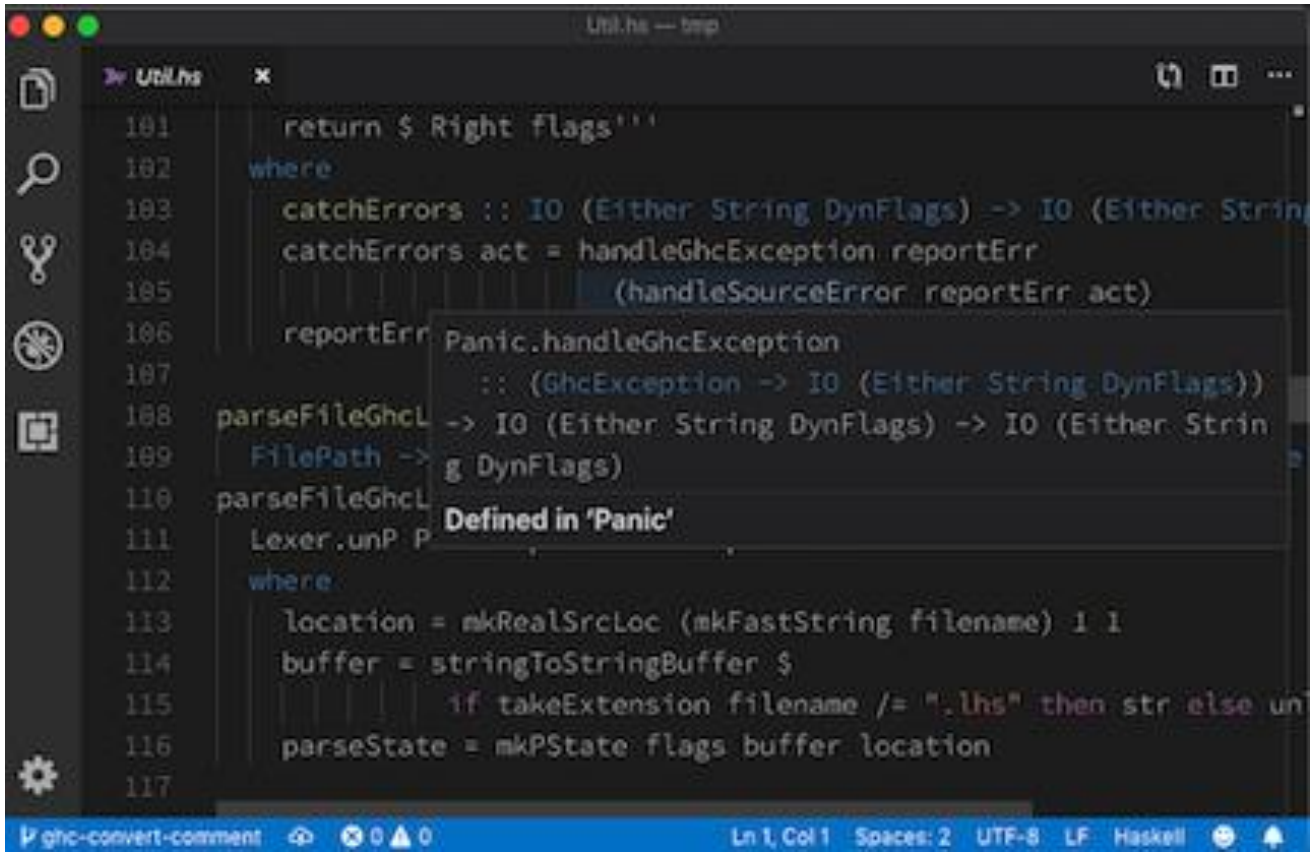
Making a Haskell IDE



Neil Mitchell, https://ndmitchell.com

Poll

- Which Editor?
  - VS Code | Emacs | Vim | …
- What feedback mechanism?
  - haskell-ide-engine | ghc-mod | GHCid | GHCi | …
- Code exploration?
  - haskell-ide-engine | ghc-mod | Hoogle | grep | …
- Who wants more?

“with the IDE I'm about
25% more productive
than without it”
Me, Sep 2014

2004-2014: Basic tools

- Text editor with Syntax coloring (TextPad)
- Hoogle – search, https://hoogle.haskell.org
- Hugs/GHCi for fast reloading

Cycle: Edit. Save. Switch. :r. Find error. Repeat.

