Chapter 3. The Programming Language

3.1. Introduction

DSSSL specifies a programming language based on Scheme [Dybvig1996], a standard functional programming language. The two languages differ in some ways but these differences are not relevant to this book. DSSSL is a small and exceptionally clear language, designed to have just a few regular constructions that can be composed easily. It follows a formal mathematical model, the lambda calculus.

Summarizing in a few words, it can be said that DSSSL is “a statically scoped expression language with block structure where types are associated to values. Evaluation is strict and parameter passing is by value. Implementations are required to be properly tail-recursive”. The rest of this chapter will explain the above phrase point by point, while the necessary notions are given to use the language.

3.2. Expressions

An expression language is a language that only manages expressions. An expression is a construct that returns a value. Everything that one can write using the DSSSL programming language is an expression. There are only five kinds of expression: literals, variables, lambda, procedure call and conditional. As has already been pointed out, it is a small language with few constructions.
3.2.1. Literals

Literals are self-evaluating expressions. If an expression is that which returns a value, a literal directly represents a value. It is the most basic type of expression.

Example 3-1. Literal

5 is a literal that self-evaluates itself to the numeric quantity five.

3.2.2. Lambda Expressions

The underlying formal mathematical model of the expression language is the lambda calculus, as was explained in the introduction. We are not going to enter into a detailed explanation of the lambda calculus model, as this is not the aim of the book. However, a quick introduction to a few concepts will be helpful before entering into the use that DSSSL makes of the lambda calculus.

In the lambda calculus, everything is a lambda expression. A lambda expression is a way to codify what is usually called a function or procedure. Example 3-2 shows the aspect a lambda expression has.

Example 3-2. A Lambda Expression

\[ \lambda x. y \]

where \( x \) is the parameter of the expression, and \( y \) is the calculation performed.

Thus, for example, in lambda calculus, a constant numeric value would be represented:

Example 3-3. Constant Lambda Expression

\[ \lambda x. 5 \]

which is interpreted as “given \( x \), return the value 5 as the result”.

\( \text{lambda} \)

\[
\text{lambda (required-arguments}
\#!optional optional-arguments
\#!rest rest-argument
\#!key keywords)
\text{expression})
\]

\( \text{lambda} \) is the DSSSL codification of the lambda expression.

- \( \text{required-arguments} \) are values necessary for the result of the lambda expression.
- \( \text{optional-arguments} \) are values that can be used for the result, but are not necessary for the calculations. Each optional argument may be a variable name or a list of two elements, a variable name and an initializer.