Haskell Foundation Technical Task Force (Slot 1)

March 17, 2021

Attendees

- Ben Gamari
- Davean
- Michael Snoyman

Potential for split-base

- Discussed the possibilities of decoupling base and GHC
- Ben listed some of the interdependencies between GHC and base
    - E.g. Foldable/Traversable/Generic deriving, list type
    - Should mostly be possible to move these things to ghc-prim (or ghc-base)
- Davean: taking this kind of approach will force people to upgrade to a new GHC before getting the benefits
- Michael: What problem are we trying to solve?
    - Ben
        - Future proof us against future changes to GHC
        - Shield users who aren't ready to update
        - Goal here isn't actually to split up base
        - Want to make it that people aren't directly importing a package that is tied into the GHC release cycle
    - Davean has heard people saying they want to fold Vector and containers into base. Maybe should handle via reexports to get the batteries included experience.
    - Michael: we're at a local optimum, lots of the things we're looking at (split base, reinstallable base, reexports, etc) are workarounds to deal with the pain of a transition period
- Ben: can we start on a repo to start trying out these ideas?
- Michael: let's create an std (bikeshed name), export types, and then include patched versions of bytestring, text, vector, etc that reuse that type
- Michael: we can put an explicit stream fusion layer into std
- Ben: we can keep a compatibility shim in the bytestring/text/vector packages doing implicit stream fusion, and the new code won't use it

Vector: unifying unboxed and boxed

- Ben: aren't we concerned about performance degradation?
- Michael: (1) if we can guarantee that it selects unboxed reliably and (2) we should expose a PrimVector to allow users to be completely explicit
- Ben: this will require some kind of closed type family + no orphans. One flavour would be a flavour of typeclass which explicitly prohibited orphan instances and a closed type family which allows the user to determine whether a given instance is in scope. E.g. WithInstance :: Constraint -> Maybe Constraint. The argument must be headed (syntactically?) by one of these special "always coherent" typeclasses (yuck).
- Davean: need to be certain that this doesn't kill compile time performance
- Ben: the coherence issue is tricky