Multiple Home Packages for GHC

Abstract

Haskell tooling has improved considerably over the last year. Haskell IDE Engine was able to build a user-base that is happy with the feature set. Moreover, competing projects have been created to explore different design spaces. With the rise of these tools, certain limitations became apparent in the GHC API. For example, it is complicated to propagate updates between dependent packages.

In this proposal, I intend to enable GHC and GHCi to work with multiple packages at once. Most importantly to the end user, this allows development of multiple packages in GHCi. There are other workflows which can benefit from such functionality, such as incremental compilation and Haskell Language Server.

About me

I have started contributing to Haskell IDE Engine over a year ago. Since then I have contributed a great amount to the project and become co-maintainer of the project hie-bios which is currently the interface between GHC API and various Language Servers, such as Haskell IDE Engine and ghcide. Additionally, I have contributed to Cabal by implementing the show-build-info command.

At the time of this writing, I am doing my master's degree in software engineering at the Vienna University of Technology.

Problem Statement

The main motivation for this proposal is to help IDEs provide a seamless developer experience. Currently, two of the bigger IDE projects in Haskell are ghcide and Haskell IDE Engine. Both of these projects aim to support a workflow where developers can work on multiple packages, such as a package's library and executable at the same time. The work-force behind it is GHC itself, which is responsible for actually compiling a user's source files. Currently when working in a single GHC session, a so-called HscEnv, only a single package can be compiled at a time. The session is then saved in case the package changes and needs to be re-compiled. To work around this limitation, Haskell IDE Engine decided to maintain one `HscEnv` per package to load. While it is wasteful to have information, such as compilation artefacts, duplicated this works reasonably well in practice. However, a serious drawback of this approach is, that if a user works on two packages at the same time, where one depends on the other and there are changes to the package which others depend on, Haskell IDE Engine possesses no knowledge that the other package needs to be recompiled as well. This leads to Haskell IDE Engine showing outdated documentation and diagnostics. The project `ghcide` explores a different design space than Haskell IDE Engine, and is based on the concept of the shake library which is comparable to a `Makefile` and was recently adopted as a build-tool for GHC itself. It mirrors certain parts of the GHC compilation process and maintains its own module graph which enables reloading modules that actually need to be updated when a source file changes. At the time of this writing, ghcide only provides experimental support for loading multiple packages at once. Multiple `HscEnv` are avoided by tricking the compiler into thinking that all the local packages, are actually part of the same package by setting all packages' identifier to a fake unit-id. A disadvantage of this is that GHC needs to be tricked in order to provide what `ghcide` needs and that parts of the GHC API need to be re-implemented, which has a higher maintenance cost.

Another motivation is to speed up development time with `GHCi`. For example assuming a developer is developing a package, e.g. an executable, by loading the package into `GHCi`. After implementing features, they need to fix a bug in some of the packages the executable depends on. After fixing it a simple reload of the executable package is not enough. `GHCi` does not notice that one of the dependencies has been modified. To apply the changes, the whole `GHCi` session needs to be restarted.

Implementing this proposal will help to solve the problems of IDEs stated above, and improve development speed when working with `GHCi`. Maintaining multiple GHC sessions will no longer be necessary, as the components will be put into the same session where `GHC` will take care of the rest. Custom tricks to make the compiler do what the developers want will no longer be necessary. Restarting `GHCi` will also no longer be necessary, because developers will be able to load multiple components at once, with `GHCi` being able to reload the modules that need reloading.

Tasks

Familiarise with GHC code-base

GHC has a large and complex code-base and it is probably impossible to prepare for the planned project perfectly. However, I plan to spend some time in the code-base, read documentation and work on simpler issues to familiarise myself with the project structure and development cycle.

