Origins of Garbage Collection

Jason Goodman

1 Recursive functions of symbolic expressions and their computation by machine, Part I

```
@article{mccarthy-60-recursive,
    author = "John McCarthy",
    title = "Recursive functions of symbolic expressions and their
        computation by machine, Part {I}",
    journal = cacm,
    volume = 3,
    number = 4,
    year = 1960,
    pages = "184--195",
}
```

Summary: McCarthy introduces the LISP language and invents garbage collection to implement the language, devoting just over a page to a mark and sweep algorithm that identifies reachable words by depth-first search. McCarthy insists that S-expression structures be immutable, allowing for copying by reference sharing. This optimization, as well as a future relaxation on immutability, gives a practical motivation for garbage collection.

Evaluation: This paper breaks ground on several areas of research, including garbage collection. Presumably after noticing the undecidability of identifying all values a program will need in the future, McCarthy arrives at the modern contract between programmers and garbage collectors: Only structures that aren't reachable from program variables can be reclaimed. The given algorithm is useful as a proof of concept but has several drawbacks: Programs can pause at unpredictable times; memory becomes fragmented as individual words are reclaimed and allocated, causing poor cache performance; expensive garbage collection calls become frequent as programs operate near memory capacity; garbage collection takes time on the order of the size of memory; and depth-first search requires a stack potentially on the order of the amount of memory used.

2 A method for overlapping and erasure of lists

```
@article{collins-60-method,
    author = "George E. Collins",
    title = "A method for overlapping and erasure of lists",
    journal = cacm,
    volume = 3,
    number = 12,
    year = 1960,
    pages = "655--657",
}
```

Summary: Collins improves upon McCarthy's LISP garbage collector by generalizing the trivial approach of allowing exactly one reference to each structure and making copies to for other viewers. Collins realizes the same approach can be used with "overlapping" lists—structures with multiple references—by counting the number of references to a word at runtime. Every assignment updates the reference counts associated with the old and new words being referenced, and removing a structure's only reference causes its memory to be reclaimed. As an optimization, Collins only allocates space for a reference count when a word has more than two references to it; when this happens, the word is copied to a new address and the original is replaced with a reference count and pointer to the value's new location.

Evaluation: This paper originates the idea of reference counting garbage collection, which is still used today. Reference counting is particularly useful for user-facing applications that can't afford to pause for several seconds at a time as in McCarthy's approach. That said, this approach adds instructions for every assignment, as opposed to other algorithms that take time proportional to some feature of the world when memory runs out. Another stumbling block is that structures with reference cycles can't be collected by reference counting. McCarthy's LISP does not originally allow cyclic data structures, but modern applications of reference counting either break McCarthy's abstraction of "abandoned" data or accept these leaks as unavoidable.

3 A LISP Garbage Collector Algorithm Using Serial Secondary Storage

```
@techreport{minksy-63-lisp,
    author = "Marvin L. Minsky",
    title = "A {LISP} Garbage Collector Algorithm Using Serial
        Secondary Storage",
    institution = "MIT",
    year = 1963,
    number = "AIM-58"
}
```

Summary: In this AI Memo, Minsky introduces a garbage collector that copies reachable words to secondary storage in a depth-first search, assigning them fresh contiguous addresses, and then loads the words back into memory. The depth-first ordering causes list structures to occupy consecutive addresses, reducing cache misses. Moreover, Minsky briefly suggests collapsing aligned linked list structures to arrays. The key observation is that data can be repositioned in memory so long as its pointer structure is preserved.

Evaluation: The paper lacks polish, introducing the algorithm with a page of ramp-up and a page of code with single-variable names, GOTO control, and conceptually unnecessary special casing. That said, the idea is an improvement over McCarthy's algorithm and kicks off significant research. Minsky's approach not only avoids fragmentation but runs in time proportional to the amount of memory used rather than the size of main memory. The paper doesn't discuss this, but the possibility of faster, more frequent garbage collection eases some downsides to McCarthy's approach and avoids a long creep of fragmentation before memory runs out.

4 A LISP garbage-collector for virtual-memory computer systems

```
@article{fenichel-69-lisp,
    author = "Robert R and Jerome C. Yochelson",
    title = "A {LISP} garbage-collector for virtual-memory computer systems",
    journal = cacm,
    volume = 12,
    number = 11,
    year = 1969,
    pages = "611--612",
    doi = "\url{http://doi.acm.org/10.1145/363269.363280}",
}
```

Summary: This two-page paper presents a copying garbage collector in the context of virtual memory. The algorithm is effectively the same as Minsky's, but instead of explicitly copying to disk uses two "semispaces" of the virtual address space, copying from one to the other during each collection. The first key observation is that memory will never appear to run out in a large enough address space. Rather, programs will begin to run slowly. The second observation is that fragmentation over pages amplifies this effect, with page misses analogous to processor cache misses. The authors suggest some possible triggers for garbage collection, such as slow memory accesses and I/O blocking, but don't recommend a particular policy.

Evaluation: This paper is significant in identifying the changing priorities of garbage collection with the problem of "running out" of memory less of a concern. At this point, McCarthy's approach of visiting each address would be infeasible, and garbage collection isn't strictly necessary for most programs. Instead, the new role of garbage collection is to increase program efficiency by compacting structures into contiguous regions that cross as few page boundaries as possible. Also notable is a clear presentation of the depth-first copy collection algorithm, identical in spirit to Minsky's, in under twenty lines of readable code.

5 A nonrecursive list compacting algorithm

```
@article{cheney-70-nonrecursive,
    author = "C. J. Cheney",
    title = "A nonrecursive list compacting algorithm",
    journal = cacm,
    volume = 13,
    number = 11,
    year = 1970,
    pages = "677--678",
}
```

Summary: In this two-page paper, Cheney presents a copying garbage collector that uses a breath-first traversal of reachable memory rather than the depth-first order used in prior algorithms. This approach avoids the need for a stack on the order of the size of memory used, but at the expense of not coalescing list structures. Like Fenichel and Yochelson's, this algorithm copies between regions of a virtual address space, but Cheney takes advantage of the ability to read from the "to" space while copying—something not possible with Minsky's approach of writing address-value pairs to disk out of order.

Evaluation: Like reference counting, this is the first big improvement on the amount of space required by garbage collection. The algorithm is also straightforward and well-presented, making it a practical alternative to depth-first ordering in a world where page misses are costlier than processor cache misses.