Documentation Report
Assignment 3: Abstract Syntax Tree Generation

COMP 442-NN

WINTER 2023

Jeremy Piperni
40177789

1. Attribute Grammar

The following is the grammar used for the parser augmented with syntax-directed translation. Semantic actions appear in this grammar and start with an # (Ex: #A)

START -> #B REPTSTART0 #AK .

ADDOP -> plus #P .

ADDOP -> minus #P .

ADDOP -> or #P .

APARAMS -> EXPR #AA #B REPTAPARAMS1 #AB #AC .

APARAMS -> #AC2 .

APARAMSTAIL -> comma EXPR #AA .

ARITHEXPR -> TERM RIGHTRECARITHEXPR .

ARRAYSIZE -> lsqbr Flsqbr .

Flsqbr -> intlit #A rsqbr .

Flsqbr -> rsqbr #BA .

ASSIGNOP -> equal .

ASSIGNSTAT -> VARIABLE ASSIGNOP EXPR .

CLASSDECL -> class id #A #B OPTCLASSDECL2 #M lcurbr #B REPTCLASSDECL4 #K rcurbr semi #N .

CLASSDECLORFUNCDEF -> CLASSDECL .

CLASSDECLORFUNCDEF -> FUNCDEF .

EXPR -> FACTOR RIGHTRECTERM RIGHTRECARITHEXPR EXPR2 .

EXPR2 -> RELOP ARITHEXPR #Z .

EXPR2 -> .

FACTOR -> intlit #A .

FACTOR -> floatlit #A .

FACTOR -> lpar ARITHEXPR rpar .

FACTOR -> not #S FACTOR #V .

FACTOR -> SIGN FACTOR #W .

FACTOR -> #B FACTOR1 .

FACTOR1 -> id #A FACTOR2 .

FACTOR2 -> #B REPTVARIABLE2 #AO FACTOR3 .

FACTOR3 -> dot #AQ FACTOR1 .

FACTOR3 -> #AM #AR .

FACTOR2 -> lpar APARAMS rpar FACTOR4 .

FACTOR4 -> dot #AQ FACTOR1 .

FACTOR4 -> #AM #AT .

FPARAMS -> id #A colon TYPE #G #B REPTFPARAMS3 #C #B REPTFPARAMS4 #E #F .

FPARAMS -> #AZ .

FPARAMSTAIL -> comma id #A colon TYPE #G #B REPTFPARAMSTAIL4 #C #D .

FUNCBODY -> lcurbr #B REPTFUNCBODY1 #AI rcurbr .

FUNCDEF -> FUNCHEAD FUNCBODY #AJ .

FUNCHEAD -> function Ffunction .

Ffunction -> id #A Fid2 .

Fid2 -> lpar FPARAMS rpar arrow RETURNTYPE #G #O1 .

Fid2 -> sr Fsr .

Fsr -> id #A lpar FPARAMS rpar arrow RETURNTYPE #G #O2 .

Fsr -> constructor lpar FPARAMS rpar #O3 .

FUNCTIONCALL -> id #A FFUNCid .

FFUNCid -> lpar APARAMS rpar FFUNCid2 .

FFUNCid -> REPTIDNEST1 dot #AQ FUNCTIONCALL.

FFUNCid2 -> dot #AQ FUNCTIONCALL.

FFUNCid2 -> .

IDNEST -> id Fid3 .

Fid3 -> REPTIDNEST1 dot .

Fid3 -> lpar APARAMS rpar dot .

INDICE -> lsqbr ARITHEXPR rsqbr .

LOCALVARDECL -> localvar Flocalvar .

Flocalvar -> id #A Fid .

Fid -> colon Fcolon .

Fcolon -> TYPE #G FTYPE .

FTYPE -> #B REPTLOCALVARDECL4 #C semi #U .

FTYPE -> lpar APARAMS rpar semi #U.

LOCALVARDECLORSTMT -> LOCALVARDECL .

LOCALVARDECLORSTMT -> STATEMENT .

MEMBERDECL -> MEMBERFUNCDECL .

MEMBERDECL -> MEMBERVARDECL .

MEMBERFUNCDECL -> function id #A colon lpar FPARAMS rpar arrow RETURNTYPE #G semi #H1 .

MEMBERFUNCDECL -> constructor colon lpar FPARAMS rpar semi #H2 .

MEMBERVARDECL -> attribute id #A colon TYPE #G #B REPTMEMBERVARDECL4 #C semi #I .

MULTOP -> mult #Q .

MULTOP -> div #Q .

MULTOP -> and #Q .

OPTCLASSDECL2 -> isa id #A #B REPTOPTCLASSDECL22 #L .

OPTCLASSDECL2 -> .

RELEXPR -> ARITHEXPR RELOP ARITHEXPR #Z .

RELOP -> eq #R .

