REPORT ON PETITPURESCRIPT COMPILER PROJECT

PIERRE-GABRIEL BERLUREAU, NGUYEN DOAN DAI

This memo summarises our project for the first part.

1. Compilation and usage

Compilation requires build tool dune of version 5.10.3. The simulator is implemented in OCaml version 5.1.0. The implementation was compiled and tested in Linux Ubuntu 20.04.6 LTS through Window Subsystem for Linux. The source can be found on `https://github.com/enbugging/CompilateursENS`.

The source code is equipped with a `Makefile`, where command

- `make` builds the projects and results in an executable `ppurs.exe` available in the project's directory
- `make tests` builds the executable if necessary, and tests the compiler against the tests regarding syntactic analysis and typing analysis.
- `make clean` removes the build files and the executable.

The compiler has basic interface, of the form
`./ppurs.exe [--parse-only] [--type-only] file.purs`,
with

- `--parse-only`: flag to print only the netlist after scheduling;
- `--type-only`: flag to specify `nr`, the number of cycles to be simulated;
- `file.purs`, the `.purs` file containing the source code in `PetitPurescript`. For further information, consult the documentation provided in `./doc`, title `sujet-v3.pdf`.

2. Report

2.1. **Syntactic analysis.** The compiler supports `PetitPurescript`'s syntax with indentation, and reports lexxing errors such as bad indentation, unexpected characters, unexpected line feed in gap, malformed strings and comments. The error messages are to follow the behaviours of `PureScript` as much as possible, whilst also following the error message format specified in the guideline.

Regarding the parsing error, however, the compile only reports the position of the error, and not the nature.

2.2. **Typing analysis.** Typing proved to be the most difficult and time-consuming part, which is why in order to create an operational compiler (at least on a non-zero part of the requested features) We preferred to move to code production despite the fact that it is not perfectly accomplished. Thus, even if it is only one test that it does not pass, it lacks some features such as checking that there are no two unifyable instances for example. The input point of the typing is the type_file function of the typer.ml file, it takes as input the syntax ast tree and returns a typed ast tree provided with the global environment obtained at the end of typing. In order to have a code as clean and organized as possible statements and expressions are typed by

modules with corresponding names. Overall we tried to follow the subject as much as possible, whereby the functions have for the most part very explicit names. Error messages also comply with the subject. Finally, the environments are managed by lists and lists of associations and most of the questions with the envrionnements uses the functions of the module gestionEnv, in addition to better segmenting the code it should allow us to replace the lists and the lists of associations by Maps and Sets which we did not do for lack of time.

2.3. **Code production.** We succeeded in compiling `log`, `show`, arithmetic expressions and operations with strings, `do` block, conditional, and function calls. We have been working on compiling `let` statements, but that proved to be difficult due to lack of time.

Initially we did not take into account the space for local variables, commonly denoted by `fpcur` when we compiled, so in order to compile function call, we considered the old value of `rsp` as an additional argument, and saved it in the stack. That means we had to increase the shifts from arguments to `rbp`, and unfortunately that also affects the calculation of `fpcurs` for `let` statements, and ultimately the `let` statements are compiled but produce incorrect results.

We have not been able to tackle constructors, i.e., `data`, nor `class` and `instances`, nor `case`. Some ideas were used at some point, such as adding hashes of types of arguments to distinguish different definitions of functions, but ultimately compiling pattern matching, which is necessary for the compilation of function definitions when there are many, requires significantly adding to the typing module.

We also had some difficulties compiling void functions, i.e. `pure`. We have scaffold of the assembly code, but the produced code has segmentation fault, which we suspect to be some stack alignment issues.