Wrasse Documentation

Jesse Linossier

October 14, 2022

Contents

1 Overview

Wrasse [1] is a web tool that lives on top of the ChameleonIDE [2], that is aimed at making haskell developers able to understand GHC [3] error messages by providing contextual information and suggested fixes. Wrasse relies both on ChameleonIDE as a front-end interface, and the Haskell Error Message Index [4] as a means to provide error codes.

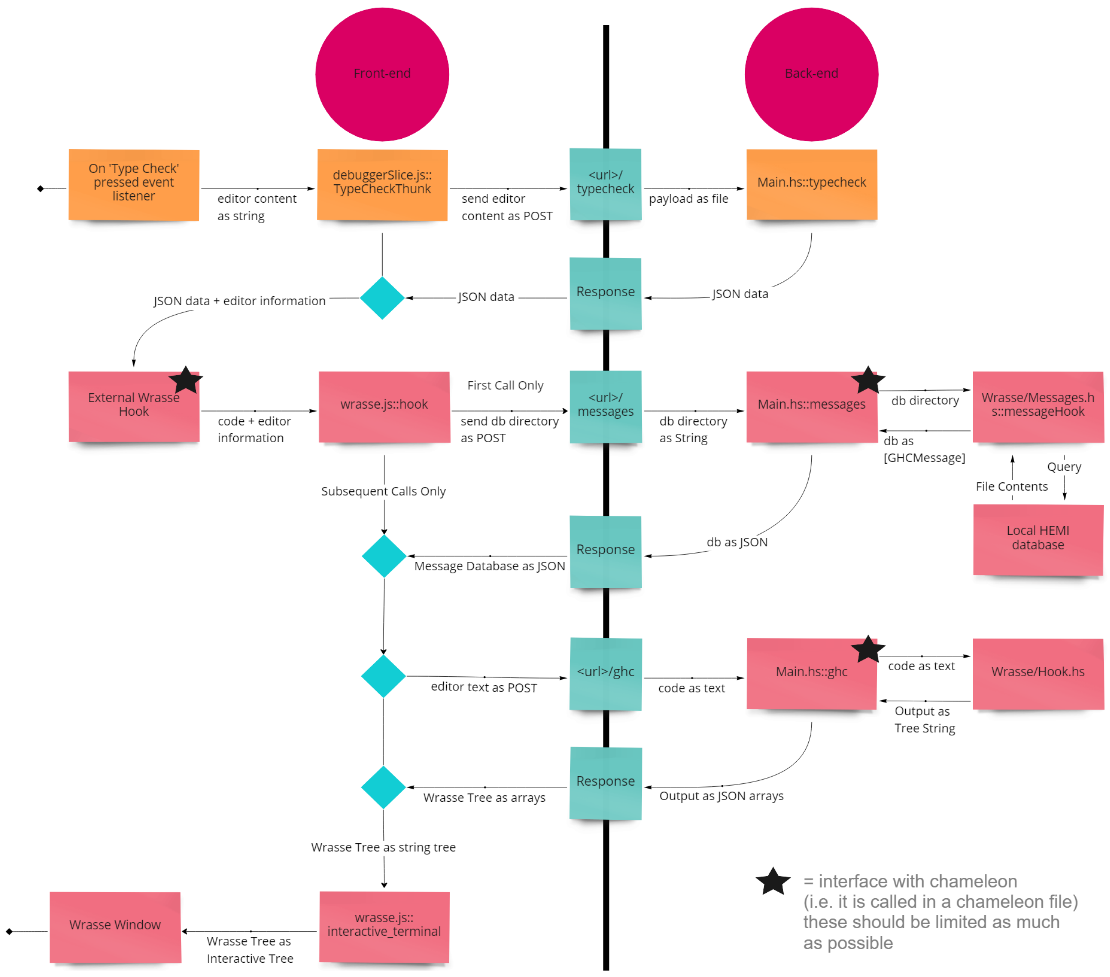


Figure 1: The general control flow of Wrasse. The javascript front-end is on the left side while the haskell back-end is on the right. Blue diamonds represent javascript Promise callbacks or other multistep control flow.

2 Back-end

The back-end is a Haskell Scotty server with endpoints exposed to handle the GHC and HLint analysis on the code.

