|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**Лабораторная работа № 8**

**«Множества FIRST для РБНФ»**

***по курсу «Конструирование компиляторов»***

Студент *Ионов Т.Р. 61Б*

Преподаватель *Коновалов А.B.*

*Москва, 2022 г.*

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc106227603)

[2 Индивидуальное задание 3](#_Toc106227604)

[3 Реализация 3](#_Toc106227605)

[4 Тестирование 6](#_Toc106227606)

[5 Особенности языка Kotlin 7](#_Toc106227607)

[Вывод 7](#_Toc106227608)

# 1 Цель работы

Целью данной работы является изучение алгоритма построения множеств FIRST для расширенной формы Бэкуса-Наура.

# 2 Индивидуальное задание

1. Составление описаний лексической структуры и грамматики входного языка на основе примера
2. Разработка лексического анализатора для входного языка
3. Разработка синтаксического анализатора для входного языка методом рекурсивного спуска
4. Реализация алгоритма вычисления множества FIRST для всех нетерминальных символов грамматики

Пример:

|  |  |
| --- | --- |
|  | <E <T { < |
|  | <+> |
|  | <-> |
|  | > T } > > |
|  | <T <F { < |
|  | <\*> |
|  | </> |
|  | > F } > > |
|  | <F <n> |
|  | <-F> |
|  | <( E ) > > |

# 3 Реализация

Грамматика:

S ::= R S | .  
R ::= <OPEN> <NTERM> A <CLOSE>  
A ::= <OPEN> A' <CLOSE> A | .  
A' ::= <NTERM> A' | <TERM> A' | <OPEN> I <CLOSE> A' | <IOPEN> A' <ICLOSE> A' | .  
I ::= <OPEN> A' <CLOSE> I | .

Листинг 1 – lexer.

import kotlin.system.exitProcess  
  
class Lexer(program: String) {  
 private var position = Position(program)  
  
 fun nextToken(): Token {  
 while (!position.isEOF()) {  
 while (position.isWhiteSpace())  
 position = position.next()  
  
 if (position.isEOF())  
 break  
  
 val token = when (position.getCode().toChar()) {  
 '<', '>', '{', '}' -> readSpecialToken(position)  
 in 'A'..'Z' -> readNterm(position)  
 '\*', '/', '+', '-', '(', ')', in 'a'..'z' -> readTerm(position)  
 else -> readUnknownToken(position)  
  
 }  
  
 position = token.coords.following  
  
 return token  
 }  
 return Token(DomainTag.*EOP*,  
 Fragment(position, position),  
 "")  
 }  
  
 private fun readUnknownToken(position: Position): Token {  
 *println*("ERROR ${Fragment(position, position)}: unrecognized token")  
 *exitProcess*(0)  
 }  
  
 private fun readSpecialToken(position: Position): Token {  
 val tag = when (position.getCurSymbol()) {  
 '<' -> DomainTag.*OPEN* '>' -> DomainTag.*CLOSE* '{' -> DomainTag.*IOPEN* '}' -> DomainTag.*ICLOSE* else -> DomainTag.*UNK* }  
 return Token(tag,  
 Fragment(position, position.next()),  
 position.getCurSymbol().*toString*())  
 }  
  
 private fun readNterm(position: Position): Token {  
 return Token(DomainTag.*NTERM*,  
 Fragment(position, position.next()),  
 position.getCurSymbol().*toString*())  
 }  
  
 private fun readTerm(position: Position): Token {  
 return Token(DomainTag.*TERM*,  
 Fragment(position, position.next()),  
 position.getCurSymbol().*toString*())  
 }  
}

Листинг 2 −Parser.

import kotlin.system.exitProcess  
  
class Parser(private val tokens: List<Token>) {  
 private var curToken = tokens.*first*()  
 private var ntermsLeft = HashSet<String>()  
 private var ntermsRight = HashSet<String>()  
 var mapRule = HashMap<String, Rule>()  
 private var curState = ""  
 private var states = *mutableListOf*<String>()  
 private var curStar = false  
  
 fun parse() {  
 parseS()  
*// mapRule.forEach { key, value -> println("<$key $value>") }* if (!ntermsLeft.containsAll(ntermsRight)) {  
 *println*("Undefined NTERMs:")  
 *println*(ntermsRight - ntermsLeft)  
 exit()  
 }  
 }  
 *// R -> <OPEN> <NTERM> A {A} <CLOSE>  
 // A -> <OPEN> E <CLOSE> {<OPEN> E <CLOSE>} | .  
 // E -> <NTERM> {E} | <TERM> {E}| <OPEN> I <CLOSE> {E} | <IOPEN> E <ICLOSE> {E} | .  
 // I -> <OPEN> E <CLOSE> {I} | .* private fun parseS() {  
 states.add("S")  
  
 while (curToken.tag == DomainTag.*OPEN*) {  
 parseR()  
 }  
 if (curToken.tag != DomainTag.*EOP*) exit()  
 }  
  
