HIE files in GHC 8.8

Zubin Duggal Matthew Pickering

Introduction

.hie files - why do they exist?

- Tooling like haddock and HIE need to recompile source to get access to semantic information

.hie files - why do they exist?

- Tooling like haddock and HIE need to recompile source to get access to semantic information
- Semantic information is completely unavailable for code that doesn't compile.

.hie files - why do they exist?

- Tooling like haddock and HIE need to recompile source to get access to semantic information
- Semantic information is completely unavailable for code that doesn't compile.
- No reasonable way to get information about code in dependencies

.hie files - why do they exist?

- Tooling like haddock and HIE need to recompile source to get access to semantic information
- Semantic information is completely unavailable for code that doesn't compile.
- No reasonable way to get information about code in dependencies
- Setting up a GHC session that can load what you want it to load is hard

Contents of .hie files

- The original source of the file we compiled
- An interval tree, where each interval corresponds to a span in the source

```
Node { nodeInfo :: NodeInfo type
     , nodeSpan :: RealSrcSpan
     , nodeChildren :: [HieAST type]
     }
```

```
foo = bar where bar = 1 : foo
^^^   ^^^       ^^^   ^ ^ ^^^
                      ^^^^^^^
                ^^^^^^^^^^^^^
      ^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

Information captured in nodes

- The type(s) assigned by GHC to this Node, if any

Information captured in nodes

- The type(s) assigned by GHC to this Node, if any
- The name of the GHC AST constructors that this Node corresponds to

Information captured in nodes

- The type(s) assigned by GHC to this Node, if any
- The name of the GHC AST constructors that this Node corresponds to
- The identifiers that occur at this span

Information captured in nodes

- The type(s) assigned by GHC to this Node, if any
- The name of the GHC AST constructors that this Node corresponds to
- The identifiers that occur at this span
- Information about the identifiers, like:

Information captured in nodes

- The type(s) assigned by GHC to this Node, if any
- The name of the GHC AST constructors that this Node corresponds to
- The identifiers that occur at this span
- Information about the identifiers, like:
  - Are they names or modules

Information captured in nodes

- The type(s) assigned by GHC to this Node, if any
- The name of the GHC AST constructors that this Node corresponds to
- The identifiers that occur at this span
- Information about the identifiers, like:
    - Are they names or modules
    - Their type, if any

Information captured in nodes

- The type(s) assigned by GHC to this Node, if any
- The name of the GHC AST constructors that this Node corresponds to
- The identifiers that occur at this span
- Information about the identifiers, like:
  - Are they names or modules
  - Their type, if any
  - The context in which they occur: Are they being defined or used, imported, exported, declared etc..

Information captured in nodes

- The type(s) assigned by GHC to this Node, if any
- The name of the GHC AST constructors that this Node corresponds to
- The identifiers that occur at this span
- Information about the identifiers, like:
  - Are they names or modules
  - Their type, if any
  - The context in which they occur: Are they being defined or used, imported, exported, declared etc..
  - If they are being defined, the full span of the definition, and an approximated scope over which their definition can be used.

Information captured in nodes

- The type(s) assigned by GHC to this Node, if any
- The name of the GHC AST constructors that this Node corresponds to
- The identifiers that occur at this span
- Information about the identifiers, like:
  - Are they names or modules
  - Their type, if any
  - The context in which they occur: Are they being defined or used, imported, exported, declared etc..
  - If they are being defined, the full span of the definition, and an approximated scope over which their definition can be used.
- Types (stored in a hash consed representation)

Generating .hie files

- Pass the option -fwrite-ide-info to ghc to generate them next to .hi/.o files

Generating .hie files

- Pass the option -fwrite-ide-info to ghc to generate them next to .hi/.o files
- Can control path with -hiedir

Generating .hie files

- Pass the option -fwrite-ide-info to ghc to generate them next to .hi/.o files
- Can control path with -hiedir
- Cabal/Stack and other build tools need to learn to manage these.

Tools that make use of .hie files

Haddock hyperlinked-source

- Now with type information!

Haddock hyperlinked-source

- Now with type information!
- Can be extended to support richer code navigation

Demo

HieDb

- Index .hie files to get reference information for your whole source tree!

HieDb

- Index .hie files to get reference information for your whole source tree!
- Extremely fast searching and indexing thanks to SQLite

HieDb

- Index .hie files to get reference information for your whole source tree!
- Extremely fast searching and indexing thanks to SQLite
- Supports many queries on .hie files and suitable for a lightweight IDE interface

Demo

hie-lsp

- An LSP server that uses .hie files and HieDb

hie-lsp

- An LSP server that uses .hie files and HieDb
- Fast, low resource usage

hie-lsp

- An LSP server that uses .hie files and HieDb
- Fast, low resource usage
- Type information on hover

hie-lsp

- An LSP server that uses .hie files and HieDb
- Fast, low resource usage
- Type information on hover
- References

hie-lsp

- An LSP server that uses .hie files and HieDb
- Fast, low resource usage
- Type information on hover
- References
- Go to definition

hie-lsp

- An LSP server that uses .hie files and HieDb
- Fast, low resource usage
- Type information on hover
- References
- Go to definition
- Browse all symbols

hie-lsp

- An LSP server that uses .hie files and HieDb
- Fast, low resource usage
- Type information on hover
- References
- Go to definition
- Browse all symbols
- All of this works with your entire dependency tree

hie-lsp

- An LSP server that uses .hie files and HieDb
- Fast, low resource usage
- Type information on hover
- References
- Go to definition
- Browse all symbols
- All of this works with your entire dependency tree
- Testing ground for features that will make their way into haskell-ide-engine

Demo

hie-lsif

- Generate LSIF files from .hie files

hie-lsif

- Generate LSIF files from .hie files
- To be used for things like Github's online code navigation and review for PRs

hie-lsif

- Generate LSIF files from .hie files
- To be used for things like Github's online code navigation and review for PRs

hie-lsif

- Generate LSIF files from .hie files
- To be used for things like Github's online code navigation and review for PRs

Language Server Index Format

- Language agnostic, project wide cache/database for LSP requests

hie-lsif

- Generate LSIF files from .hie files
- To be used for things like Github's online code navigation and review for PRs

Language Server Index Format

- Language agnostic, project wide cache/database for LSP requests
- Contains cached responses to a subset of LSP requests(hover, definition etc.) for mostly static files like dependencies.

Consuming .hie files

