.acm.org/10.1145/207110.207111