 *// R -> <OPEN> <NTERM> A {A} <CLOSE>* private fun parseR() {  
 states.add("R")  
  
 if (curToken.tag != DomainTag.*OPEN*) exit()  
 nextToken()  
  
 if (curToken.tag != DomainTag.*NTERM*) exit()  
 val left = curToken  
 val rule = Rule(RuleTag.*Token*, null)  
 ntermsLeft.add(left.value)  
 nextToken()  
 parseA(rule)  
 while (curToken.tag == DomainTag.*OPEN*) {  
 rule.addAlternatives()  
 parseA(rule)  
 }  
 mapRule[left.value] = rule  
 if (curToken.tag != DomainTag.*CLOSE*) exit()  
 nextToken()  
 }  
  
 *// A -> <OPEN> E <CLOSE> {<OPEN> E <CLOSE>} | .* private fun parseA(rule: Rule) {  
 curState = "A"  
 states.add(curState)  
  
 if (curToken.tag != DomainTag.*OPEN*) exit()  
 nextToken()  
  
 val newRule = Rule(RuleTag.*Normal*, null)  
 newRule.addAlternatives()  
 if (curToken.tag == DomainTag.*IOPEN*){  
 newRule.tag = RuleTag.*NormalStar* }  
 parseE(newRule, false)  
  
 if (curToken.tag != DomainTag.*CLOSE*) exit()  
 nextToken()  
 rule.addAlternatives()  
 rule.addRule(newRule)  
  
 while (curToken.tag == DomainTag.*OPEN*) {  
 nextToken()  
 val newRule = Rule(RuleTag.*Normal*, null)  
 if (curToken.tag == DomainTag.*IOPEN*){  
 newRule.tag = RuleTag.*NormalStar* }  
 newRule.addAlternatives()  
 parseE(newRule, false)  
 if (curToken.tag != DomainTag.*CLOSE*) exit()  
 nextToken()  
 rule.addAlternatives()  
 rule.addRule(newRule)  
 }  
 }  
  
 *// E -> <NTERM> E | <TERM> E | <OPEN> I <CLOSE> E | <IOPEN> E <ICLOSE> E | .* private fun parseE(rule: Rule, star: Boolean) {  
 curState = "E"  
 states.add(curState)  
  
 when (curToken.tag) {  
 DomainTag.*NTERM*, DomainTag.*TERM* -> {  
 val token = curToken  
 if (curToken.tag == DomainTag.*NTERM*) {  
 ntermsRight.add(token.value)  
 }  
 rule.addRule(Rule(RuleTag.*Token*, token))  
 nextToken()  
 parseE(rule, false)  
 }  
 DomainTag.*OPEN* -> {  
 nextToken()  
 parseI(rule, star)  
 if (curToken.tag != DomainTag.*CLOSE*) exit()  
 nextToken()  
 parseE(rule, false)  
 }  
 DomainTag.*IOPEN* -> {  
 nextToken()  
 val iRule = Rule(RuleTag.*NormalStar*, null)  
 iRule.addAlternatives()  
 parseE(iRule, true)  
 if (curToken.tag != DomainTag.*ICLOSE*) exit()  
 nextToken()  
 rule.addRule(iRule)  
 parseE(rule, false)  
 }  
 DomainTag.*EOP* -> return  
 }  
 }  
  
 *// I -> <OPEN> E <CLOSE> I | .* private fun parseI(rule: Rule, star: Boolean) {  
 curStar = star  
 curState = "I"  
 states.add(curState)  
  
 when (curToken.tag) {  
 DomainTag.*EOP* -> return  
 DomainTag.*OPEN* -> {  
 nextToken()  
 rule.addAlternatives()  
 parseE(rule, star)  
 if (curToken.tag != DomainTag.*CLOSE*) exit()  
 nextToken()  
 parseI(rule, star)  
 }  
 DomainTag.*TERM*, DomainTag.*NTERM* -> {  
 nextToken()  
 parseE(rule, star)  
 }  
 }  
 }  
  
  
 private fun nextToken() {  
 val index = tokens.indexOf(curToken)  
 curToken = tokens[index + 1]  
*// print("curToken: $curToken on state $curState\n")* }  
  
  
 private fun printNextTokens() {  
 *println*("Next tokens! :")  
 val index = tokens.indexOf(curToken)  
 for (i in index *until* tokens.size)  
 *println*(tokens[i])  
 *println*("End of next tokens! ")  
 }  
  
 private fun exit() {  
 *println*("ERROR")  
 *println*("On token: $curToken")  
 *println*(curState)  
*// println("passed states: $states")  
 exitProcess*(0)  
 }  
}