The Wrasse code is found in 'src/Wrasse', with some minor additions to 'app/-Main.hs' to provide the Wrasse endpoints.

Wrasse has currently only been tested on a Linux server. The server must have the Haskell Error Message Index GHC built and set as system ghc to provide error codes. The application also has to be built with ‘stack install’ before running directly from the installation, as the project is a GHC 8 project while the Haskell Error message Index GHC is GHC 9.

2.1 Interactions with ChameleonIDE

The only interactions with ChameleonIDE in the Back-end, is that the endpoint functions are called in ‘app/Main.hs’ as that is where the Scotty server is started.

2.2 Hook.hs

See:

- wrasse.js
- Tree.hs
- Types.hs

2.2.1 Functions

```haskell
hook :: String -> IO WrasseResult -- entrypoint
-- hook functions for each tool (GHC 9, GHC 8, HLint)
toolHook :: String -> FilePath -> IO [(String, [(String, [String])])]
ghcHook :: String -> FilePath -> IO GHCResult
ghcAltHook :: String -> FilePath -> IO ([String], [String], [String])
hlintHook ::  FilePath -> IO  ([String], [String])
-- utility functions
ghcFile :: String -> FilePath
moduleParser :: Parsec String () String
getModuleName :: String -> String
processHLint :: Either Language.Haskell.HLint.ParseError ModuleEx -> [String]
```

2.2.2 Responsibilities

- Act as an entrypoint for the Wrasse backend
- Run GHC and HLint on provided code and return the results
- Save provided code into ‘generated/Infile.hs’ with module information prepended if necessary
- Interrogate provided code for inferred type, definition location, and all other available information of all available symbols

2.2.3 External Interaction

- Hook::hook is called in 'app/Main.hs::main'
- The data provided by Hook::hook is sent directly to the front-end server as a JSON object
- Hook::ghcHook and Hook::ghcAltHook call the system installed version of GHC
- Tree::multilevel is used to create the interactive tree from the provided GHC error messages

2.3 Instance.hs

See:

- Types.hs

2.3.1 Functions

```haskell
instance Default <class> where
    def :: <class>
```

2.3.2 Responsibilities

- Implement the instances for classes. Currently this only consists of class 'Default'

2.3.3 External Interaction

- The class definitions are defined in Types.hs

2.4 Messages.hs

See:

