For this sprint, we decided that optimization was not the most important feature to implement. The most needed feature at the moment is monadic IO, which requires typeclasses. However, typeclasses would be a bit difficult to implement in the time we have left, and would require some fairly major augmentations to the runtime, so instead we implemented type inference, which will come in very useful when typeclasses are implemented, as it saves the user the effort of needing to define the type for every expression they write. This change also introduces support for type variables of arbitrary kind, which is absolutely a must in order for typeclasses to support monadic IO.

Type inference works somewhat similarly to type checking, except the types of lower elements may have to be updated later as more types are determined. If the user has defined a type, the inferred type is checked against the user defined type, and ensures that the inferred type is a superset of the user defined type in terms of polymorphism. The user still has to define types for recursive functions though, as the type checker would get into an infinite loop otherwise in some cases. There is a way to infer recursive functions, as haskell does it just fine, it's something I have to figure out still.

Although arbitrary kinded type variables are allowed right now, there is no kind checking, so a malformed type expression like "m a b -> m a" is allowed, and may result in undefined behaviour. This can easily be solved by a kind inference and checking system, which is essentially a subset of the current type inference system.

To test the new changes, five test cases have been added. You can run them as usual.

inference.thc contains nearly the same code as functional.thc, however the types of most non-recursive functions have been removed, in order to test the type inference system.

kinds.thc contains a simple test of types of other kinds, defining an identity function "id :: m a -> m a"

either.thc tests an oversight in the previous test cases where only single parameter type constructors were tested

transformer.thc defines a maybe transformer, which is a very common pattern in haskell. This requires a type variable of kind "* -> *" in both the data definition, and in a function definition.

inferfail.thc contains a faulty function application, where no types are specified, and thus all types must be inferred, and a failure detected automatically.