Comp 442: Compiler Design

Concordia University, Montreal
Winter 2024
William Zijie Zhang
Concordia ID: 40176870

Assignment 1

William Zijie Zhang

1 Transformed Grammar into LL(1)

Find below the complete grammar after removing all EBNF notations, left recursions and ambuigities from the provided grammar.

```
START                 → prog
aParams               → expr rept-aParams1 | ε
aParamsTail           → , expr
addOp                 → + | - | or
arithExpr             → term rightrec-arithExpr
arraySize             → [ arraySize2
arraySize2            → intNum ] | ]
assignOp              → =
expr                  → arithExpr expr2
expr2                 → relOp arithExpr | ε
fParams               → id : type rept-fParams3 rept-fParams4 | ε
fParamsTail           → , id : type rept-fParamsTail4
factor                → id factor2 reptVariableOrFunc | intLit | floatLit
factor2               → ( aParams ) | rept-idnest1
reptVariableOrFunc    → idnest reptVariableOrFunc | ε
funcBody              → { rept-funcBody1 }
funcDecl              → funcHead ;
funcDef               → funcHead funcBody
funcHead              → func id ( fParams ) arrow returnType
functionCall          → rept-functionCall0 id ( aParams )
idnest                → . id idnest2
idnest2               → ( aParams ) | rept-idnest1
implDef               → impl id { rept-implDef3 }
indice                → [ arithExpr ]
memberDecl            → funcDecl | varDecl
multOp                → * | / | and
opt-structDecl2       → inherits id rept-opt-structDecl22 | ε
prog                  → rept-prog0
relExpr               → arithExpr relOp arithExpr
relOp                 → eq | neq | lt | gt | leq | geq
rept-aParams1         → aParamsTail rept-aParams1 | ε
rept-fParams3         → arraySize rept-fParams3 | ε
rept-fParams4         → fParamsTail rept-fParams4 | ε
```

```
rept-fParamsTail4      → arraySize rept-fParamsTail4 | ϵ
rept-funcBody1         → varDeclOrStat rept-funcBody1 | ϵ
rept-functionCall0     → idnest rept-functionCall0 | ϵ
rept-idnest1           → indice rept-idnest1 | ϵ
rept-implDef3          → funcDef rept-implDef3 | ϵ
rept-opt-structDecl22  → , id rept-opt-structDecl22 | ϵ
rept-prog0             → structOrImplOrFunc rept-prog0 | ϵ
rept-statBlock1        → statement rept-statBlock1 | ϵ
rept-structDecl4       → visibility memberDecl rept-structDecl4 | ϵ
rept-varDecl4          → arraySize rept-varDecl4 | ϵ
rept-variable0         → idnest rept-variable0 | ϵ
rept-variable2         → indice rept-variable2 | ϵ
returnType             → type | void
rightrec-arithExpr     → ϵ | addOp term rightrec-arithExpr
rightrec-term          → ϵ | multOp factor rightrec-term
sign                   → + | −
statBlock              → { rept-statBlock1 } | statement | ϵ
statement              → if ( relExpr ) then statBlock else statBlock ; | v
statement2             → rept-idnest1 statement3 | ( aParams ) statement4
statement3             → . id statement2 | assignOp expr ;
statement4             → . id statement2 | ;
structDecl             → struct id opt-structDecl2 { rept-structDecl4 } ;
structOrImplOrFunc     → structDecl | implDef | funcDef
term                   → factor rightrec-term
type                   → integer | float | id
varDecl                → let id : type rept-varDecl4 ;
varDeclOrStat          → varDecl | statement
variable               → id variable2
variable2              → rept-idnest1 reptvariable | ( aParams ) varIdnest
reptvariable           → varIdnest reptvariable | ϵ
varIdnest              → . id varIdnest2
varIdnest2             → ( aParams ) varIdnest | rept-idnest1
visibility             → public | private
```

2 First and Follow Sets

Table 1: First Set

nonterminal	first set
ADDOP	plus minus or
ARRAYSIZE2	intlit rsqbr
EXPR2	eq neq lt gt leq geq
FACTOR2	lpar lsqbr
FUNCBODY	lcurbr
FUNCHEAD	func
FPARAMS	id