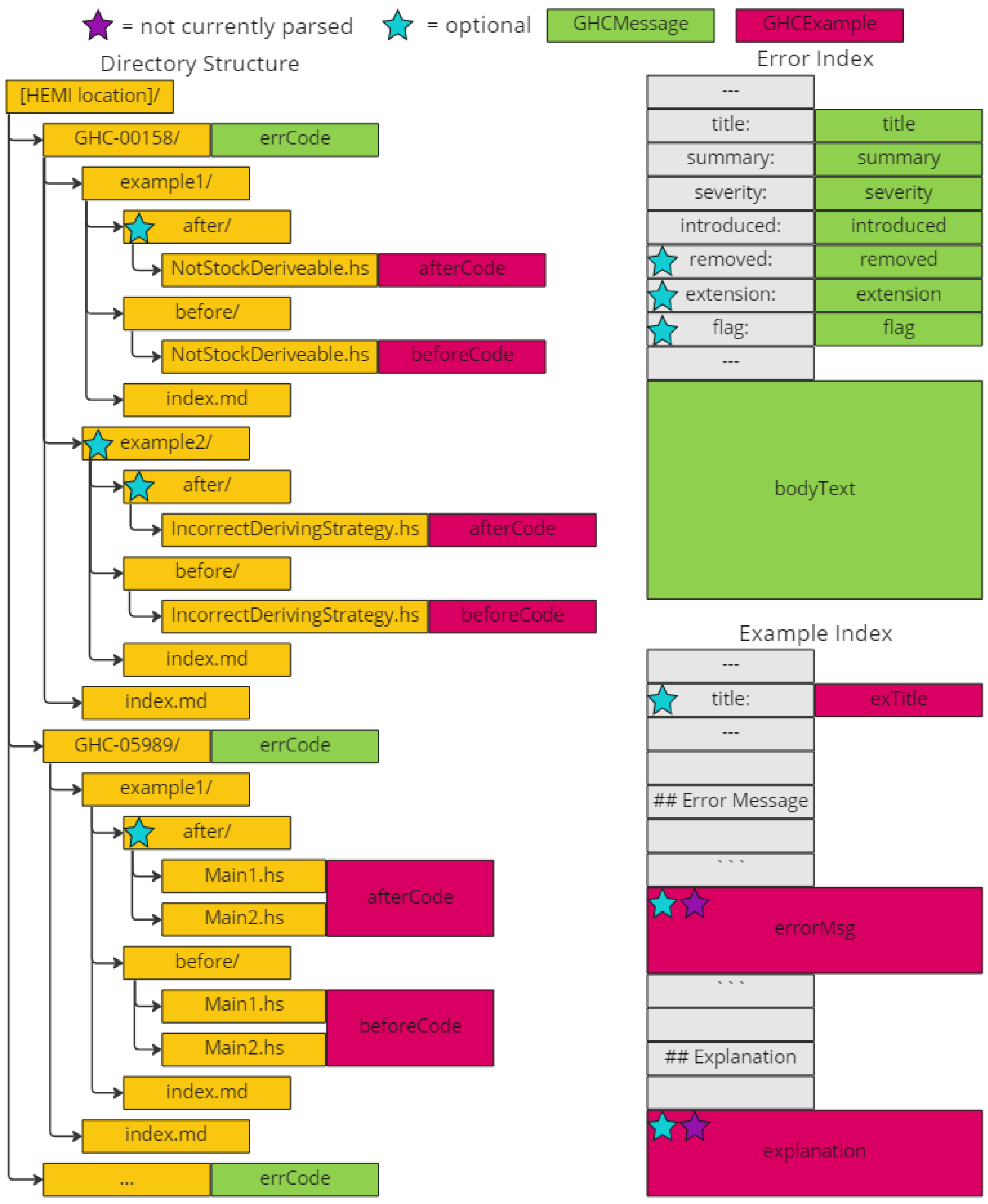
- Types.hs

2.4.1 Functions

```haskell
messageHook :: FilePath -> IO [GHCMessage] -- entrypoint
-- example database parsing
loadMessages :: FilePath -> IO [FilePath]
readMessage :: FilePath -> IO GHCMessage
loadExamples :: FilePath -> IO [GHCExample]
readExample :: FilePath -> IO GHCExample
-- utility functions
```

```haskell
maybeReadFile :: FilePath -> IO String
listDirectoryFull :: FilePath -> IO [FilePath]
```

2.4.2 Responsibilities



= not currently parsed
= optional
GHCMessage
GHCExample
Directory Structure
Error Index
[HEMI location]/
GHC-00158/
errCode
example1/
after/
NotStockDeriveable.hs
afterCode
before/
NotStockDeriveable.hs
beforeCode
index.md
example2/
after/
IncorrectDerivingStrategy.hs
afterCode
before/
IncorrectDerivingStrategy.hs
beforeCode
index.md
index.md
GHC-05989/
errCode
example1/
after/
Main1.hs
Main2.hs
afterCode
before/
Main1.hs
Main2.hs
beforeCode
index.md
index.md
...
errCode
---
title:
title
summary:
summary
severity:
severity
introduced:
introduced
removed:
removed
extension:
extension
flag:
flag
---
bodyText
Example Index
---
title:
exTitle

## Error Message
```
errorMsg
```
## Explanation
explanation

Figure 2: The structure of the Haskell Error Message Index (HEMI) and its files.

