GRIN: Dead data elimination in the context of dependently typed languages

Péter Dávid Podlovics[*], Csaba Hruska

February 4, 2019

Abstract

GRIN is short for Graph Reduction Intermediate Notation [1], a modern back end for lazy functional languages. Most of the currently available compilers for such languages share a common flaw: they can only optimize programs on a per-module basis. The GRIN framework allows for interprocedural whole program analysis, enabling optimizing code transformations across functions and modules as well.

One of the optimizations supported by GRIN is dead data elimination, a special case of dead code elimination [2]. Conventional dead code eliminating optimizations usually only remove statements or expressions from programs; however, dead data elimination can transform the underlying data structures themselves. Essentially, it can specialize a certain data structure for a given use-site by removing or transforming unnecessary parts of it. It is a very powerful optimization technique that can significantly decrease memory usage and reduce the number of required heap operations.

In the context of dependently typed languages, dead data elimination (with the help of other front end side transformations) is capable of erasing unnecessary type indices from dependent data constructors. In this talk, I will outline a general overview of GRIN, elaborate on dead data elimination, and also present some initial results of their applications to dependently typed languages.

References

[1] U. Boquist, "Code Optimisation Techniques for Lazy Functional Languages," Ph.D. dissertation, Chalmers University of Technology and Göteborg University, 1999.

[2] R. Turk, "A modern back-end for a dependently typed language," Master's thesis, Universiteit van Amsterdam, 2010.

[*]The author has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).