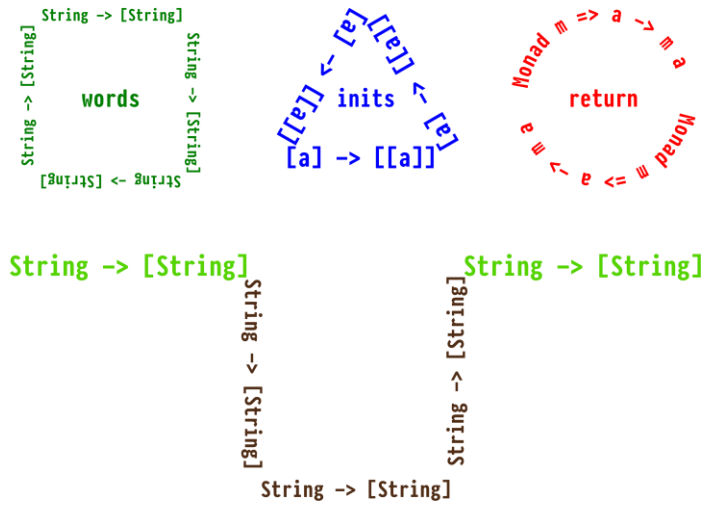
Hole Fit Plugins

- and the future of interactive programming in GHC



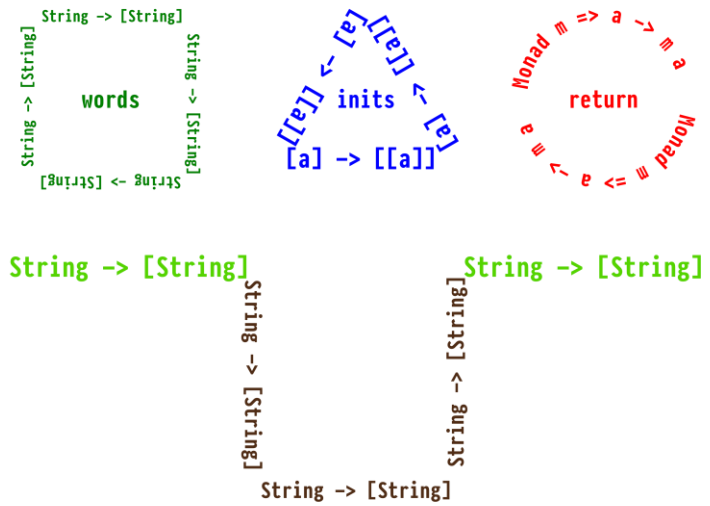
Matthías Páll Gissurarson

@Tritlo

(Chalmers University of Technology)

Hole Fit Plugins

- The Hol(e)y Grail for Plug-in' Holes



Matthías Páll Gissurarson

@Tritlo

(Chalmers University of Technology)

