Linting Linearity in Core/System FC

Rodrigo Mesquita

November 8, 2022

1 Introduction

Since Linear Haskell's**??** publication and implementation release in GHC 9.2, Haskell's type system supports linearity annotations in functions – bringing linear types into a mainstream pure and lazy functional language.

System FC is the formal system in which the implementation of GHC's intermediate representation language *Core* is based on.

There are at least two distinct typecheckers in core. The first is run on the frontend language, i.e. the Haskell we write, and is a big and complex typechecker. The second is run on the intermediate language *Core* that we obtain from desugaring Haskell.

Core is a much smaller and more principled language than the whole of Haskell (even though we can compile the whole of Haskell to it), and the typechecker for it is small and fast due to *Core* being explicitly typed and having a very small abstract syntax tree. This typechecker is called *Lint* and gives us guarantees of correctness (i.e. sanity) in face of the complexity of all the transformations a Haskell program undergoes, such as type inference, desugaring and optimising transformations (and other Core related passes?).

The addition of linear types comes at two levels: frontend and Core. What's the first and foremost reason for permeating Core with linearity? I can't phrase it correctly, something wrt to the foundational formal design and system FC. In the frontend, we can now annotate with linearity type declarations and typecheck programs that use them accordingly (a linear fuction will consume its argument exactly once if it is consumed exactly once, for some definition of *consume* that I should revise here). In Core, we still have linearity annotations and ideally *Lint* would check that all the GHC transformations to our linear program preserved its linearity. However, **it can't**!

Despite the strong formal foundations of linear types driving the implementation, their interaction with the whole of GHC is still far from trivial. Diverse problems with linearity spring up when we get past the desugarer. In particular, optimizing transformations, coercions from GADTs and type families, recursive lets, and empty case expressions don't currently fit in with linearity.

We believe that GHC's transformations are correct, and it is the linear type system that can't accommodate the resulting programs. We aim to formalise a

type system that is able to type check *Core/System FC* at all points in the core pipeline and implement it into GHC to be able to preserve linearity accross the stages and to enable *Lint* to preserve our sanity regarding linearity.

2 Motivation

- Are we really preserving linearity?
- Inform/unlock other optimisations that take into account linearity

3 Typing Usage Environments

The first set of problems appears in the core-to-core optimisation passes. GHC applies many optimising transformations to *Core* and we believe those transformations preserve linearity. However, our linear type system cannot check that they indeed preserve linearity.

The simpler examples come from straightforward and common optimising transformations. Then we have recursive let definitions that don't accommodate linearity even though it might converge to only use the value once. Finally, we have the empty case expression introduced with the *EmptyCase* language extension that we currently can't typecheck either.

- Transformations, occurrences?
- Recursive lets
- Empty case expression

4 Multiplicity Coercions

The second set of problems arises from our inability to coerce a multiplicity into another (or say that one is submultiple of another?).

When we pattern match on a GADT we ...

Taking this example (copy example over) we can see that we don't know how to say that x is indeed linear in one case and unrestricted in the other, even though it is according to its type. We'd need some sort of coercion to coerce through the multiplicity to the new one we uncover when we pattern match on the GADT evidence (...)

- Are we really preserving linearity?
- Inform/unlock other optimisations that take into account linearity