RELOP -> neq #R .

RELOP -> lt #R .

RELOP -> gt #R .

RELOP -> leq #R .

RELOP -> geq #R .

REPTAPARAMS1 -> APARAMSTAIL REPTAPARAMS1 .

REPTAPARAMS1 -> .

REPTCLASSDECL4 -> VISIBILITY MEMBERDECL #J1or2 REPTCLASSDECL4 .

REPTCLASSDECL4 -> .

REPTFPARAMS3 -> ARRAYSIZE REPTFPARAMS3 .

REPTFPARAMS3 -> .

REPTFPARAMS4 -> FPARAMSTAIL REPTFPARAMS4 .

REPTFPARAMS4 -> .

REPTFPARAMSTAIL4 -> ARRAYSIZE REPTFPARAMSTAIL4 .

REPTFPARAMSTAIL4 -> .

REPTFUNCBODY1 -> LOCALVARDECLORSTMT REPTFUNCBODY1 .

REPTFUNCBODY1 -> .

REPTFUNCTIONCALL0 -> IDNEST REPTFUNCTIONCALL0 .

REPTFUNCTIONCALL0 -> .

REPTIDNEST1 -> INDICE REPTIDNEST1 .

REPTIDNEST1 -> .

REPTLOCALVARDECL4 -> ARRAYSIZE REPTLOCALVARDECL4 .

REPTLOCALVARDECL4 -> .

REPTMEMBERVARDECL4 -> ARRAYSIZE REPTMEMBERVARDECL4 .

REPTMEMBERVARDECL4 -> .

REPTOPTCLASSDECL22 -> comma id #A REPTOPTCLASSDECL22 .

REPTOPTCLASSDECL22 -> .

REPTSTART0 -> CLASSDECLORFUNCDEF REPTSTART0 .

REPTSTART0 -> .

REPTSTATBLOCK1 -> STATEMENT REPTSTATBLOCK1 .

REPTSTATBLOCK1 -> .

REPTVARIABLE2 -> INDICE REPTVARIABLE2 .

REPTVARIABLE2 -> .

RETURNTYPE -> TYPE .

RETURNTYPE -> void .

RIGHTRECARITHEXPR -> .

RIGHTRECARITHEXPR -> ADDOP TERM #Y RIGHTRECARITHEXPR .

RIGHTRECTERM -> .

RIGHTRECTERM -> MULTOP FACTOR #X RIGHTRECTERM .

SIGN -> plus #T .

SIGN -> minus #T .

STATBLOCK -> lcurbr #B REPTSTATBLOCK1 #AU rcurbr .

STATBLOCK -> STATEMENT .

STATBLOCK -> #AH2 .

STATEMENT -> if #AX lpar RELEXPR rpar then STATBLOCK else STATBLOCK semi #AY #AH1 .

STATEMENT -> while #AV lpar RELEXPR rpar STATBLOCK semi #AW #AH1 .

STATEMENT -> read #AL lpar #B VARIABLE #AM #AR rpar semi #AN #AH1 .

STATEMENT -> write #AD lpar EXPR #AA rpar semi #AF #AH1 .

STATEMENT -> return #AE lpar EXPR #AA rpar semi #AG #AH1 .

STATEMENT -> #B STATEMENT1 .

STATEMENT1 -> id #A STATEMENT2 .

STATEMENT2 -> #B REPTVARIABLE2 #AO STATEMENT3 .

STATEMENT3 -> dot #AQ STATEMENT1 .

STATEMENT3 -> semi .

STATEMENT3 -> #AM #AR ASSIGNOP EXPR #AA semi #AS #AH1 .

STATEMENT2 -> lpar APARAMS rpar STATEMENT4 .

STATEMENT4 -> dot #AQ STATEMENT1 .

STATEMENT4 -> #AM semi #AT #AH1 .

STATEMENT4 -> ASSIGNOP EXPR semi .

TERM -> FACTOR RIGHTRECTERM .

TYPE -> integer .

TYPE -> float .

TYPE -> id .

VARIABLE -> id #A FVARid .

FVARid -> #B REPTVARIABLE2 #AO FVARid2.

FVARid -> lpar APARAMS rpar dot #AQ VARIABLE .

FVARid2 -> dot #AQ VARIABLE .

FVARid2 -> .

VISIBILITY -> public #A .

VISIBILITY -> private #A .

VISIBILITY -> .

