Design Decisions and Evaluation of Parser

Maryam Dabaghchian-u1078006

February 1, 2018

1 Grammar

$Exp \rightarrow LtLevel$
$Exp \rightarrow Exp$ "&&" $LtLevel$
$LtLevel \rightarrow PlusLevel$
$LtLevel \rightarrow LtLevel$ " < " $PlusLevel$
$PlusLevel \rightarrow MultLevel$
$PlusLevel \rightarrow PlusLevel$ (" + "|" − ") $MultLevel$
$MultLevel \rightarrow DotLevel$
$MultLevel \rightarrow MultLevel$ " ∗ " $DotLevel$
$DotLevel \rightarrow PrimaryExp$ ("[" Exp "]" | "." "*length*" | "." Id "(" (Exp ("," Exp)*)? ")")*
$DotLevel \rightarrow$ "!" $DotLevel$
$PrimaryExp \rightarrow$ "*true*" | "*false*" | "*this*" | "*new*" "*int*" "[" Exp "]" | "*new*" Id "(" ")" | $< NUM >$ | "(" Exp ")"

1.1 Eliminate Left Recursion

$Goal \rightarrow MainClass$ ($ClassDecl$)* $< EOF >$
$MainClass \rightarrow$ "*class*" Id "{" "*public*" "*static*" "*void*" "*main*" "(" "*String*" "[" "]" Id ")" "{" $Stmt$ "}" "}"
$ClassDecl \rightarrow$ "*class*" Id ("*extends*" Id)? "{" ($VarDecl$)* ($MethodDecl$)* "}"
$VarDecl \rightarrow Type\ Id$ ";"
$MethodDecl \rightarrow$ "*public*" $Type\ Id$ "(" ($Type\ Id$ ("," $Type\ Id$)*)? ")" "{" ($VarDecl$)* ($Stmt$)* "*return*" Exp ";" "}"
− −
$Type \rightarrow$ "*int*" "[" "]" | "*boolean*" | "*int*" | Id
− −
$Stmt \rightarrow$ "{" ($Stmt$)* "}" |
"*if*" "(" Exp ")" $Stmt$ "*else*" $Stmt$ |
"*while*" "(" Exp ")" $Stmt$ |
"*System.out.println*" "(" Exp ")" ";" |
Id " = " Exp ";" |
Id "[" Exp "]" " = " Exp ";" |
− −
$Exp \rightarrow LtLevel\ Exp'$
$Exp' \rightarrow$ "&&" $LtLevel\ Exp'$
$Exp' \rightarrow$

LtLevel → PlusLevel LtLevel′
LtLevel′ → " < " PlusLevel LtLevel′
LtLevel′ →
PlusLevel → MultLevel PlusLevel′
PlusLevel′ → (" + "|" − ") MultLevel PlusLevel′
PlusLevel′ →
MultLevel → DotLevel MultLevel′
MultLevel′ → " ∗ " DotLevel MultLevel′
MultLevel′ →
DotLevel → PrimaryExp ("[" Exp "]" | "." "length" | "." Id "(" (Exp ("," Exp)*)? ")")*
DotLevel → "!" DotLevel
PrimaryExp → "true" | "false" | "this" | "new" "int" "[" Exp "]" | "new" Id "(" ")" | < NUM > | < ID > | "(" Exp ")"

1.2 Left Factoring

Goal → "class" Id MainClass ("class" Id RegClass)* < EOF >
MainClass → "{" "public" "static" "void" "main" "(" "String" "[" "]" Id ")" "{" Stmt "}" "}"
RegClass → ("extends" Id)? "{" (VarDecl)* (MethodDecl)* "}"
VarDecl → Type Id ";"
MethodDecl → "public" Type Id "(" (Type Id ("," Type Id)*)? ")" "{" (VarDecl)* (Stmt)* "return" Exp ";" "}"
− −
Type → "int" "[" "]" | "boolean" | "int" | Id
− −
Stmt → "{" (Stmt)* "}" |
"if" "(" Exp ")" Stmt ElseStmt |
"while" "(" Exp ")" Stmt |
"System.out.println" "(" Exp ")" ";" |
Id AssignStmt
ElseStmt → ("else" Stmt)?
AssignStmt → " = " Exp ";" |
"[" Exp "]" " = " Exp ";"
− −
Exp → LtLevel Exp′
Exp′ → "&&" LtLevel Exp′
Exp′ →
LtLevel → PlusLevel LtLevel′
LtLevel′ → " < " PlusLevel LtLevel′
LtLevel′ →
PlusLevel → MultLevel PlusLevel′
PlusLevel′ → (" + "|" − ") MultLevel PlusLevel′
PlusLevel′ →
MultLevel → DotLevel MultLevel′
MultLevel′ → " ∗ " DotLevel MultLevel′
MultLevel′ →
DotLevel → PrimaryExp ("[" Exp "]" | "." "length" | "." Id "(" (Exp ("," Exp)*)? ")")*

$DotLevel \rightarrow$ "!" $DotLevel$
$PrimaryExp \rightarrow$ "$true$" | "$false$" | "$this$" | "new" $NewExp$ | $< NUM >$ | $< ID >$
| "(" Exp ")"
$NewExp \rightarrow$ "int" "[" Exp "]" | Id "(" ")"

2 Design Decisions