- Messages::messageHook acts as an entrypoint for the messages backend of Wrasse

- Read and Parse the local copy of the Haskell Error Message Index
- Provide the HEMI database as a JSON object to the front-end

2.4.3 External Interaction

- Messages::messageHook is called in 'app/Main.hs'
- The data provided by Messages::messageHook is sent directly to the front-end as a JSON object
- Messages relies on a local copy of the HEMI database

2.5 Tree.hs

See:

- Hook.hs
- Types.hs

2.5.1 Functions

```haskell
multiLevel :: [ToolInfo] -> Tree String -- entrypoint
-- tree construction rules
layer :: [String] -> [Tree String]
-- utility functions
prefixLength :: String -> Int
extract :: (a -> Bool) -> [a] -> [(Maybe a, [a])]
```

2.5.2 Responsibilities

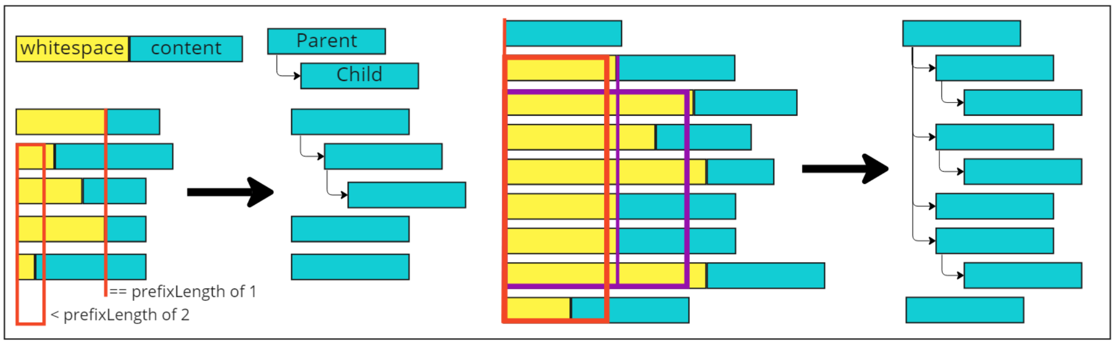


Figure 3: How lines of text are parsed and written into the Tree structure. The process is done recursively, making every node with the same prefix length as the first node or a smaller prefix than the second node at the current root level, and all other nodes are handled recursively with each continuous group treated as one subtree.

- Parse a multi-line text string into a Tree.

2.5.3 External Interaction

- Tree::multiLevel is called by Hook::hook to turn GHC error message strings into GHC error message trees

2.6 Types.hs

See:

- Instance.hs

2.6.1 Functions

```haskell
-- | creates a default / empty instance. Normally used for padding
class Default a where
    def :: a
-- type aliases
type ToolInfo = (String, [(String, [String])])
-- Dataclasses
data WrasseResult -- a result from /ghc endpoint
data GHCResult -- a result from a GHC pass
data GHCExample -- an example from HEMI database
data GHCMessage -- a HEMI database error message
data GHCIInfo -- a result from GHCI about a term
```

2.6.2 Responsibilities

- Define the Types used throughout the Wrasse Back-end
- Define the Classes used throughout the Wrasse Back-end
- Define the Type Aliases used throughout the Wrasse Back-end

2.6.3 External Interaction

- The types defined in here are used throughout the Wrasse Back-end
- Notably the types are given a ToJSON instance to allow them to be sent to the front-end as JSON objects

2.7 Util.hs

See:

- N/A

2.7.1 Functions

This file contains various utility functions for all sorts of tasks. Most of it is self explanatory with a possible exception being the combinator utilities.

```haskell
-- <[.(N times)]> is just like '.', but for N arguments
(<..>) :: (b -> c) -> (a1 -> a2 -> b) -> a1 -> a2 -> c
(<..>) f1 f2 = (f1 .) . f2
-- <[.(N times)]$[.(M times)]> is a map using a function that needs to be
↪  partially curried M times, over a function that needs N arguments
(<..$.>) :: Functor f => (a -> c -> d) -> (b1 -> b2 -> f c) -> a -> b1 -> b2
↪  -> f d
(<..$.>) f1 f2 a b c = f1 a <$> f2 b c
-- <[&& or ||]> applies an argument to two predicate functions, joining their
↪  results with the inner boolean combinator
(<&&>) :: (a -> Bool) -> (a -> Bool) -> a -> Bool
(<&&>) p1 p2 a = p1 a && p2 a
-- Other, more common utility functions omitted for brevity
```

2.7.2 Responsibilities

- Define utility functions for use throughout the Wrasse back-end

2.7.3 External Interaction

- Provides functions for use in the Wrasse back-end

3 Front-end

The front-end is a website using html, css, and js. It uses the fetch API to interact with the back-end.