Implement Multiple Home Modules

The main part of this proposed project is modifying the type of HscEnv. Currently, this holds one set of dynamic compilation flags and one value of Home Package Table (HPT). The dynamic compilation flags can be different for every package to be loaded, for example the optimisation level can be different. To lift this limitation, the dynamic compilation flag within a `HscEnv` is replaced by a map from a unit-id (used to identify packages) to a tuple of dynamic compilation flags and a HPT. This makes it possible to load multiple packages into the same `HscEnv`. This approach is not without its difficulties, when two packages expose a module with the same name, GHC needs to be able to tell them apart. Avoiding this can be done by resolving imports in the Home Package Table.

This task has already been started in the PR !935. As some time has passed since the initial PR, I expect that it needs some adjustments.

Rebasing the changes and finishing off the implementation is the first step for this project.

This task is absolutely required for this project. Estimated time: 6 weeks.

Implement a suitable CLI

Since every package may have a different set of dynamic compilation flags, GHC itself needs to be made aware of this. Currently, you may pass only a single set of dynamic compilation flags for the current package to compile. This limitation needs to be lifted to enable tools such as `Cabal` and `Stack` to load multiple packages at once.

Since new feature requests go through the GHC proposal process, it will be neccessary to get the proposed changes accepted. There already exists a write-up of the proposed changes in PR #263. However, the write-up did not specify a suitable CLI for Multiple Home Packages, yet. Therefore, I will push a CLI design and implement it, once it reaches consensus.

One goal of the CLI is for it to remain backwards-compatible. This makes a clean design more diffcult, because currently GHC supports that dynamic compilation flags are interleaved with flags that are unrelated to compilation of the package. To illustrate, assume the following invocation of GHC:

```
ghc -i. -v -O2 Lib.hs
```

This informs GHC to be verbose (`-v`) and compile the module `Lib.hs` with optimisation level two (`-O2`) and adds an include path for module imports (`-i.`). As can be seen, the dynamic compilation flags can be interleaved with flags that affect GHC. This needs to work after implementing this proposal as well.

We propose that this behaviour is preserved and add an alternate CLI format:

```
--package-flags <unit-id> {<dynamic compilation flags>, ...}
```

It informs GHC for which package (identified by a unit-id) the dynamic compilation flags are. This flag can be repeated for each `<unit-id>` and the options ought to be merged with later entries overriding newer ones. Additionally, providing the `--package-flags <unit-id>` flag implies the flag `--this-unit-id <unit-id>`.

The proposed CLI format would be incompatible with the previous format, e.g. it is an error to provide both types of arguments, e.g.

```
ghc -i. -v --package-flags unit1 -O2 Lib
```

shows an error message. This is intended to force users to be more strict about their arguments and to avoid potential confusion about what such invocations should actually do. It would be reasonable to assume that you can have global compilation flags which apply to all packages and compilation flags that apply to only a single package. We think that explicitly setting common arguments for each package is more clear and avoids users mixing up the new CLI with the old one.

This task can likely take place in parallel with other tasks, but might take a considerable amount of time since there is waiting for feedback involved.

This task is absolutely required for this project. Estimated time: 3 weeks

Modify cabal to make use Multiple Home Packages

I plan to put the newly implemented feature to good use and extend the `cabal repl` command to be able to load multiple components at once. An example call might look like this:

```
cabal repl lib:packages exe:package
```

to load the library component of a package and an associated executable at once.

This task is optional and nice to have. Estimated time: 2 weeks

Modify stack to make use Multiple Home Packages

As of the issue !10827, the build-tool stack is already implementing the proposal's feature. However, since they have no support from GHC, the following problems can not be solved:

- Loading multiple packages fails if two of them expose the same module.
- A GHCi session manages only a single set of compilation flags, but each package might be compiled with different ones.

These restrictions would be mitigated by this project, and can be implemented in `stack`.

This task is optional and nice to have. Estimated time: 2 weeks

Related Work

The work of Daniel Gröber from the Google Summer Of Code 2019 is related to this proposal. It mitigated the need to have only one GHC session per process. This proposal now adds the possibility to have multiple packages per session, increasing efficiency and improving the user experience. It is therefore a natural extension of the existing work.