Valid Hole Fits

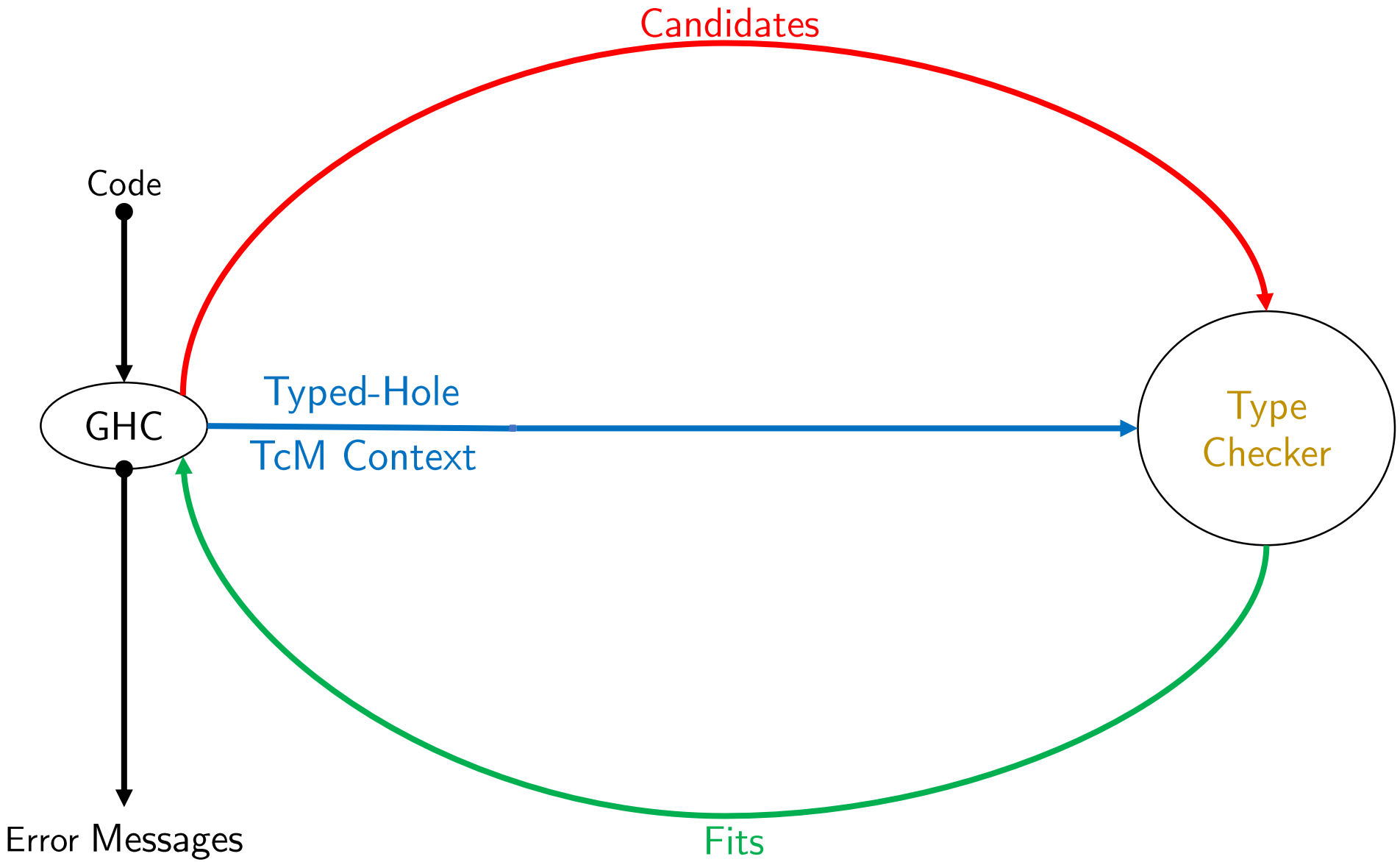
```
f :: Int
f = _ [(1 :: Int), 2, 3]

Found hole: _ :: [Int] -> Int
Valid hole fits include
  head :: forall a. [a] -> a
  last :: forall a. [a] -> a
  length :: forall (t :: * -> *) a.
                    Foldable t => t a -> Int
  maximum :: forall (t :: * -> *) a.
                    (Foldable t, Ord a) => t a -> a
  minimum :: forall (t :: * -> *) a.
                    (Foldable t, Ord a) => t a -> a
```

Refinement Hole Fits

```
{-# OPTIONS -frefinement-level-hole-fits=2 #-}
f :: Int
f = _ [(1 :: Int),2,3]

Found hole: _ :: [Int] -> Int
Valid refinement hole fits include
  foldl1 (_ :: Int -> Int -> Int)
    where foldl1 :: forall (t :: * -> *) a. Foldable t
                    => (a -> a -> a) -> t a -> a
  foldr1 (_ :: Int -> Int -> Int)
    where foldl1 :: forall (t :: * -> *) a. Foldable t
                    => (a -> a -> a) -> t a -> a
  foldl (_ :: Int -> Int -> Int) (_ :: Int)
    where foldl :: forall (t :: * -> *) b a. Foldable t
                   => (b -> a -> b) -> b -> t a -> b
```


