185.A05 Advanced Functional Programming SS 21
Assignment 5: MINI Project Documentation

Martin Gius und Markus Rinke

June 16, 2021

1. Which project build tool is used for the project? (`cabal` or `stack`)

   Answer: `stack`

2. Which GHC version is used?

   Answer: The Glorious Glasgow Haskell Compilation System, version 8.10.4.

3. How can the program binary be built? How can it be run?

   Answer: The binary can be by built using the command `stack build`. The program can be run using

   ```
   stack run -- PATH ARGS
   ```

   or see

   ```
   stack run -- --help
   ```

   for more options.

4. Which libraries are included as dependencies and which Haskell language extensions are enabled?

   Libraries: containers, parsec, mtl, prettyprinter, generic-random, optparse-applicative, QuickCheck, hspec

   Language extensions: DeriveGeneric

5. Which Framework and libraries are used for writing tests?

   Answer: `QuickCheck` and `hspec`, see above.

6. Which approach is used for the source code parser?

   Answer: We chose to use the library `parsec`. More specifically, we used the monadic parser for strings imported by `Text.Parsec.String`.

7. What is the signature of the main interpreter function? Which MINI extensions had an effect on its type if any?

   Answer: The main interpreter function has the signature `[Integer] -> Program -> IO Value` where `Program` is the AST representation of MINI-programs and `Value` is a type synonym for `Either Integer Procedure`. Implementing the usability of I/O interactions (3.2) required us to lift the right most type from `Value` to `IO Value`. For the implementation of named procedure calls (3.1) we changed the type of `Value` from simple integer values to the above mentioned type.

8. Which MINI language extensions are implemented?

   Answer: We implemented the extension for named procedure calls (3.1), the extension for IO interaction (3.2) and the extension for source code formatting (3.3). We also wrote a small command line interface for our program (4.1) and documented the most important types and functions of the project `haddock`-conformly. Note: you can generate the documentation yourself using `stack haddock MINterpreter --open`.

9. How is the mutating state of the interpreted program implemented?

   Answer: An environment maps strings to either integer values or an AST for procedures. We then definied the following monad using the `StateT` monad transformer from the `mtl` library:

   ```
   type StateIO = StateT Env IO.
   ```

   We first started with the same definition, but environments would simply map strings to integer values and the inner monad in the above definition was the identity monad. We extended the definition of environements to implement the bonus task for procedure calls and we extended the definition of `StateIO` for the bonus task for I/O interaction. This definition allows us to interact with the environment of a program during it's evaluation through the `get` and `put` commands.

10. How is the functionality partitioned into different modules?

    Answer: We partitioned our program into five modules. There are two main modules, `ParseMINI` and `InterpretMINI`. In these two modules the main data type definitions and functionality of parsing a MINI-program and interpreting a parsed program are implemented. In the module `Cli` we implemented the simple command line interface for our program. In the module `Formatting` we implemented syntax formatting functions for MINI-programs. In the module `Correctness` we implemented some MINI-programs and their equivalent Haskell programs.

11. How do you test your program? Which parts are the focus of your tests? Do there exist parts of the code that cannot be tested?

    Answer: We made use of an automatic test-suite for the major components of the project (see question 5), thereby covering the modules `ParseMINI`, `InterpreteMINI` as well as the modules `Correctness` for quickchecking property tests and `PrettyPrinter`. Regarding the former, we focused on testing the major components, e.g. parsing and interpreting of complex expressions, statements and programs, including various programs to test the MINI extension for named procedures, to name a few. Technically, there do not exist parts of code that cannot be tested, but, needless to say, there exist parts of the code which are less preferable being subject to tests than others, e.g. functions with range `StateIO ()`.

12. Are there known issues and limitations of your program?

    Answer: Since we didn't implement the fourth extension, both the parser and the interpreter can give a bit vague error messages.