FUNCTIONCALL	id dot
IDNEST2	lpar lsqbr
FUNCDECL	func
ARITHEXPR	intlit floatlit lpar not plus minus
RELOP	eq neq lt gt leq geq
APARAMSTAIL	comma
REPTAPARAMS1	comma
REPTFPARAMS3	lsqbr
FPARAMSTAIL	comma
REPTFPARAMS4	comma
REPTFPARAMSTAIL4	lsqbr
REPTFUNCBODY1	let if while read write return
REPTFUNCTIONCALL0	dot
REPTIMPLDEF3	func
REPTOPTSTRUCTDECL22	comma
REPTPROG0	struct impl func
MEMBERDECL	let func
ARRAYSIZE	lsqbr
REPTVARIABLE0	dot
INDICE	lsqbr
REPTVARIABLE2	lsqbr
IDNEST	dot
REPTVARIABLEORFUNC	dot
RETURNTYPE	void integer float id
RIGHTRECARITHEXPR	plus minus or
MULTOP	mult div and
SIGN	plus minus
START	struct impl func
PROG	struct impl func
REPTSTATBLOCK1	if while read write return
RELEXPR	intlit floatlit lpar not plus minus
STATBLOCK	lcurbr if while read write return
STATEMENT3	dot equal
ASSIGNOP	equal
EXPR	intlit floatlit lpar not plus minus
STATEMENT4	dot semi
ID	ϵ
STATEMENT2	lpar dot lsqbr equal
OPTSTRUCTDECL2	inherits
REPTSTRUCTDECL4	public private
STRUCTORIMPLORFUNC	struct impl func
STRUCTDECL	struct
IMPLDEF	impl
FUNCDEF	func
TERM	intlit floatlit lpar not plus minus
FACTOR	intlit floatlit lpar not plus minus

RIGHTRECTERM	mult div and
TYPE	integer float id
REPTVARDECL4	lsqbr
VARDECLORSTAT	let if while read write return
VARDECL	let
STATEMENT	if while read write return
VARIABLE	id
VARIABLE2	lpar lsqbr dot
REPTVARIABLE	dot
VARIDNEST2	lpar lsqbr
APARAMS	intlit floatlit lpar not plus minus
VARIDNEST	dot
REPTIDNEST1	lsqbr
VISIBILITY	public private

Table 2: Follow Set

Non-Terminal	Follow Set
ADDOP	intlit floatlit lpar not plus minus
ARRAYSIZE2	semi lsqbr rpar comma
EXPR2	comma rpar semi
FACTOR2	mult div and dot rsqbr
	eq neq lt gt leq geq plus
	minus or comma rpar semi
FUNCBODY	struct impl func rcurbr
FUNCHEAD	semi lcurbr
FPARAMS	rpar
FUNCTIONCALL	ϵ
IDNEST2	mult div and dot id rsqbr
	eq neq lt gt leq geq plus
	minus or comma rpar semi
FUNCDECL	rcurbr public private
ARITHEXPR	rsqbr eq neq lt gt leq geq comma rpar semi
RELOP	intlit floatlit lpar not plus minus
APARAMSTAIL	comma rpar
REPTAPARAMS1	rpar
REPTFPARAMS3	rpar comma
FPARAMSTAIL	comma rpar
REPTFPARAMS4	rpar
REPTFPARAMSTAIL4	comma rpar
REPTFUNCBODY1	rcurbr
REPTFUNCTIONCALL0	id
REPTIMPLDEF3	rcurbr
REPTOPTSTRUCTDECL22	lcurbr
REPTPROG0	ϵ

MEMBERDECL	rcurbr public private
ARRAYSIZE	semi lsqbr rpar comma
REPTVARIABLE0	ϵ
INDICE	equal mult div and lsqbr
	id dot rsqbr eq neq lt gt leq geq
	plus minus or comma rpar semi
REPTVARIABLE2	ϵ
IDNEST	mult div and dot id
	rsqbr eq neq lt gt leq
	geq plus minus or comma rpar semi
REPTVARIABLEORFUNC	mult div and rsqbr
	eq neq lt gt leq geq
	plus minus or comma rpar semi
RETURNTYPE	semi lcurbr
RIGHTRECARITHEXPR	rsqbr eq neq lt gt leq geq comma rpar semi
MULTOP	intlit floatlit lpar not plus minus
SIGN	intlit floatlit lpar not plus minus
START	ϵ
PROG	ϵ
REPTSTATBLOCK1	rcurbr
RELEXPR	rpar
STATBLOCK	else semi
STATEMENT3	else semi let if while
	read write return rcurbr
ASSIGNOP	intlit floatlit lpar not plus minus
EXPR	comma rpar semi
STATEMENT4	else semi let if while
	read write return rcurbr
ID	mult div and id lpar dot lsqbr equal
	rsqbr eq neq lt gt leq geq plus
	minus or comma rpar semi
STATEMENT2	else semi let if while
	read write return rcurbr
OPTSTRUCTDECL2	lcurbr
REPTSTRUCTDECL4	rcurbr
STRUCTORIMPLORFUNC	struct impl func
STRUCTDECL	struct impl func
IMPLDEF	struct impl func
FUNCDEF	struct impl func rcurbr
TERM	rsqbr eq neq lt gt leq
	geq plus minus or
	comma rpar semi
FACTOR	mult div and rsqbr eq neq
	lt gt leq geq plus
	minus or comma rpar semi
RIGHTRECTERM	rsqbr eq neq lt gt

