CHALMERS
UNIVERSITY OF TECHNOLOGY

Matthías Páll Gissurarson
pallm@chalmers.se
Department of Computer Science and Engineering

Hole Fit Plugins for GHC

– making GHC hole fits customizeable for richer development and experimentation.

Problem

Valid hole fit suggestions were introduced in GHC 8.4 and extended in 8.6 and 8.8. However, since they are bundled with GHC, their functionality is hampered by the need for their implementation to be compatible with GHC's distribution model. This means that interacting with tools such as *Hoogle*, *QuickCheck* and *Djinn,* constraint solvers such as *Z3*, and machine learning tools is not an option.

Approach and Contributions

To alleviate the constraints that GHC's distribution model places on valid hole fit suggestions, I have extended GHC's plugin framework with a new type of plugin, *Hole Fit* plugins. These allow developers to write plugins that add, remove or re-order candidate hole fits and hole fit suggestions. The main result of this work is a patch that has been merged to GHC HEAD that adds *Hole Fit* plugins, which are expected to be generally available in GHC 8.10.1. Additionally, three proof of concept plugins have been written that interface with *Djinn*, *Hoogle* and *QuickCheck* respectively. As an example, the code shown to the right uses a plugin that invokes Djinn to synthesize simple functions, invokes Hoogle to search from the type of the hole, and can filter by module, in addition to showing the regular valid hole fit suggestions.

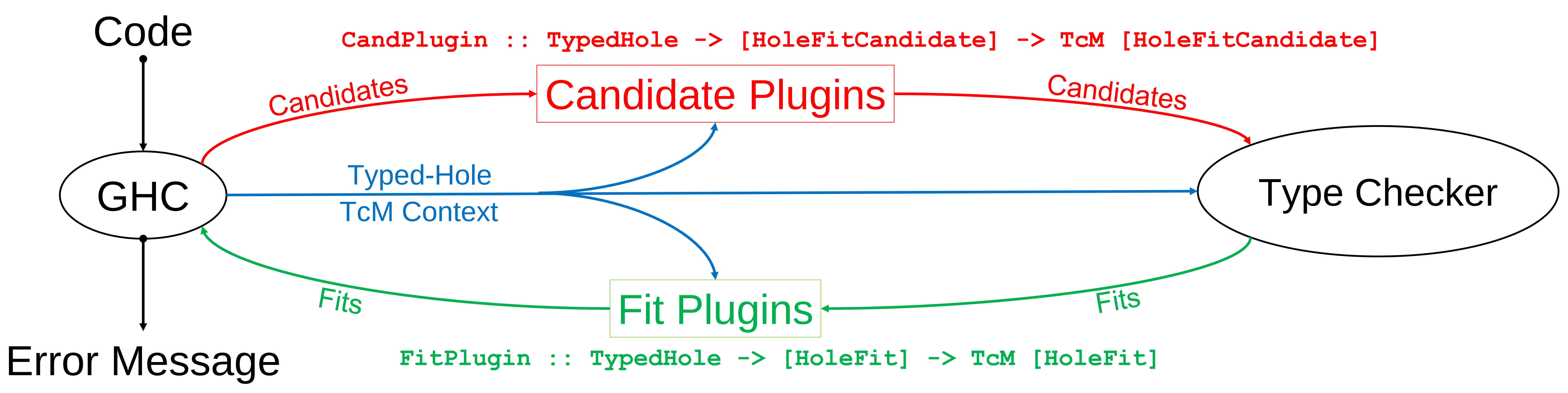
```
{-# OPTIONS -fplugin=DjinnHoogleModPlugin #-}
module Main where
import Control.Monad
f :: (a,b) -> a
f = _invoke_Djinn
g :: [a] -> [[a]]
g = _invoke_Hoogle
h :: [[a]] -> [a]
h = _module_Control_Monad
```

Artifact available!
git.io/fj78g

```
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

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]

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
```



```
CandPlugin :: TypedHole -> [HoleFitCandidate] -> TcM [HoleFitCandidate]
FitPlugin :: TypedHole -> [HoleFit] -> TcM [HoleFit]
```

Background

Richly typed functional programming languages facilitate a style of programming called ***Type-Driven Development*** *(TDD)*, in which users allow the types in a program to guide development. This style encourages the programmer to provide as much typing to the compiler up front as they can, which means that there is a lot of information available to the compiler to produce rich, informative error messages.

Typed-holes were added in GHC 7.8 to allow developers to interact with this typing information, by making the compiler produce an informative error message when a hole (denoted by a '_') is encountered in an expression in the code. The error message was extended further in GHC 8.4, 8.6 and 8.8 with **valid hole fit suggestions**, which uses the typing information and functions in scope to suggest valid expressions that can be put in place of the hole.

Compiler plugins are a feature of GHC that allow users to write *plugins* that are invoked at particular times during compilation. *Type Checker* plugins allow users to extend the type checker, *Core* plugins apply transformations to GHC's intermediate language *Core,* and *Source* plugins allow users to change the compilation pipeline at various stages: after parsing, after type checking, after renaming, during macro expansion, and when modules are being loaded.