COMP332—Programming Languages
Assignment Two Report

Christian Nassif-Haynes – 42510023

8 October 2013

1 Introduction

This document describes the implementation and testing of assignment two in COMP332. The aim of the assignment was to extend the given program to implement semantic analysis (including name analysis) for the Func332 functional programming language by using the Kiama library.[1]

The following sections discuss the design of the analyser conceptually, it's implementation and the way in which the project was tested.

2 The Semantic Analyser

2.1 Design

2.1.1 Name Analysis

The bulk of name analysis can be conceptualised by annotating each node in a program's syntax tree with a list of variables which are visible at that node. As, in Func332, the variables which are visible at a node depend on those visible at the parent, name analysis can be carried out recursively.

A more complex feature, scoping rules, can be though of as partitioning a syntax tree into a subtree above a node and a subtree including and below it. The outer scope[2] contains both subtrees, however the inner scope contains only the lower one.

In Func332, function and let expressions have their own scope. In other words, the program's syntax tree is partitioned at these nodes. As well, variables in the inner scope hide those in the outer scope.

2.1.2 Type Analysis

Func332 is statically and strongly typed, making type analysis comparatively straightforward; the type of an expression depends on its constituent expressions. This knowledge can be used to infer the type of an expression recursively. Additionally, the "expected type" of an expression (the type suggested by its context) can be inferred to aid Func332 programmers' debugging.

2.2 Implementation

2.2.1 Name Analysis

Name analysis was primarily facilitated by a) using the `define` function to make Kiama "aware" of variables and; b) using the `enter` function to give variables the correct scope and implement variable hiding. It was not required to use `leave` because in Func332 variables can only be defined in a nested scope, and only one variable can be defined at each scope level.

2.2.2 Type Analysis

Type analysis is aided by Kiama's `lookup` function, which finds the `entity` referred to by a variable. From there, `tipe` is applied recursively to determine each expression's data type.

The `tipe` and `exptipe` functions are used by `check` to ensure that the type of each expression matches up with the context of that expression.

2.3 Testing

A fair number of type and name tests were devised. These included tests covering multiple cases where:

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

[2] Each of these scopes is referred to as an *environment* in Kiama.

- The operators $+$, $-$, $\div$ and $\times$ were compatible with their operands.
- If statements had boolean-valued conditions, and if- and else- bodies of matching type.
- Let expressions had the correct type.
- Functions were typed correctly.[3]
- Variables could not be used unless they were declared.
- Variables had the correct visibility based on their scope.

[3]Actually, this case wasn't well covered. Indeed, type analysis is not correctly implemented for function definitions whose argument types are a functions.