Data Reachability Analysis for Partial Compile-Time Garbage Collection in Java

Samir Jindel Logan Brooks

{sjindel, lcbrooks}@andrew.cmu.edu

Web page: http://www.andrew.cmu.edu/user/lcbrooks/15745project/

Research Question

Reference-counting garbage collection schemes typically need to be complemented with systems that collect cyclical data structures at runtime, such as mark-and-sweep algorithms. Unfortunately, there are performance drawbacks to the mark-and-sweep that limit utility in real-time applications, and incur memory and runtime overhead over manual memory management. In general, reference counting is insufficient to free cyclical data structures, and a runtime cycle detector is necessary to some extent. We intend to mitigate the overhead of garbage-collection by minimizing the scope of the mark-and-sweep collector by using a data reachability analysis at compile-time to identify which allocation sites within the program may or may not spawn objects involved in cyclic data structures.

75% goal	Data reachability analysis — identify a notable portion of all structures used in a collection of test programs as acyclic
100% goal	Also have a runtime system that implements the changes to memory management described
125% goal	Fine-tune garbage collection or analysis with the goal of improving on the results stated in [2]

Logistics

Plan of Attack and Schedule: Table 1 shows our tentative schedule for work on major tasks.

Week		Samir	Logan
3/21–3/27	Formalize the analysis		
3/28–4/3	Implement basic analysis	Intraprocedural	Interprocedural
4/4–4/10	Extend analysis to handle full Java	Other hairy features	Virtual function calls
4/11–4/17	Begin to implement runtime system	Modified garbage collector	Modified code generation
4/18–4/24	Finish implementing runtime system		
4/25–4/30	Write up paper		

Table 1: Proposed project schedule

Milestone: Achieve most of 75% goal, the static analysis. It should be able to handle basic Java programs, and compute useful results for them.

Literature Search: We have examined a variety of papers that relate to compile-time garbage collection:

- Compile-time garbage collection for Java programs [2], which tries to replace heap memory management of certain objects using a garbage collector with `malloc`'s and `free`'s

- Shape analysis which tries to identify whether certain data structures as a whole could contain cycles [3]
- Related papers [1, 9, 8, 6, 11, 7, 5, 4, 10]
- If we are missing anything, we are unaware.

Resources Needed: We intend to use an existing Java optimization framework (such as Soot) as well as open-source runtime (such as that in Jikes), but have not yet decided on which software to use. Such software should be openly available. We already have access to the hardware and machines that should be needed (personal/cluster computers, etc.). Except for the optimization framework and runtime as noted above, we have all the resources needed to conduct this study.

Getting Started: We have an outline of the analysis algorithm and of the garbage collector. We do not have any questions or constraints that would prevent us from continuing work immediately.

References

[1] Chandrasekhar Boyapati, Alexandru Salcianu, William Beebee Jr, and Martin Rinard. Ownership types for safe region-based memory management in real-time java. *ACM SIGPLAN Notices*, 38(5):324–337, 2003.

[2] Sigmund Cherem and Radu Rugina. Compile-time deallocation of individual objects. In *Proceedings of the 5th international symposium on Memory management*, pages 138–149. ACM, 2006.

[3] Brian Hackett and Radu Rugina. Region-based shape analysis with tracked locations. *SIGPLAN Not.*, 40(1):310–323, January 2005.

[4] Geoff W Hamilton. Compile-time garbage collection for lazy functional languages. In *Memory Management*, pages 119–144. Springer, 1995.

[5] Lucy Hederman. *Compile-time garbage collection using reference count analysis.* PhD thesis, Masters thesis, Rice University, Aug. 1988. Also Rice University Technical Report TR88–75 but, according to Rice Universitys technical report list, this report is no longer available for distribution, 1989.

[6] Wei Huang, Werner Dietl, Ana Milanova, and Michael D Ernst. Inference and checking of object ownership. In *ECOOP 2012–Object-Oriented Programming*, pages 181–206. Springer, 2012.

[7] Thomas Kotzmann and Hanspeter Mössenböck. Escape analysis in the context of dynamic compilation and deoptimization. In *Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 111–120. ACM, 2005.

[8] Ana Milanova and Yin Liu. Practical static ownership inference. Technical report, Technical Report RPI/DCS-09-04, Rensselaer Polytechnic Institute, 2009.

[9] Ana Milanova and Yin Liu. Static ownership inference for reasoning against concurrency errors. In *ICSE Companion*, pages 279–282, 2009.

[10] Cristina Ruggieri and Thomas P Murtagh. Lifetime analysis of dynamically allocated objects. In *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 285–293. ACM, 1988.

[11] Alisdair Wren, Paul Kelly, and Nobuko Yoshida. Inferring ownership. *Master's thesis, Imperial College London, UK (June 2003)*, 2003.