Objectives

- (Practice) To gain experience using a Domain Specific Language
- (Theory) To understand the design and use of combinators within a DSL
- (Theory) To appreciate the role of equational reasoning in FP

Resources

You should refer to the following resources accessible via Blackboard:

- Topic 8 Lecture videos 8A, 8B, and 8C.
- **Topic 8 folder** (zipped) – includes the notes and slides for this topic.
- External website for Scala doc and other **Learning Resources** (see the folder on Blackboard).

Introduction

Domain Specific Languages (DSLs) are programming languages that have been written for a specific application area. As such these languages are not normally suited to general purpose programming but, rather, they are related to a single activity. The purpose of them is to provide the programmer with a language in which to express the concepts within the domain clearly and in a way that is consistent with the way the concepts are used within that domain. In this example, a library of (ASCII) Pictures has been presented. Higher order functions have been used to create picture combinators that allow various layouts of picture on the page to be constructed.

DSLs can be external or internal. External DSLs exist as standalone languages that have to be parsed to create objects that can be manipulated within another programming language. Such languages have their own parsers which generate the internal data structures for specified target programming languages. Internal DSLs are a language within a language. The programming features of the host language are used to construct the DSL.

The Picture library is an example of an internal DSL. There is a drawback to internal DSLs in that they are necessarily constrained by the syntax and semantics of their host. However, the benefit of an internal DSL is that the domain language itself benefits from total interoperability with its host. Many programming language features in Scala facilitate the construction of internal DSLs. Such features include higher order functions, infix method syntax, case classes and objects, implicit functions, and immutable data structures.

A picture is a rectangular block of (ASCII) characters. For example, the following picture has four rows and 12 columns:

```
This is a
picture with
depth 4 and
width 12
```

The smallest picture is the empty picture (no rows, no columns). Operations have been provided for combining pictures – e.g. one above another, or one beside another. If the pictures are nonconformant (i.e. different widths when stacked, or different depths when placed side by side) then the one with the smaller dimension needs to be padded so that the result is a rectangular block with no jagged edges. Furthermore, combinators are provided for stacking and spreading lists of pictures; for framing pictures and putting borders around pictures. The result is a very powerful library for laying out pictures within a terminal or for displaying within a text editor etc.

A feature of the picture library is that its components obey certain mathematical laws. In fact, these laws derive from those that pertain to the List data structure upon which it is based. Although you do not need to be able to derive proofs of such laws yourself, the main point to appreciate is that these laws can be stated and proved. The benefit of doing this is two-fold: (1) certain properties facilitate the transformation of picture expressions into other, equivalent picture expressions; and (2) certain guarantees can be made about the behaviour of the library components, including the outcome of certain combinations of operators. This is facilitated by the underlying value semantics – the lack of side effects and the use of immutable data structures.

Activities

3.1 Watch the videos

Watch the videos 8A – 8C. The accompanying slides can be found in the topic-8-folder.

3.2 Install and run the PictureDemo programs

You will need to add a new package **picture** to the **lib** folder. Then install in that package the files **Picture.scala** and **Timetable.scala**.

Open your Scala IDE and within the **src** folder (create and) move to the **demo.picture** package

Copy the Scala files **PictureDemo1.scala** and **PictureDemoJ.java** into the **list** package. The exercises are embedded within these programs. Follow the instructions in the comments.