Candidates
Code
GHC
Typed-Hole
TcM Context
Type
Checker
Error Messages
Fits

Valid Hole Fits

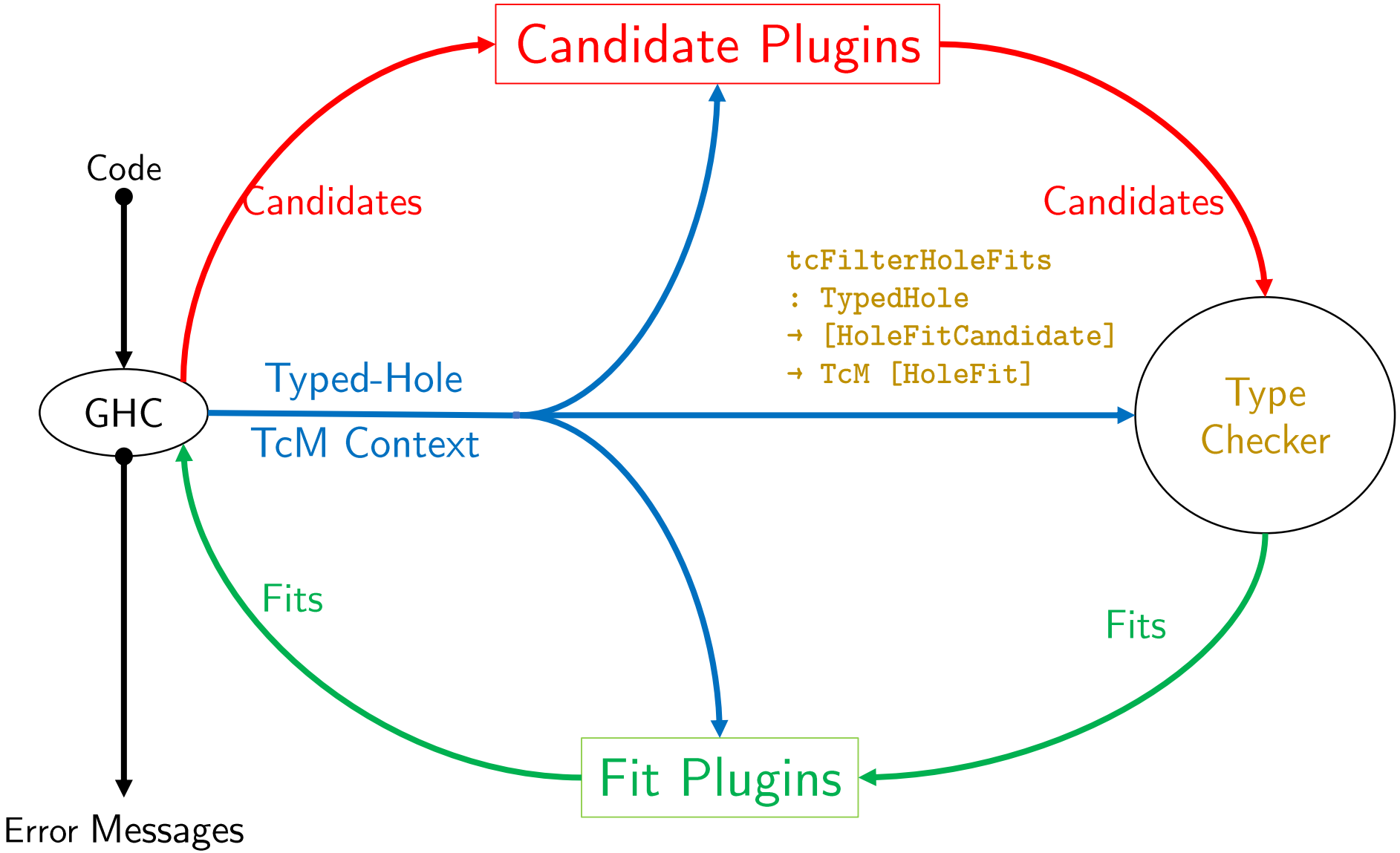
```
f :: Int
f = _ [(1 :: Int), 2, 3]

Found hole: _ :: [Int] -> Int
Valid hole fits include
  head :: forall a. [a] -> a
  last :: forall a. [a] -> a
  maximum :: forall (t :: * -> *) a.
             (Foldable t, Ord a) => t a -> a
  minimum :: forall (t :: * -> *) a.
             (Foldable t, Ord a) => t a -> a
```

Hardcoded!

Embedded in GHC!

```
CandPlugin : TypedHole → [HoleFitCandidate] → TcM [HoleFitCandidate]
```




```
FitPlugin : TypedHole → [HoleFit] → TcM [HoleFit]
```

Why plugins?

- Allow you to analyze, extend and experiment easily.
- Gives you access to all the GHC environment and function.
- Can depend on external tools like Z3 or Hoogle

