Types & Programming Languages
Exercises 3

These exercises are based on the typechecker for the Simple Expression Language: first compiling and testing it, then modifying it. The instructions below are oriented towards Linux; if you are working with Windows then you will need to make appropriate modifications.

1. Get the SableCC tool (file `SableCC.zip`) from the course web page. When unzipped, the directory `SableCC` contains files `sablecc`, `sablecc.bat` and `sablecc.jar`.

2. Get the SEL implementation (file `SEL.zip`) from the course web page and unzip it.

3. Move the directory `SEL` to a convenient place in your filespace.

4. Move `sablecc.jar` and `sablecc` (or `sablecc.bat` for Windows) to a convenient place in your filespace.

5. Edit your copy of `sablecc` so that it contains the correct path for `sablecc.jar`

6. In (your copy of) the directory `SEL`, execute `sablecc sel.grm` (you might need to specify a full path for `sablecc` or else put it in a directory which is included in your `PATH`).

7. Still in `SEL`, compile the typechecker with `javac sel/Main.java` (alternatively you might want to set up a makefile or convert the whole thing into a project in your favourite development environment).

8. Test the typechecker with `check test.sel` and try out some other examples of correct and incorrect input.

9. Extend the Simple Expression Language by adding a new operator, for example multiplication. You will need to:

   (a) Modify `sel.grm` to define the syntax of your operator.

   (b) Define new classes in `error` to represent any new type errors associated with your operator.

   (c) Modify `checker/Checker.java` to include an `out` method for the syntax tree class corresponding to your operator.

10. If you feel ambitious, add a new type (for example `float`) with suitable syntax, literal values, operators and typing rules. You will need to add definitions to `types`. The simplest way to add a new numeric type such as `float` is to keep it entirely separate from `int` except for explicit conversion operators if you want to include them. In this scenario, the literal value `1` would *only* be an `int` and to get the value 1 as a `float` you would write `1.0`. Floating point addition would be a separate operation with a different name such as `.+` or similar.

    For a more complicated exercise you could replicate the way that numeric types can be mixed in languages such as Java, with implicit conversion from `int` to `float` when required. Doing this systematically requires a treatment of subtyping, which we will cover later in the course.