	leq geq plus minus
	or comma rpar semi
TYPE	rpar lcurbr comma lsqbr semi
REPTVARDECL4	semi
VARDECLORSTAT	let if while read write return rcurbr
VARDECL	public private let if while read write return rcurbr
STATEMENT	else semi let if while read write return rcurbr
VARIABLE	rpar
VARIABLE2	rpar
REPTVARIABLE	rpar
VARIDNEST2	rpar dot
APARAMS	rpar
VARIDNEST	rpar dot
REPTIDNEST1	equal mult div and
	id dot rsqbr eq neq
	lt gt leq geq plus minus
	or comma rpar semi
VISIBILITY	let func

3 Design of recursive parser

In this section I will detail the overall structure of my implementation and explain some important helper methods. First and foremost, a lot of time was spent in the preparation phase before touching a single line of code. This consisted of familiarising myself with the provided grammar tools and applying techniques learned in class to derive an LL(1) grammar. The parser's goal is to validate the syntax of the code, it will get tokens from the lexer and make sure they form a correct sequence. To help in the implementation, it is crucial that every derivation must be deterministic, hence we need to remove ambiguities and left recursions.

Once I had obtained an LL(1) grammar, it was time to start the implementation. I once again chose to stray away from the table driven method because I was unfamiliar with the style and it seemed less intuitive to me. I began by creating a parser class that was initialized with a corresponding lexer. Then I proceeded by defining all the non-terminals from my grammar as methods of the parser. After that, the implementation simply consisted of calling the appropriate functions of the RHS of these non-terminals. I also made use of helper functions such as *match* and *nextToken* to iterate through the list of tokens provided by the lexer.

After making sure that my parser passed through the right sequence of function calls, I moved on to the derivation. This part was quite simple and consisted of keeping track of previous derivations and using the built-in Python string method *replace*() with the corresponding RHS of the non-terminals. In essence, the derivation serves as a visual representation of the steps that the predictive parser is taking.

Finally, I spent some time implementing the FIRST and FOLLOW sets of every non-terminal in my grammar. I made a crucial realisation that it would allow me to almost entirely remove the necessity to return a boolean for the non-terminal methods. In particular, I made heavy use of the FIRST set to see which sequence of tokens to consume given a non-terminal that had multiple options. I also use these sets to implement the error recovery method proposed in the lecture. This step required some fine-tuning of my implementation to allow for efficient skipping of syntax errors. However, it remains that I need to do a lot more testing to ensure robustness of my predictive recursive parser.

4 Tools

In this section I will be describing the set of tools and libraries I used in the analysis and implementation of my predictive parser. As mentioned above, I made use of the Java grammar converter tool along with the sample partial solution provided by Professor Paquet. This allowed me to get a head start towards deriving the LL(1) grammar. Then I made use of both recommended tools, namely FLACI and uCalgary CFG checker. The former served as my main tool for grammar verification along with testing valid parsing sequences provided by my classmates. The latter served its purpose by providing suggestions to make the grammar LL(1), which hinted at a general strategy to resolve ambiguities, along with a very valuable automatic generation of the FIRST and FOLLOW sets of every non-terminal. I also once again drew inspiration for the initial structure of my parser from the TeenyTinyCompiler tutorial. And finally, I have to declare that I also made heavy use of chatGPT along with a HTML table to LaTeX converter to turn the FIRST and FOLLOW sets from the uCalgary tool into it's equivalent Python code, using the appropriate $TokenType$ value from the lexer.