ATL Compile

1 Introduction

The function `compile` takes a language file name, `L.atf_lang`, and a list of source file names, `x1.atf_src, ..., xN.atf_src`, and does the following:

1. Parse `L.atf_lang`.
2. Compile the parse `L.atf_lang` into an ATF language specification. This specification, L, specifies the source syntax to be parsed and the target text format to write the compiled source as. The same L will be used for transpiling each of the `xi.atf_src`.
3. For each `xi.atf_src`, do the following.
   (a) Parse `xi.atf_src`.
   (b) Compile `xi.atf_src` into x_i, which is interpreted abstractly as in the framework of the target format specified by L.
   (c) Translate x_i into text format, written into `xi.atf_tgt`, where "`atf_tgt`" is the target file format specified by L.

```
module Compile
( compile
) where

import Debug
```

2 The Compile Function

```
compile :: FilePath -> [FilePath] -> IO ()
compile fp_lang fp_srcs = do
    putStrLn $ "compiling language: " ++ fp_lang
    langcode <- readFile fp_lang
    lang <- compile_language fp_lang
    foldl (>>) (putStr "") $
        map (\fp_src -> do
            putStrLn $ "compiling source: " ++ fp_src
            srccode <- readFile fp_src
            tgtcode <- compile_source lang fp_src
            writeFile (convert_filepath lang fp_src) tgtcode)
        fp_srcs
```

3 Tokens

```
type Token = String
```

4 Compiling Language

```
type SourceCode = String
type LangCode   = String

data Language = Language
    { reserved_tokens :: [Token]
    , convert_filepath :: FilePath -> FilePath
    }

compile_language :: LangCode -> IO Language
compile_language langcode = -- TODO: implementation
    return example_language
```

5 Example Language

```
example_language = Language
    [ "(", ")" ]
    (\fp -> fp ++ ".exp_tgt")
```

6 Compiling Source

```
type TargetCode = String

data Block = Block

compile_source :: Language -> SourceCode -> IO TargetCode
compile_source lang srccode =
    let
        -- separates SourceCode into Tokens, splitting by
        -- the tokens reserved by the Language
        separate :: SourceCode -> [Token]
        separate _ = unimplemented
        -- breaks Token list into a Block tree
        interpret_blocks :: [Token] -> Block
        interpret_blocks _ = unimplemented
        -- arranges the Block tree into the finalized TargetCode
        arrange_blocks :: Block -> SourceCode
        arrange_blocks _ = unimplemented
    in
        return
            $ arrange_blocks $ interpret_blocks $ separate
            $ srccode
```