2014-2019: GHCid

- Wrote GHCid (GHCi + a bit of an IDE)

Cycle: Edit. Save. ~~Switch. :r. Fi~~nd error. Repeat.

- Saved switching to GHCi
- Saved typing :r
- Reformatted errors better to reduce finding
- Huge productivity boost!

2019: hie-core

- A real IDE (although not best of breed)

Cycle: Edit. ~~Save. Switch. :r. Find~~ error. Repeat.

- Full type checking on every single keystroke
- Errors inline and integrated
- Plus some code navigation stuff
- Huge productivity boost!

Demo

Hard Truths

- Setting up hie-core isn't as easy as it should be
  - I'll explain how in this talk
- hie-core doesn't have enough users to be viable
  - You could use it! (No project does at the beginning)
  - And it does have commercial backing (Digital Asset)
- Writing an IDE isn't easy
  - I'll explain how in this talk
- Hacking an IDE *is* easy (if well designed)

Installing hie-core

- https://tinyurl.com/hie-core (1)
  - Install hie-core and hie-bios from GitHub
  - Install VS Code extension
  - Check your project works with hie-bios/hie-core
    - The hard bit!
  - Use through VS Code
    - Works from any LSP editor, e.g. Emacs

(1) https://github.com/digital-asset/daml/tree/master/compiler/hie-core

The hard bit

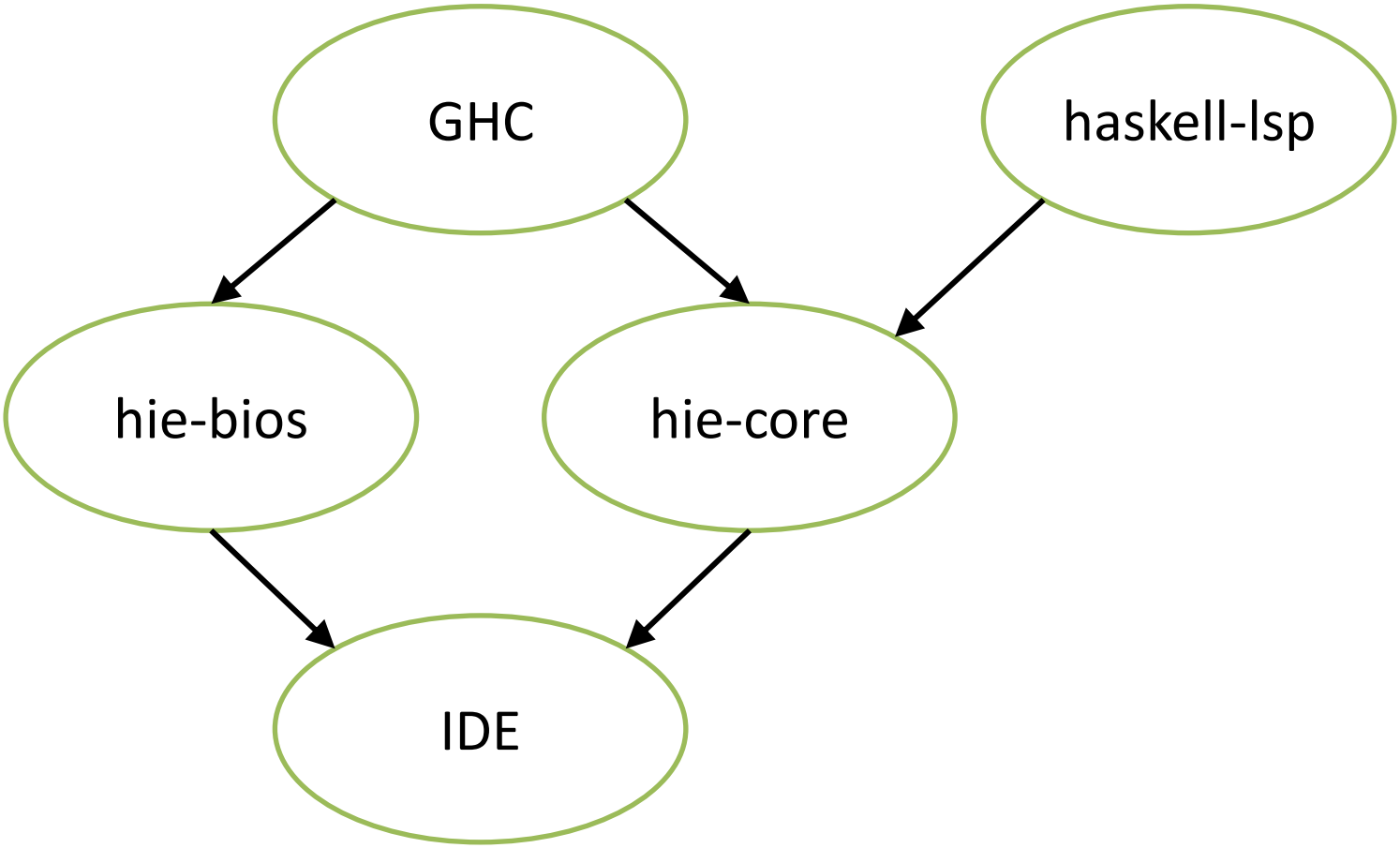
- Get it so your project can be loaded

```
/shake$ hie-core
... snip ...
Files that worked: 152
Files that failed: 4
 * .\model\Main.hs
 * .\model\Model.hs
 * .\model\Test.hs
 * .\model\Util.hs
Done
```

