Haskell's way of supporting ad hoc polymorphism. Any instance of `Typeclass` the user can write:

Example usage

In HVX, a user supplies an objective, constraints, variables, and method-specific parameters to a subgradient method. To solve the optimization problem

```
 minimize ||Ax||2 + ||y||2 + ||c + x - y||2
 subject to y ≤ 2
```

the user can write:

```
y = EVar "y"
ans = ellipsoidMinimize
radius = 1.0 e10 -- radius of initial ellipsoid sphere centered at origin
tolerance = 1.0e -10 -- U - L < tolerance termination condition
```

Why Haskell?

At the outset we chose the Haskell language for this project because:

- It has a very powerful type system. It is possible to implement DCP rules at the type level, and have the compiler enforce them during typechecking.
- It has an implementation of Automatic Differentiation in the form of the `double-precision` package. We leveraged this library to completely avoid writing any subgradient code ourselves. We eventually did write our own symbolic subgradient module due to some compatibility issues, but we still believe that automatic differentiation would be useful.
- The `hmatrix` package provides a simple interface to fast underlying matrix libraries such as BLAS, LAPACK, etc from Haskell. This allowed us to easily support vector valued variables, expressions, and functions.

Disciplined convex programming with types

HVVX implements the disciplined convex programming (DCP) framework within the Haskell type system, so the convexity of a user’s programs can be checked when the optimization routine is compiled, rather than when it is run. This has two benefits: first, it eliminates the performance overhead of parsing the DCP syntax tree when starting up a solver, and second, it prevents users from ever attempting to run programs that are not convex.

HVVX defines generalized algebraic data types (GADTs) representing functions and expressions. These algebraic data types combine the convexity type and monotonicity type of the expressions and functions they represent. Function application in HVX is only allowed when both the function and the expression argument have an appropriate type according to the DCP composition rules. This causes the Haskell compiler to refuse to compile programs that attempt to construct expressions of indeterminate convexity.

```
module Fun vex mon where
instance Applicable Convex Nondec Affine Nonmon Convex Nonmon
```

Subdifferentiation

HVVX obtains subdifferentials by computing them symbolically. It recursively applies the subgradient composition rule until a function is broken down into its constituent primitives, whose subdifferentials are known.

Our implementation of the subgradient composition rule is as follows:

```
• jacobianVar : EVar → EVar → EVar
• jacobianFun : Fun → Fun → Fun
```

Subgradient

HVVX currently includes two subgradient methods: the basic subgradient method and the ellipsoid method. The following code snippet shows a portion of our basic subgradient method:

```
subgradient : Int → ... 
```

Subgradient composition

```
-- Chain rule : ddx f(e) = dde f * ddx e
```

Subgradient implementation

```
thisSubgradient :: (forall e m . evalGradient (f x y z) vars) 
```

Subgradient convergence

```
if itr >= maxItr || vars == varsNext 
then (vars , ( evaluate obj vars ) @@ > (0 ,0) ) 
```

Acknowledgements

We would like to thank Bryan O’Sullivan and Professor David Mazieres for their help with all things Haskell. We would also like to thank Ahmed Bou-Rabee, Alon Kipnis, Brandon Jones, Ernest Ryu, Jaehyun Park, Linyi Gao, Mainak Chowdhury, Milind Rao, Nicholas Moehle, Professor Stephen Boyd, and Steven Diamond for giving us valuable feedback on our project throughout the quarter. We want to especially acknowledge Jaehyun and his partner Maurizion Calò for their 2011 class project entitled 3PY, which inspired this project.