Programming Languages
A Journey into Abstraction and Composition
1st Exercise: Scala and Interpreters for Simple Arithmetic Expressions

Programming Group
guido.salvaneschi@uzh.ch

University of Zurich – June 28th, 2022

Scala Warm-Up

The following tasks are mainly about learning Scala.

Info: We use `sbt` as a build tool for our tasks. See `http://www.scala-sbt.org/` for installation instructions. You can run the command `sbt` to enter an interactive console where you can execute build commands. You can use `test` to run the included tests and the ones you might have added for yourself. You can also use a development environment of your choice, IntelliJ can import the sbt project directly by opening the `build.sbt` file.

Task 1: Factorial Function

Write one factorial function in each of the following styles:

a) for loop with accumulator variable

b) recursion (no mutation!)

c) folding (no mutation!)

Use the template provided in `scala/FactorialFunction.scala`.

In the following exercises, you shall get familiar with using the language and its implementation of pattern matching, lists, and recursion.

Only use list construction (`List(...)`, `::`, `Nil`), pattern matching (`case head :: tail =>...`), data extraction via `.head` and `.tail`, and `List.isEmpty` which tests whether a list is empty in this exercise. Your code should be minimal and concise. Avoid if-else constructs in favor of pattern matching.

Warning: Do *not* use imperative, non-functional style for the following tasks (i.e., loops), or any non-trivial, built-in members of `List`, like `List.reverse` or other methods for appending lists such as `:+`, `++`, `:::`, etc. If you want to use them, you have to provide your own implementation with the restrictions stated above.

Task 2: Peano Number Addition

Implement the addition of two natural numbers as a recursive function only in terms of successor (succ) and predecessor (pred). Use the template provided in `scala/PeanoNumberAddition.scala`.

Task 3: Flatten

Implement the function `flatten`. Use the template provided in `scala/Flatten.scala`.

The function `flatten` converts a list of lists `xss` to a list `xs`, including all elements of the lists in `xss`. For example, `List(List(1), List(2, 3), List(5))` is converted to `List(1, 2, 3, 5)`.

Task 4: Reverse

Implement the function `reverse`. Use the template provided in `scala/Reverse.scala`.

The function `reverse` reverses the order of a list. For example, `List(1, 3, 2, 4)` is converted to `List(4, 2, 3, 1)`.

Boolean Expressions

Task 5: Interpreter

Implement an interpreter for a small boolean expression language (BE) that supports boolean literals, conjunction, disjunction, and negation. Use pattern matching and case classes. Use the template provided in `boolean_expressions/Interpreter.scala`.

Task 6: Preprocessor

Implement a preprocessor to add more features to the boolean expression language without modifying the interpreter. The preprocessor shall implement implications and biimplications by transforming a given BE tree. The resulting tree must be free of any `Imp` or `BiImp` expressions. Use the template provided in `boolean_expressions/Preprocessor.scala`.

Info: Have a look at the test cases for the expected definition of implication and biimplication.