CSC488H1S - Assignment 2: Design Document.

Group 2
g0dalaln - Natasha Dalal
g0getter - Tony (Hao) Cheng
g0faizan - Faizan Rashid
g0alimuh - Showzeb Ali
g3ksingh - Karandeep Basi

February 5, 2014

When designing the grammar, I started by just simply following the language specification, correcting obvious problems as I went. The first real variation from the given specification was splitting up grammar rules that could lead to multiple parse trees. For instance, $Statement \rightarrow Statement\ Statement$. In order to disambiguate cases like these, I split up rules with two recursive references into another rule. Therefore with the example of Statement, I created $StatementList \rightarrow Statement \mid Statement\ StatementList$. Then wherever there were references to Statement in the reference language rules, I replaced them with $StatementList$.

Another deviation from the reference grammar involved replacing all references to variablenames, parameternames, functionnames, prcedurenames, etc... with $IDENT$.

The part of the grammar I found most challenging was the expressions. Specifically, expressions needed to be defined in a way to allow for operator precedence. The way that I handled this was by starting at a base form that I called $baseExpression$. This base form defined some of the most basic forms of expressions: Terminals, argument lists, expressions in parantheses etc.... From there, I built up at every step starting with the operator of highest precedence, including in its rules a rule to generate the previous type and then a rule using the previous type with the current operator. For example: the unary minus has highest precedence and so I did $UnaryExpression \rightarrow BaseExpression \mid MINUS\ BaseExpression$. In this way, I built up all the way to OR which had the lowest precedence. Therefore, OrExpression was inclusive of all other generatable expressions, as well as such expressions with an OR symbol between them. I then used $ORExpressions$ as my $finalExpression$ which is used wherever the reference langugae makes reference to expression. This $finalExpression$ was also used in $baseExpressions$ within parentheses.

In order to maintain associativity from Left to Right for multiplication, addition etc... My rules for those have a recursive part as the left most non-terminal on the RHS of the rule. This ensures the left to right associativity.