Programming with destinations in Haskell

Thomas Bagrel[1]

[1]INRIA/LORIA, Vandœuvre-lès-Nancy, 54500, France
[2]TWEAG, Paris, 75012, France

Destination-passing style programming introduces destinations, which represents the address of a write-once memory cell. Those destinations can be passed around as function parameters, and thus enable the caller of a function to keep control over memory allocation: the body of the called function will just be responsible of filling that memory cell. This is especially useful in functional programming languages such as Haskell, in which the body of a function is typically responsible for allocation of the result value.

Programming with destination in Haskell is an interesting way to improve performance of critical parts of some programs, without sacrificing memory warranties. Indeed, thanks to a linearly-typed API we designed, a write-once memory cell cannot be left uninitialized before being read, and is still disposed of by the garbage collector when it is not in use anymore, eliminating the risk of uninitialized read, memory leak, or double-free errors that can arise when memory is handled manually.

With the implementation of destinations for Haskell through compact regions we provide in this article, we reach a 15-40% improvement over memory allocation in a simple parser example, and 0-50% improvement in run time. We also provide a few examples of programs that can be implemented in a tail-recursive fashion thanks to destinations, which is crucial for performance in strict contexts.

Safety proofs for the API are not provided in this article though, and will be the subject of a future article.

1 Introduction

Blabla intro.

References