COMP332—Programming Languages
Assignment One Report

Christian Nassif-Haynes – 42510023

2 September 2013

1 Introduction

This document describes the implementation and testing of assignment one in COMP332. The aim of the assignment was write a program structure tree (PST) builder for the Func332 functional programming language by using the Kiama library.[1] A skeleton project and context-free grammar (CFG) was provided for which the PST builder and accompanying tests had to be implemented.

The following sections discuss the design of the tree builder and the way in which the project was tested.

2 The Lexical Analyser

The implementation and testing of the PST builder was reasonably straightforward as a CFG and language processing library were given.

2.1 Design

The CFG which was provides is as follows:

```
program : (exp ";")+.

exp  : "let" idndef "=" exp "in" exp
     | exp0.

exp0 : exp exp
     | exp "=" exp
     | exp "<" exp
     | exp "+" exp
     | exp "-" exp
     | exp "*" exp
     | exp "/" exp
     | "true"
     | "false"
     | idnuse
     | integer
     | "if" exp "then" exp "else" exp
     | "(" idndef ":" tipe ")" "=>" exp
     | "(" exp ")".

tipe : "int"
     | "bool"
     | tipe "->" tipe
     | "(" tipe ")".
```

This CFG is ambiguous and changes were required to allow for well-defined operator precedence and associativity. Specifically, it was required that the following constructs have precedence in the order listed:

1. Let, if and function.
2. Equal and less than.
3. Application.

[1] `https://code.google.com/p/kiama/`

4. Addition and subtraction.

5. Multiplication and division.

6. All other kinds of expressions.

Additionally, it was required that the application binary operator and the arrow function type were right associative and that all other binary operators were left associative. Subsequently, the CFG was modified to account for these desiderata as follows:

```
program : (exp ";")+.

exp   : "let" idndef "=" exp "in" exp
      | "if" exp "then" exp "else" exp
      | "(" idndef ":" tipe ")" "=>" exp
      | exp0.

exp0  : exp0 "=" exp1
      | exp0 "<" exp1
      | exp1.

exp1  : exp2 exp1
      | exp2.

exp2  : exp2 "+" exp3
      | exp2 "-" exp3
      | exp3.

exp3  : exp3 "*" exp4
      | exp3 "/" exp4
      | exp4.

exp4  : "true"
      | "false"
      | idnuse
      | integer
      | "(" exp ")".

tipe  : tipe0 "->" tipe
      | tipe0.

tipe0 : "int"
      | "bool"
      | "(" tipe ")".
```

Operator precedence has been specified in the modified CFG by forcing high precedence operators further down the parse tree. For example, as multiplication takes precedence over addition, `exp3` (the production dealing with multiplication) is only mentioned in `exp2` (which deals with addition).

Left and right associative operators were specified by using left and right-recursive productions, respectively. The `exp0` and `tipe` are two such recursive productions.

After the CFG was rewritten, what was left was to implement it. Implementation entailed further rewriting the CFG in Scala, augmenting productions with their corresponding PST node where necessary. Shown below for illustration is the `exp0` production rewritten in Scala, augmented to infer the `EqualExp` and `LessExp` nodes:

```
lazy val exp0 : PackratParser[Expression] =
    (exp0 <~ "=") ~ exp1 ^^ { case a ~ b => EqualExp (a, b) } |
    (exp0 <~ "<") ~ exp1 ^^ { case a ~ b => LessExp  (a, b) } |
    exp1
```

2.2 Testing

The included tests covered many cases. The additional tests focused on operator associativity and precedence as rewriting the given CFG to specify them seemed to be the crux of this assignment. As well as this, some such tests were omitted from the included test set.

Selected additional tests are

- '* is left associative' and '/ is left associative' and;
- '- has lower precedence than / (to right)' and '+ has lower precedence than / (to left)'.

Additionally, it was checked that a rudimentary program could be parsed correctly as well as that parsing a chain of functions (as is common in functional languages) succeeded.