CS644 Project Report: A4

Submitted in partial fulfillment of the requirements for the course

Submitted by

Name	ID
Siwei Yang	s8yang
Justin Vanderheide	jtvander
Jian Li	j493li

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada – N2L 3G1

Winter 2015

1 General Strategy

This section describes the meta strategy we applied in developing solutions to each of the deliveribles. In principle, we are using the attribute grammar approach to finish all the tasks: name resolution, type linking and static analysis.

However, it takes more than merely coding the attribute rules to solve the problem. Auxiliary information might be needed at every node to make the right decision. And the information flows both ways which is hard to support is a pure functional language like Haskell. Besides, we would like to achieve a nice separation between the supporting data structures and the grammar structures.

Therefore, we modelled our solution as three parts: a transformed data structure captures the essense of the AST as well as supporting bi-directional information flow, a newly build data structure that answers queries from the attribute rules and the implementations of the attribute grammars.

1.1 Engineering a Pure Functional Double-linked AST

1.2 Consolidating Type Hierarchy into a Database

1.3 Coding Attribute Rules

2 Name Resolution and Type Linking

2.1 Division of Responsibility

The scanner is implemented in the file *Scanner.hs*.

Each token type has an associated function that implements a recognizer for it. For example *scanChar* recognizes char literals, and *scanBool* recognizes boolean literals. Each of these functions consumes a string containing the unscanned portion of the file, and returns a Maybe representing the result.

If the recognizer did not find its token type, it returns Nothing. For example if the input is "while (..." then *scanDecimalInteger* will return Nothing since the beginning of the input is not a valid integer literal token. If the recognizer does find its token type then it returns a pair where the first element is the token type, and the second element is the lexeme of the token. For example in the input "while (..." the *scanKeyword* function will return ("KEYWORD", "while"). Each of these functions is run in parallel, and the longest lexeme is taken.

```
*Scanner> scanDecimalInteger "while ("
Nothing
*Scanner> scanKeyword "while ("
Just ("KEYWORD","while")
```