Optimizing lazy functional languages with precise reference counting

Martin Fredin

June 2023

1 Introduction

All values in purely functional languages are immutable. Immutability is important because it limits shared state, and makes it easier to reason about the program. In practice, this means that a function such as map returns a new list instead of modifying the input.

$$
\begin{aligned}
&\mathsf{map} : (A \to B) \to \mathsf{List}\ A \to \mathsf{List}\ B \\
&\mathsf{map}\ f\ [] = [] \\
&\mathsf{map}\ f\ (x :: xs) = f\ x :: \mathsf{map}\ f\ xs
\end{aligned}
$$

This is great if the input list (List A) is used later in the program. Otherwise, it is better to reuse the allocation for List A by updating the list in place. This avoids both the deallocation of List A and the allocation of List B. To preserve the semantics of the program, destructive updates should only be perfomed on values which are garanteed to be non-shared. We call a value shared if there is multiple references to it.

Reference counting (Collins, 1960) is a memory management technique which can detect and free resources as soon as they are no longer needed, allowing memory reuse and destructive updates. Common reference counting algorithms are easy to implement; each allocation contains an extra field which tracks the number of references to a value, and reclaims the heap space once the reference count drops to zero. As a result of the interleaved collection strategy, memory usage remain low and the throughput is continuous throughout the computation[1] (Jones & Lins, 1996). However, ...

[1] Not entirely true.

- Lower throughput than tracing GC
- Not very good at handling short lived values which is most values (especially in pure FP)
- Reinking et al. (2021) and Ullrich and de Moura (2021) minimize RC operations through precise reference counting and a linear calculus.
- more...

2 Graph Reduction Intermediate Notation

3 Extending GRIN with reference counting

4 Result

5 Relevant Work

6 Conclusion and Future Work

- Add reuse and borrowing
- Utilse GRIN's whole program compilation strategy and the heap points-to analysis to statically determine unshared values during compile time, and thus minimizing the number of reference countinging operations.
- It would also be intresting to utilse 0-modality (erasure) in Agda's type system, and later also 1-modality when Agda gets it.

References

Collins, G. E. (1960). A method for overlapping and erasure of lists. *Commun. ACM*, *3*(12), 655–657. https://doi.org/10.1145/367487.367501

Jones, R., & Lins, R. (1996). *Garbage collection: Algorithms for automatic dynamic memory management*. John Wiley & Sons, Inc.

Reinking, A., Xie, N., de Moura, L., & Leijen, D. (2021). Perceus: Garbage free reference counting with reuse. *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 96–111. https://doi.org/10.1145/3453483.3454032

Ullrich, S., & de Moura, L. (2021). Counting immutable beans: Reference counting optimized for purely functional programming. *Proceedings of the 31st Symposium on Implementation and Application of Functional Languages*. https://doi.org/10.1145/3412932.3412935