Exercise 3 — assigned Thursday 29 February — due Thursday 21 March

New in this exercise: recursive types; reduction semantics; CC machine; nameless terms (DeBruijn notation).

The primary reference for this exercise is Pierce, *Types and Programming Languages*, Chapters 6 and 20, and the Flatt & Felleisen notes.

3.1 Extending the core lambda language

Language features you should implement: recursive types in the iso-recursive formalism, discussed in Chapter 20 of TAPL. Update the concrete syntax description according to the examples given. Extend all previous code to handle the new syntactic forms.

3.2 DeBruijn notation

Here we re-implement our small-step operational semantics using the nameless representation of source terms, also known as DeBruijn notation. This is discussed at length in Chapter 6 of TAPL. Feel free to consult Pierce's OCaml code, or even the TAPL in Haskell code, which you can find online.

Implement the functions *shift* and *subst* in module *DeBruijnWithIntegerLabelsAndTags*. Implement the function *eval1* in module *StructuralOperationalSemantics_CBV_forDeBruijn*.

The implementation should cover the complete core language.

3.3 Main program

Write a main program which will:

1. read the program text from a file into a string,
2. invoke the parser to produce an abstract syntax tree for the program,
3. print the free variables of the program,
4. in case the program is closed (has no free variables), type-check the program,
5. in case the program has a type, evaluate it with respect to call-by-value structural operational semantics (yielding a term),
6. evaluate the program using standard reduction,

7. evaluate the program using the CC machine,

8. convert the program to DeBruijn notation and then evaluate the converted program with respect to call-by-value structural operational semantics for DeBruijn terms.

3.4 Provided code

`AbstractSyntax.lhs` (updated)
`EvaluationContext.lhs`
`ReductionSemantics.lhs`
`CCMachine.lhs`
`DeBruijnWithIntegerLabelsAndTags.lhs`
`StructuralOperationalSemantics_CBV_forDeBruijn.lhs`
`test41.corelambda`
`test43.corelambda`
`test55.corelambda`
`test74.corelambda`
`test77.corelambda`
`test78.corelambda`
`test79.corelambda`
`test82.corelambda`
`test99.corelambda`
`test100.corelambda`
`test101.corelambda`
`test105.corelambda`
`test110.corelambda`
`test114.corelambda`
`test115.corelambda`
`test133.corelambda`
`test134.corelambda`
`test141.corelambda`
`test142.corelambda`
`test143.corelambda`
`test145.corelambda`

3.5 What to submit

Prepare a project report (PDF, named `report.pdf`). If you want, you can prepare the code and the report together in literate Haskell (like this assignment).

Your project report should include your commentary on the code structure, interesting design decisions, etc. As an appendix, include the complete program listing as well as example executions.

In the project report, **please remember to describe your team composition and how the team collaborated on the task, and comment on the level of effort needed for the project**.

Use Canvas to submit a tar file with all the code and the report.

3.6 Oral presentations

Teams will give oral presentations in class on Tuesday 19 March. They can be informal, and they can include code walkthroughs and live demonstrations. All team members should report their contributions. If you have used GHC language extensions, be ready to teach us how they work. Plan on 15–20 minutes per team.