Demo

```
{-# OPTIONS -fplugin=DjinnHoogleModPlugin #-}
module Main where
import Control.Monad

f :: (a,b) -> a
f = _invoke_Djinn

Found hole: _invoke_Djinn :: (a,b) -> a
Valid hole fits include
  (\ (a, _) -> a)
  (\ _ -> head (cycle (h (g ([])) ++ h (g ([])))))
  f :: (a, b) -> a
  fst :: forall a b.(a, b) -> a
```

```haskell
{-# OPTIONS -fplugin=DjinnHoogleModPlugin #-}
module Main where
import Control.Monad

f :: (a,b,c) -> a
f = _invoke_Djinn
```

```
Found hole: _invoke_Djinn :: (a,b,c) -> a
Valid hole fits include
  (\ (_, _, a) -> a)
    f :: (a, b, c) -> a
```

```
{-# OPTIONS -fplugin=DjinnHoogleModPlugin #-}
module Main where
import Control.Monad

f :: (a,b,c) -> a
f = _invoke_Hoogle

Found hole: _invoke_Hoogle :: (a,b,c) -> a
Valid hole fits include
  Hoogle: Data.Tuple.Utils fst3 :: (a,b,c) -> a
  Hoogle: Data.Tuple.HT fst3 :: (a,b,c) -> a
  g :: [a] -> [[a]]
  repeat :: forall a. a -> [a]
```

```
{-# OPTIONS -fplugin=DjinnHoogleModPlugin #-}
module Main where
import Control.Monad

g :: [a] -> [[a]]
g = _invoke_Hoogle

Found hole: _invoke_Hoogle :: [a] -> [[a]]
Valid hole fits include
  Hoogle: Data.List subsequences :: [a] -> [[a]]
  Hoogle: Data.List permutations :: [a] -> [[a]]
  g :: [a] -> [[a]]
  repeat :: forall a. a -> [a]
```

```
{-# OPTIONS -fplugin=DjinnHoogleModPlugin #-}
module Main where
import Control.Monad

h :: [[a]] -> [a]
h = _module_Control_Monad

Found hole: _module_Control_Monad:: [[a]] -> [a]
Valid hole fits include
  h :: [[a]] -> [a]
  join :: forall (m :: * -> *) a. Monad m => m (m a) -> m a
  msum :: forall (t :: * -> *) (m :: * -> *) a.
          (Foldable t, MonadPlus m) => t (m a) -> m a
  forever :: forall (f :: * -> *) a b.
             Applicative f => f a -> f b
```

Implementation

```
type CandPlugin = TypedHole
                  -> [HoleFitCandidate]
                  -> TcM [HoleFitCandidate]


type FitPlugin = TypedHole
                 -> [HoleFit]
                 -> TcM [HoleFit]


data HoleFitPlugin =
  HoleFitPlugin { candPlugin :: CandPlugin
                , fitPlugin :: FitPlugin }
```

```
data TypedHole = TyH { tyHRelevantCts :: Cts
                     , tyHImplics :: [Implication]
                     , tyHCt :: Maybe Ct }
```

```
data HoleFitPluginR = forall s. HoleFitPluginR  {
      hfPluginInit :: TcM (TcRef s)
    , hfPluginRun  :: TcRef s -> HoleFitPlugin
    , hfPluginStop :: TcRef s -> TcM ()
}
```

```haskell
fitPlugin :: FitPlugin
fitPlugin hole hfs = do
    dflags <- getDynFlags
    let tyString =
           showSDoc dflags . ppr . ctPred <$> tyHCt hole
    res <- case tyString of
             Just ty -> liftIO $ searchHoogle ty
             _ -> return []
    return $ (take 2 $ map (RawHoleFit
             . text . ("Hoogle: " ++)) res) ++ hfs
  where searchHoogle ty =
          lines <$> (readProcess "hoogle" [(show ty)] [])

hfp :: [CommandLineOption] -> Maybe HoleFitPluginR
hfp _ =
 Just (fromPureHFPlugin $ HoleFitPlugin (const return) fp)

plugin :: Plugin
plugin = defaultPlugin { holeFitPlugin = hfp,
                         pluginRecompile = purePlugin }
```

We need more interactivity!

```
{-# OPTIONS -fplugin=DjinnHoogleModPlugin #-}
module Main where


g :: [a] -> [[a]]
g = _{hoogleLookup "+Control.Applicative"}0

Found hole: _{…}0 :: [a] -> [[a]]
Valid hole fits include
  Hoogle: Control.Applicative
    some :: Alternative f => f a -> f [a]
  Hoogle: Control.Applicative
    many :: Alternative f => f a -> f [a]
  g :: [a] -> [[a]]
  repeat :: forall a. a -> [a]
```

```
{-# OPTIONS -fplugin=LiquidPlugin #-}
module Main where

g :: [Int] -> Int
g = _{}0ses "Data.List"
      . satisfies "x:[Int] -> {v:Int | len x == v}"}0
```

```
Found hole: _{…}0 :: [Int] -> Int
Valid hole fits include
  head :: forall a. [a] -> a
  last :: forall a. [a] -> a
  length :: forall (t :: * -> *) a.
                    Foldable t => t a -> Int
  maximum :: forall (t :: * -> *) a.
                     (Foldable t, Ord a) => t a -> a
  minimum :: forall (t :: * -> *) a.
                     (Foldable t, Ord a) => t a -> a
```

Conclusion

- Hole Fit plugins are available on GHC HEAD
- Makes experimenting with hole fits a lot easier
- Relieves us of deciding "the right thing to do"
- Already in use in the wild[1]
- Give us a way to have a dialogue with the compiler

[1] https://github.com/jippiedoe/Recursively-finding-valid-hole-fits

Takk fyrir mig!

https://github.com/Tritlo/ExampleHolePlugin/