

# Static Garbage Collection for L3 (15-745)

Neelakantan Krishnaswami (neelk+@cs.cmu.edu)  
William Lovas (wlovas+@cs.cmu.edu)

2 April 2006

Safe languages like L3 use garbage collection to automatically manage the allocation and deallocation of program objects. Garbage collection has the advantage that it is provably safe, eliminating a large class of program errors.

However, the reachability heuristic collectors use is always an approximation to the actual lifetimes of program objects, and so garbage-collected programs can use more memory than they really need. Since modern architectures have a very complex memory hierarchy<sup>1</sup> with severe performance penalties for moving down the hierarchy, this means that excessive memory usage can potentially have a drastic impact on the performance of a program.

We propose to mitigate this performance hazard by doing analyses to identify object lifetimes and using that information in order to drive optimizations to statically allocate and deallocate some program data. We hope to potentially improve the performance of a program by reducing its total memory usage. This will have the effect of reducing the number and frequency of garbage collections the runtime system must perform, and will also tend to move program data up the memory hierarchy.

We plan on taking advantage of the strongly-typed, first-order nature of the L3 language to statically bound the lifetimes of dynamically allocated objects.

Because L3 is a safe language, we can examine the structure type declarations in a program, and figure out which pointers can contain other pointers.[3] Furthermore, since L3 is a purely first-order language, the call graph of the entire program is known at compile time. Together, these features will let us examine a procedure body and conservatively determine whether or not it is possible to store a newly-allocated value in a data structure that will outlive the procedure call or not.

In this way, we hope to avoid having to implement the very complex analyses[1, 2] that are necessary in languages like C, due to unsafe features like type casts. We will need to change the runtime system to support manual memory allocation and deallocation, and to propagate the necessary information from the L3 source into the IR in order to estimate object lifetimes.

---

<sup>1</sup>Three levels of cache, main memory, plus disk-based virtual memory is not an atypical memory hierarchy.

We plan on evaluating this modification by comparing both the runtime and the maximum allocated memory of the L3 test suite before and after our changes. Measuring memory usage may involve instrumenting the allocators, so we will measure the runtime on the uninstrumented version.

## References

- [1] Vikram Adve, Dinakar Dhurjati, Sumant Kowshik and Chris Lattner. Memory Safety Without Runtime Checks or Garbage Collection. In *Proceedings of Languages Compilers and Tools for Embedded Systems 2003*, San Diego, CA, June 2003.
- [2] Dan Grossman, Greg Morrisett, Trevor Jim, Michael Hicks, Yanling Wang, and James Cheney. Region-based memory management in Cyclone. In *Proceedings of the ACM Conference on Programming Language Design and Implementation*, pages 282–293, June 2002.
- [3] Frederick Smith, David Walker, and Greg Morrisett. Alias types. *Lecture Notes in Computer Science*, 1782:366–381, 2000.