```haskell
readHieFile :: NameCache -> FilePath
            -> IO (HieFileResult, NameCache)
-- ^ Takes and returns a NameCache so it can
-- play nice with existing GHC sessions
generateReferencesMap
  :: HieASTs a
  -> M.Map Identifier [(Span, IdentifierDetails a)]
selectSmallestContaining
  :: Span -> HieAST a -> Maybe (HieAST a)
selectLargestContainedBy
  :: Span -> HieAST a -> Maybe (HieAST a)
```

Future developments

Explaining typeclass evidence

- Typeclass evidence is a bit opaque

Explaining typeclass evidence

- Typeclass evidence is a bit opaque
- Record information about typeclass evidence into .hie files

```
  | EvidenceVarBind
      EvVarSource  -- ^ how did this bind come into being
      Scope        -- ^ scope over which the value is bound
      (Maybe Span) -- ^ span of the binding site
  -- | Usage of evidence variable
  | EvidenceVarUse
data EvVarSource
  = EvPatternBind -- ^ bound by a pattern match
  | EvSigBind -- ^ bound by a type signature
  | EvWrapperBind -- ^ bound by a hswrapper
  | EvImplicitBind -- ^ bound by an implicit variable
  | EvExternalBind -- ^ Bound by some instance
  | EvLetBind [Name] -- ^ A direct let binding
```

```haskell
class C a where
  f :: a -> Char

instance C Char where
  f x = x

instance C a => C [a] where
  f x = 'a'

foo :: C a => a -> Char
foo x = f [x]
--      ^ (31,9)
```

```
At (31,9), found evidence of type: C [a]

Evidence from SrcSpanOneLine "HieQueries.hs" 31 1 14 of type: C [a]
Is bound by a let, depending on:

Evidence of type: forall a. C a => C [a]
bound by an instance at RealSrcSpan SrcSpanOneLine "HieQueries.hs" 27 10 2

Evidence from SrcSpanOneLine "HieQueries.hs" 31 1 14 of type: C a
Is bound by a signature
```

GHCi :set +c

GHCi's `:set +c` command captures types and references information, but:

- it is terribly slow

GHCi :set +c

GHCi's `:set +c` command captures types and references information, but:

- it is terribly slow
- needs to recompile every module

GHCi :set +c

GHCi's `:set +c` command captures types and references information, but:

- it is terribly slow
- needs to recompile every module
- WIP merge request to reimplement all the `:set +c` functionality using `.hie` files

Merge with .hi files

- Tooling integration for free

Merge with .hi files

- Tooling integration for free
- One place to look for everything GHC knows about haskell source

Capturing more type information

- Typechecked AST doesn’t actually contain type information for all nodes

Capturing more type information

- Typechecked AST doesn't actually contain type information for all nodes
- Until now every tool desugared *every single subexpression* in the AST to get its type

Capturing more type information

- Typechecked AST doesn't actually contain type information for all nodes
- Until now every tool desugared *every single subexpression* in the AST to get its type
- Because of this, we only save type information in the `HieAST` when the node has it available or it is a leaf node

Capturing more type information

- Typechecked AST doesn't actually contain type information for all nodes
- Until now every tool desugared *every single subexpression* in the AST to get its type
- Because of this, we only save type information in the `HieAST` when the node has it available or it is a leaf node

Capturing more type information

- Typechecked AST doesn't actually contain type information for all nodes
- Until now every tool desugared *every single subexpression* in the AST to get its type
- Because of this, we only save type information in the `HieAST` when the node has it available or it is a leaf node

We need a function

```
exprType :: HsExpr GhcTc -> Type
```

Links

- hiedb: https://github.com/wz1000/HieDb
- hie-lsp: https://github.com/wz1000/hie-lsp
- hie-lsif: https://github.com/mpickering/hie-lsif
- More information on .hie files: https://gitlab.haskell.org/ghc/ghc/wikis/hie-files