The following are the semantic action brief descriptions:

#A: pushes id node to stack (saves lexeme)
#B: pushes epsilon node to stack
#C: makes ArraySize node, gathers all children, pushes FamilyArraySize node
#D: makes FParamsTail node, gathers 3 children, pushes FamilyFParamsTail node
#E: makes FParamsTailList node, gathers all children, pushes FamilyFParamsTailList node
#F: makes FParams node, gathers 4 children, pushes FamilyFParams node
#G: pushes type node to stack (saves lexeme)
#H1: makes MemberFuncDecl node, gathers 3 children, pushes FamilyMemberFuncDecl node
#H2: makes MemberFuncDecl node, gathers 1 child, pushes FamilyMemberFuncDecl node
#I: makes MemberVarDecl node, gathers 3 children, pushes FamilyMemberVarDecl node
#J1: makes MemberDecl node, gathers 2 children, pushes FamilyMemberDecl node
#J2: makes MemberDecl node, gathers 1 child, pushes FamilyMemberDecl node
#K: makes MemberDeclList node, gathers all children, pushes FamilyMemberDeclList node
#L: makes InheritanceListTail node, gathers all children, pushes FamilyInheritanceListTail node
#M: makes InheritanceList node, gathers all children, pushes FamilyInheritanceList node
#N: makes ClassDecl node, gathers 3 children, pushes FamilyClassDecl node
#O1: makes FuncHead node, gathers 3 children, pushes FamilyFuncHead node
#O2: makes FuncHead node, gathers 4 children, pushes FamilyFuncHead node
#O3: makes FuncHead node, gathers 2 children, pushes FamilyFuncHead node
#P: pushes addOp node to stack (saves lexeme)
#Q: pushes multOp node to stack (saves lexeme)
#R: pushes relOp node to stack (saves lexeme)
#S: pushes not node to stack (saves lexeme)
#T: pushes sign node to stack (saves lexeme)
#U: makes LocalVarDecl node, gathers 3 children, pushes FamilyLocalVarDecl node

#V: pushes FamilyNot node
#W: pushes FamilySign node
#X: pushes FamilyMultOp node
#Y: pushes FamilyAddOp node
#Z: makes RelExpr node, gathers 3 children, pushes FamilyRelExpr node
#AA: makes Expr node, gathers 1 child, pushes FamilyExpr node
#AB: makes ExprTailList node, gathers all children, pushes FamilyExprTailList node
#AC: makes AParams node, gathers 2 children, pushes FamilyAParams node
#AC2: pushes empty AParams node
#AD: pushes write node to stack (saves lexeme)
#AE: pushes return node to stack (saves lexeme)
#AF: pushes FamilyWrite node
#AG: pushes FamilyReturn node
#AH1: makes Statement node, gathers 1 child, pushes FamilyStatement node
#AH2: pushes empty Statement node
#AI: makes FuncBody node, gathers all children, pushes FamilyFuncBody node
#AJ: makes FuncDef node, gathers 2 children, pushes FamilyFuncDef node
#AK: makes Prog node, gathers all children, pushes FamilyProg node
#AL: pushes read node to stack (saves lexeme)
#AM: makes IdNest node, gathers 2 children then gathers all other children (IdNestTemp), pushes FamilyIdNest node
#AN: pushes FamilyRead node
#AO: makes Indice node, gathers all children, pushes FamilyIndice node
#AP: makes AExpr node, gathers 1 child, pushes FamilyAExpr node
#AQ: makes IdNestTemp node, gathers 2 children, pushes FamilyIdNestTemp node
#AR: makes Variable node, gathers 3 children, pushes FamilyVariable node
#AS: makes AssignStat node, gathers 2 children, pushes FamilyAssignStat node
#AT: makes FunctionCall node, gathers 3 children, pushes FamilyFunctionCall node
#AU: makes StatBlock node, gathers all children, pushes FamilyStatBlock node
#AV: pushes while node to stack (saves lexeme)
#AW: pushes FamilyWhile node
#AX: pushes if node to stack (saves lexeme)
#AY: pushes FamilyIf node
#AZ: pushes empty FParams node
#BA: pushes EmptyArraySize node

2. Design

These are the following things I did to implement the parser augmented with syntax-directed translation to generate an AST data structure.

First I had to create an AST data structure. I did this the same way the professor did in his slides. By creating makeNode(), makeSiblings(),adoptChildren(), and makeFamily() functions. The AST data structure uses a Node class that stores the type and lexeme of every node. Nodes in the AST keep track of their parent, leftMostSibling, rightSibling, and leftMostChild.

I was then able to add semantic actions inside my parsing table (design of A1 and A2). Semantic actions always began with # and was followed by some capital letter or numbers. If #AB was found while parsing, the semantic function AB is called, and #AB is popped from the stack. The semantic functions push and pop as required on a new stack called the Semantic Stack. At the end of the program, only one node is left in the AST, the node prog that represents the whole program.

3. Use of Tools

Tools used:

1. Lecture slides provided by Joey Paquet for abstract tree generation techniques.
2. University of Calgary website for verifying grammars
   https://smlweb.cpsc.ucalgary.ca/start.html

Libraries used:

1. JTable: Used to create a table object for the parser table and first-follow table