Setting up hie-bios (hie.yaml)

```
cradle:
  direct:
    arguments:
      - -ignore-package=hashmap
      - -Wunused-binds
      - -Wunused-imports
      - -Worphans
      - -isrc
      - src/Test.hs
      - src/Paths.hs
      - -idist/build/autogen
```

hie-core architecture



Division of responsibilities

- GHC – actually compile Haskell
- hie-bios – how to set up a GHC environment
  - Use “cradles”, direct, cabal, stack …
- hie-core – how to orchestrate compilations
- haskell-lsp – the LSP protocol
- IDE – 20 lines + 100 lines to help you debug
  - Inside hie-core, for now

Other Haskell IDEs

- GHCid – always reliable
- IntelliJ – good IDE, if you like IntelliJ
- Leskah – integrated, has its own editor
- Intero – tightly integrated into Stack, Emacs
- haskell-ide-engine – most closely related
  - hie-bios and haskell-lsp are parts of it
  - hie-core might one day become the core of it?
  - Also has hlint/hoogle etc integration
- Many others have come (and mostly gone)

Inside the IDE

IDEs are hard

- At one point I was responsible for six things, one of which was the IDE
- I thought the IDE was medium hard
- The IDE was the hardest
- Three approaches later, we settled on a working design

- Everything else has a paper to read!

Basic idea

- Set up dependencies
  - FilePath > Contents > Parse > Imports > TypeCheck
- Every time anything changes (e.g. keystroke)
  - Abort whatever is ongoing
  - Restart from scratch, skipping things that haven’t changed
- Report errors as you get them

IDE = Build System

- Yes:
  - Dependencies
  - Incremental minimal recomputation
  - Parallelism
- No:
  - Errors propagate and persist weirdly
  - Sometimes want stale data
  - Need to maintain diagnostics in the IDE too
  - Incremental in a slightly different way
  - Garbage collection

Build on top of Shake

- https://shakebuild.com/
- A Haskell build system – express dependencies
- Builds a K/V map
- Allows dynamic dependencies, Haskell values
- Proper doesFileExist tracking
- We added to Shake:
  - Priorities
  - In-memory no serialisation version

Development.IDE.Core.Shake

- Wrapper over Shake
- Stores values in its own Key/Value map, instead of in Shake
  - Allows garbage collection, accessing stale data
  - Errors propagate and persist, diagnostics in IDE
  - Removes Shake serialisation constraints

Key and Value types

- Key's are key name and Haskell filename
  - E.g. (TypeCheck, "Foo.hs")
  - Allows garbage collection and error reporting
- Values are optional, and have a list of errors
  - E.g. ([FileDiagnostic], Maybe TcModuleResult)
  - ([_], Just _) for warnings
  - ([], Nothing) only good for propagated errors
  - In practice, use exceptions to imply ([], Nothing)

