Liquid Haskell as a GHC Plugin
HIW 2020

Alfredo Di Napoli, Well-Typed LLP
Ranjit Jhala, University of California at San Diego
Andres Löh, Well-Typed LLP
Niki Vazou, IMDEA Software Institute
2020-08-28
Liquid Haskell in brief

- Liquid Haskell\(^\text{1}\) is a **Refinement Type Checker** for Haskell.
- **Refinement Types** are types with a logical predicate attached to them.

\(^\text{1}\)http://ucsd-progsys.github.io/liquidhaskell
Example

{-@ safeDiv :: Int -> {y : Int | y /= 0} -> Int @-}

safeDiv :: Int -> Int -> Int
safeDiv = div

If we try to call safeDiv 3 0, Liquid Haskell rejects the program as UNSAFE.
Previously, integrating Liquid Haskell into an existing codebase was not a zero-setup operation, which hindered larger-scale adoption.

- Available as an executable
Previously, integrating Liquid Haskell into an existing codebase was not a zero-setup operation, which hindered larger-scale adoption.

- Available as an executable
- “Hardcoded” prelude
Previously, integrating Liquid Haskell into an existing codebase was not a zero-setup operation, which hindered larger-scale adoption.

- Available as an executable
- “Hardcoded” prelude
- No \textit{ghcid} or \textit{ghcide} support
Starting from version 0.8.10.1, Liquid Haskell is available as a GHC Plugin.

Goals:

- Piggyback on GHC
Starting from version 0.8.10.1, Liquid Haskell is available as a GHC Plugin.

Goals:

- Piggyback on GHC
- Allow users to ship their own specifications as well as re-use existing ones
Starting from version 0.8.10.1, Liquid Haskell is available as a GHC Plugin.

Goals:

▶ Piggyback on GHC
▶ Allow users to ship their own specifications as well as re-use existing ones
▶ Support IDE tools
Starting from version 0.8.10.1, Liquid Haskell is available as a GHC Plugin.

Goals:

- Piggyback on GHC
- Allow users to ship their own specifications as well as re-use existing ones
- Support IDE tools
- Make Liquid Haskell easier for people to try and use
Demo
The Liquid Haskell architecture in brief

Bird’s eye view over Liquid Haskell’s architecture:

- `A.hs`
- `A.spec`
- `depsOfA`
- `HashSet LiftedSpec`
- `[CoreBind]`
- `Bare Spec`
- `makeTargetSpec`
- `Lifted Spec`
- `Target Spec`
Brief GHC Plugin architecture recap

- **Pipeline Stage**
- **Plugin Action**

GHC Plugin

- `parsedResultAction` (parse)
- `renamedResultAction` (rename)
- `typecheckResultAction` (typecheck)
- `desugar`
- `installCoreToDos` (Core)
  - `codegen`
- `optimise`
Attempt 1: Proper pipeline split
Attempt 1

Idea: Follow the natural lifecycle of the GHC pipeline.

While probably more elegant, this didn’t work for most programs.

Challenge 1

Liquid Haskell requires access to the unoptimised [CoreBind], and we cannot assume anything about the program’s -O level.
Attempt 2: Duplicate (some) work
Use the input DynFlags to generate another version with optimisations switched off.

Use the latter to parse, typecheck and desugar the module again (!), to extract a suitable [CoreBind].

This worked, until we tried to use the plugin with ghcide.

Challenge 2

When checking an input module, ghcide calls only the typeCheckResultAction hook of any registered GHC plugin.
Attempt 3: Reduce the plugin surface
The final design of the plugin does **everything** in the `typeCheckResultAction`, so that we can integrate the plugin with *ghcide*.

The double parsing, typechecking, desugaring is still necessary.
Even with “Attempt 3” implemented, we couldn’t get ghcid to work properly. The issue was twofold:

- GHC issue #18070\(^2\) prevented plugins to be properly used on 8.10.1. This is **now fixed** and is part of the **8.10.2** release;
- We had to patch **ghcid**\(^3\) to fully support GHC plugins.

Once we fixed the above, we got **ghcid working**!

---

\(^2\)[https://gitlab.haskell.org/ghc/ghc/-/issues/18070]
\(^3\)[https://github.com/digital-asset/ghcid/pull/698]
Success, at last

```haskell
33 {-@ one :: {v:Int \mid v = 1} @-}
35 one :: Int
36 one = 2
37 {-@ as
38 notThr
39 notThr
40 notThr
41 notThr
42 tw
43 two :: VV : {v : GHC.Types.Int \mid v == 2}
44 two = .
```

```
[typecheck] [E] Liquid Type Mismatch

The inferred type VV : {v : GHC.Types.Int \mid v == 2}

is not a subtype of the required type VV : {VV : GHC.Types.Int \mid VV == 1}
```
We offer drop-in replacements for some popular Haskell libraries:

- liquid-base
- liquid-containers
- liquid-bytestring
- ...

We also propose a simple PVP scheme to track the dependency on the upstream package, for users willing to contribute to the ecosystem by adding new packages:

`liquid-<package-name>-A.B.C.D.X.Y`

- A.B.C.D track the upstream package,
- X.Y allow for LH-related bug fixes and breaking changes.
Have we achieved our goals?

- Harness GHC for recompilation avoidance and dependency resolution ✓
- Allow users to ship their own specifications ✓
- Support IDE tools ✓

We hope the plugin will help with Liquid Haskell’s adoption.
Conclusions and lessons learned

The low-level nature of the GHC API makes tricky to write plugins which modify the compilation pipeline but need to be compatible with tools that extend the frontend (like ghcide).

- Each plugin action is fairly stateless, no first-class support to pass user’s state around.
- Not having access to the “unoptimised” [CoreBind] complicated the design.
- Calling the GHC API inside some actions might lead to surprising results (like ghci looping).
- The fact that our plugin worked with ghcid and ghcide with minor adjustments was very satisfactory.

Start refining your types today, use -fplugin=LiquidHaskell in your next project!