3.1 Interactions with ChameleonIDE

The front-end does not have much interaction with ChameleonIDE. Logic-wise it only has a hook in debuggerSlice.js to start the wrasse process. View-wise it changes playground.html to incorporate the wrasse html div and split the screen vertically between Chameleon and Wrasse.

3.2 wrasse.js

See:

- Types.hs-Functions WrasseResult
- Types.hs-Functions GHCMessage
- Hook.hs
- Messages.hs

3.2.1 Functions

```
const wrasse = {
    "html" : [key : HTMLElement] as Object,
    "perm" : [key : Disposable] as Object,
    "hook" : ({code, response, editor}) => Promise<void>,
    "setup" : () => void,
    "terminal" : xTerminal,
    "window" : Window,
    "tree" : {
        "content" : string,
        "active" : bool,
        "line" : number,
        "children" : [wrasse.tree],
        "link" : {
            "text" : string,
            "range" : {
                start: { x : integer, y : integer },
                end:   { x : integer, y : integer }
            },
        },
    },
    "data_0" : Wrasse/Types.hs.WrasseResult as Object
    "switch_terminal" : () => void,
    "interactive_terminal" : (wrasse.tree) => void,
    "set_hover_content" : (string) => void,
    "messages" : [Wrasse/Types.hs.GHCMessage as Object],
    "editor" : (code as string) => {payload : any, type : string},
};
```

3.2.2 Responsibilities

- Act as the main entrypoint for the Wrasse front-end
- provide ‘wrasse.hook’ as a way to hook Wrasse into the main codebase with minimal interaction
- Handle all Wrasse-specific functionality

3.2.3 External Interaction

- wrasse.hook is called in ‘static/debuggerSlice.js’
- Calls the ‘/messages’ and ‘/ghc’ endpoints of the back-end server
- Directly modifies the state of the Wrasse Terminal HTML element

3.3 ansiEscapes.js

See:

- ansiEscapes npm package

This file is largely based on the ansiEscapes npm package, with some added functionality that Wrasse required

3.3.1 Functions

```
class Colour {
    static restrict = (float) => integer;
    constructor({r,g,b}) => Colour;
    mul = (float) => Colour;
    blend = (Colour, float) => Colour;
}
// omitted: several functions for ANSI escape sequence construction
```

3.3.2 Responsibilities

- Provide functions to handle ANSI escape sequences
- Provide the Colour class to facilitate easier ANSI escape sequence construction

3.3.3 External Interaction

- Used throughout terminalWindows.js and wrasse.js

3.4 terminalWindows.js

See:

- ansiEscapes.js
- wrasse.js

3.4.1 Functions

```
class Link { // internal class
    static albedo = {click : float, hover : float, unlit : float};
    constructor(Window,
        {x, y} all as integer,
        {enter, leave, click} all as (Link) => void,
        Colour = Colour.Red
    ) => Link;
    setupHighlight = () => void;
    setHighlight = (float) => void;
};
class Window { // exported class
    constructor(xTerminal, integer, integer, integer, integer,
        {movable, scrollable, resizable, softwrap} all as bool
    ) => Window;
    addLink({
        {x, y} all as integer,
        {enter, leave, click} all as (Link) => void,
        Colour = Colour.Red
    }) => this;
    reset = () => void;
    write = (string, (string) => void) => [{pos:number, seq:string}];
    resize = (integer, integer, bool = false) => void;
    expand = (bool = false) => void;
    move = (integer, integer, bool = false) => void;
    // other functions omitted as they are mostly for internal use
}
```