The GHC API

- A scary place
- IORef's hide everywhere
- Huge blobs of state (HscEnv, DynFlags)
- The GHC Monad
- Lots of odd corners
- Lots of stuff that is not fit for IDE (e.g. downsweep)

<rant />

- Warnings from the type checker

```
data HscEnv = HscEnv
  {hsc_dflags :: DynFlags -- 148 fields
  ,hsc_targets :: [Target]
  ,hsc_mod_graph :: ModuleGraph
  ,hsc_IC :: InteractiveContext
  ,hsc_HPT :: HomePackageTable
  ,hsc_EPS :: IORef ExternalPackageState
  ,hsc_NC :: IORef NameCache
  ,hsc_FC :: IORef FinderCache
  ,hsc_type_env_var :: Maybe (Module, IORef TypeEnv)
  ,hsc_iserv :: MVar (Maybe IServ)
  }
```

Wrap the GHC API Cleanly

- We want “pure” functions (morally)

```
typecheckModule
  :: HscEnv
  -> [TcModuleResult]
  -> ParsedModule
  -> IO ([FileDiagnostic], Maybe TcModuleResult)
```

Rules from Wrappers

```
type instance RuleResult TypeCheck = TcModuleResult

define $ \TypeCheck file -> do
  pm <- use_ GetParsedModule file
  deps <- use_ GetDependencies file
  tms <- uses_ TypeCheck (transitiveModuleDeps deps)
  packageState <- useNoFile_ GhcSession
  liftIO $ typecheckModule packageState tms pm
```

Two Extensibility Points

1. Can define new values on the dependency graph
   - E.g. result of some expensive analysis pass

2. Can define new LSP handlers
   - setHandlersDefinition <> setHandlersHover <> setHandlersCodeAction

LSP Handlers

```
onHover :: IdeState -> PositionParams
                -> IO (Maybe Hover)
onHover ide (Params uri pos) = do
  …
  v <- runAction ide $ do
      use GetSpanInfo uri
  ….
```

Internal Architecture Summary

- Key/Value mappings which depend on each other
  - Wiring GHC functions and types into a graph
- Request comes in from LSP
  - Compute some values from keys
  - Format that information appropriately
- Lots of plumbing

Where we go off-piste

- GHC dependency graph is not incremental
  - Give it all files, get all results
- We want to get the dependencies of a file ourselves
  - If there are cycles, we want to still work elsewhere
  - Don’t want to have to do everything up front
  - Con: Makes TH, CPP etc harder
- Needs abstracting and sending upstream

Shake was a good idea

- IDE is a very natural dependency problem
- Robust parallelism
- Thoroughly debugged for exception handling
  - GHC API has a few issues in corner cases here
- Has good profiling (caught a few issues)
- Has lots of features – we could replicate the end state, but not the path there

Shake isn’t perfect

- Imagine two independent modules A, B
- If you are compiling A, and anything (e.g. B) changes you give up and restart
  - Ideally would *suspend* and see if its still useful
  - Not a thing GHC offers!
  - All hacks a bit like it are hard with Shake

Hacking

Contributing to hie-bios

- Hack here to make the start-up experience better
- Ideally: all projects work “out the box”
- A Cradle defines how to load, e.g. with Cabal
  - Calls “cabal repl --with-ghc=myscript”
  - Looks for the arguments to load ghci
- https://github.com/mpickering/hie-bios

Contributing to hie-core

- Support TH, more CPP, source plugins etc
- Hack here to add new features – completion, Hlint, Hoogle, refactoring
- Some will want to be hie-core plugins, of which there are currently zero (but is an API)
- Currently requires an (inoffensive) CLA
- https://github.com/digital-asset/daml compiler/hie-core

DAML by Digital Asset

- DAML is a programming language close enough to Haskell to share the same IDE core
- Designed for DLT stuff
- Everything is open source
- They're currently hiring engineers in Zurich and New York (I used to work there)
- Try it: https://webide.daml.com

Contributing Editor Plugins

- The VS Code plugin is pretty much done
- But plugins for other editors are welcome
  - Should just be the standard LSP approaches

Try it out

Bugs

GHC 8.4 support

.hi loading

Auto imports

Completion

Use it for real

HLint

Multi HscEnv

Add tests

Omulu

More code fixes

Credits

- Alan Zimmerman, Matthew Pickering, DA…