3.4.2 Responsibilities

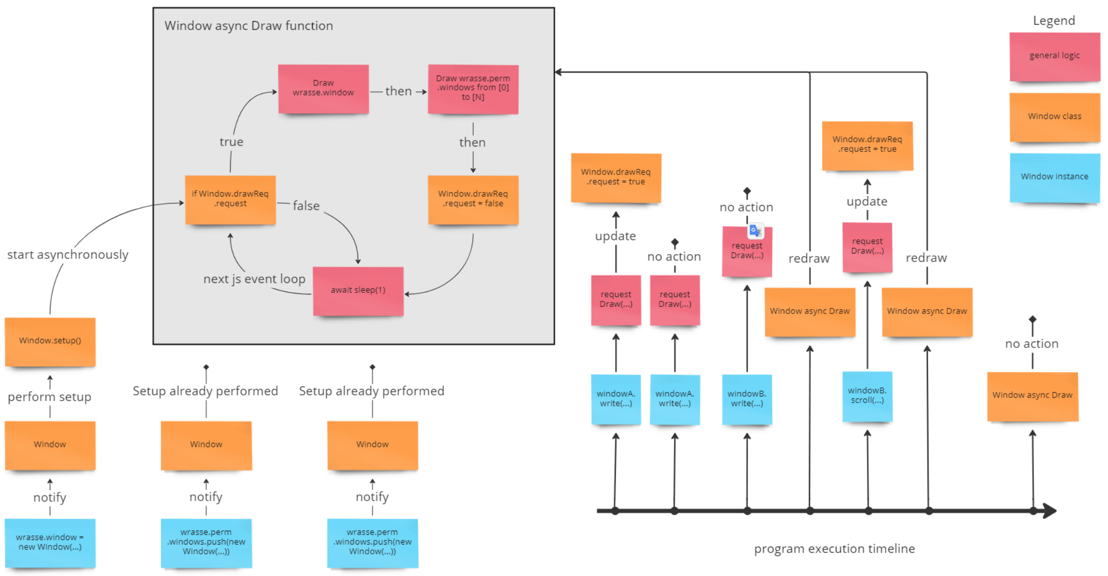


Figure 4: How windows work. The main Window class spawns an async function that periodically checks if any windows have been altered, and if so redraws all windows from the bottommost to topmost. This lowers the amount of redraws performed when many window actions are performed quickly in sequence.

- Provide the Window class as an 'ncurses' style window for use in an 'xTerm' terminal
- Provide movable, resizable, scrollable terminal windows
- Provide effect-ful, clickable 'highlights' or 'links' in a terminal interface

3.4.3 External Interaction

- Used in wrasse.js to separate the Wrasse window into the left error message window and the right contextual information window
- Uses ansiEscapes.js as a standard and source of ANSI escape sequences

3.5 util.js

See:

- N/A

3.5.1 Functions

```
sleep = async (integer) => void;
clamp = (number, number, number) => number;
within = (number, number, number) => bool;
deep_copy = (Object) => Object;
null_func = () => void;
group_n = function *([any], integer) => [any];
start_pattern_gen = function *() => string;
last = ([any]) => any;
debounce = ((...any) => any, integer = 300) => ((...args) => any);
```

3.5.2 Responsibilities

- Define common utility functions for use throughout the Wrasse front-end

3.5.3 External Interaction

- Provides functions for use in the Wrasse front-end

3.6 wrasseGHC.js

See:

- wrasse.js

3.6.1 Functions

```
const wrasseGHC = {
    "regex" : {
        keyword : RegExp,
        symbol : RegExp,
        location : RegExp,
        ambiguous : RegExp,
        error : RegExp,
        codeCommit : RegExp,
    },
    "map" : [key : string] as Object
}
```

3.6.2 Responsibilities

- Provides regular expressions for discovering 'links' / 'highlights' in the Wrasse tree

3.6.3 External Interaction

- The provided regular expressions are used in **wrasse.js** to create **terminalWindows.js-Functions** Links in the appropriate places with the appropriate context, effects, and content

References

[1] J. Linossier, “Wrasse: An extension to chameleon,” 2022. [Online]. Available: https://github.com/BladedTaco/chameleon [Accessed: August, 2022]

[2] T. Fu, “Chameleon: A tool to make solving type errors in haskell simple and fun.” 2022. [Online]. Available: https://chameleon.typecheck.me/ [Accessed: August, 2022]

[3] S. P. Jones, C. Hall, K. Hammond, W. Partain, and P. Wadler, “The glasgow haskell compiler: a technical overview,” in *Proc. UK Joint Framework for Information Technology (JFIT) Technical Conference*, vol. 93. Citeseer, 1993.

[4] Haskell Foundation, “Haskell error message index,” 2022. [Online]. Available: https://github.com/haskellfoundation/error-message-index [Accessed: August, 2022]