Листинг 3 – вычисление first

class First(val rules: HashMap<String, Rule>) {  
 var first = HashMap<String, HashSet<String>>()  
 *//var set: HashSet<String>? = null* fun f(rule: Rule): HashSet<String>? {  
 var altSet: HashSet<String>? = null  
 val set = HashSet<String>()  
 for (ruleList in rule.alternatives) {  
 altSet = HashSet<String>()  
 var hashSet: HashSet<String>? = HashSet<String>()  
 altSet.add("ε")  
 for (item in ruleList) {  
 if (!altSet.contains("ε"))  
 break  
 if (item.token?.tag == DomainTag.*NTERM* && item.tag == RuleTag.*Token*)  
 hashSet = first[item.token.value]?.clone() as HashSet<String>  
 else if (item.token?.tag == DomainTag.*NTERM* && item.tag == RuleTag.*TokenStar*) {  
 hashSet = first[item.token.value]?.clone() as HashSet<String>  
 *//hashSet = first[item.token.value]* hashSet.add("ε")  
 }  
 else if (item.token?.tag == DomainTag.*TERM* && item.tag == RuleTag.*Token*) {  
 hashSet?.clear()  
 hashSet?.add(item.token.value)  
 } else if (item.token?.tag == DomainTag.*TERM* && item.tag == RuleTag.*TokenStar*) {  
 hashSet?.clear()  
 hashSet?.add(item.token.value)  
 hashSet?.add("ε")  
 }  
 else if (item.tag == RuleTag.*Normal*)  
 hashSet = f(item)  
 else if (item.tag == RuleTag.*NormalStar*) {  
 hashSet = f(item)  
 hashSet?.add("ε")  
 }  
 altSet.remove("ε")  
 hashSet?.*forEach* **{** altSet?.add(**it**) **}** }  
 altSet.*forEach* **{** set.add(**it**) **}** }  
 return set  
 }  
  
 fun setFirst() {  
 rules.forEach **{** key, \_ **->** first.put(key, HashSet<String>()) **}** var isChanged = true  
 var hs: HashSet<String>? = null  
 while (isChanged) {  
 isChanged = false  
 rules.forEach **{** key, value **->** hs = HashSet<String>()  
 hs = f(value)  
 val len = first[key]?.size  
 */\*f(value)?.forEach {  
 hs?.add(it)  
 } \*/* first[key] = hs?.clone() as HashSet<String>  
 if (len != first[key]?.size)  
 isChanged = true  
 **}** }  
  
 }  
  
 fun printFirst() {  
 setFirst()  
 first.forEach **{** key, value **->** *print*("$key :: ")  
 var s = ""  
 value.*forEach* **{** s = s.plus("$**it**, ")  
 **}** *println*(s.*removeSuffix*(", "))  
 **}** }  
}

# 4 Тестирование

Вход:

<E <T { <  
 <+>  
 <->  
 > T } > >  
<T <F { <  
 <\*>  
 </>  
 > F } > >  
<F <n>  
 <- F>  
 <( E ) > >

Результат:

T :: (, -, n

E :: (, -, n

F :: (, -, n

# 5 Особенности языка Kotlin

Одной из самых полезных фич я бы назвал безопасность нулевого указателя.

Пример

var a: String = "abc" // Regular initialization means non-null by default

a = null // compilation error

------

var b: String? = "abc" // can be set to null

b = null // ok

print(b)

val l = a.length

val l = b.length // error: variable 'b' can be null

Можно организовывать цепочки безопасных вызовов:

bob?.department?.head?.name

Вернет null, если какое-либо null

Также, примечателен оператор !!, преобразующий любое значение в ненулевой тип.

В своей программе, я использовал безопасный доступ, особенно при работе с множествами first:

hashSet?.*forEach* **{** altSet?.add(**it**) **}**

в процессе разработки лексера, было удобно использовать when выражения:

val tag = when (position.getCurSymbol()) {  
 '<' -> DomainTag.*OPEN* '>' -> DomainTag.*CLOSE* '{' -> DomainTag.*IOPEN* '}' -> DomainTag.*ICLOSE* else -> DomainTag.*UNK*}

Улучается читабельность кода

Условия сопоставления можно передать через запятую:

val token = when (position.getCode().toChar()) {  
 '<', '>', '{', '}' -> readSpecialToken(position)  
 in 'A'..'Z' -> readNterm(position)  
 '\*', '/', '+', '-', '(', ')', in 'a'..'z' -> readTerm(position)  
 else -> readUnknownToken(position)  
  
}

А принадлежность промежутку черезе in \* .. \*

Удобно было использовать присвоение when выражения кортежу

val (start, end) = when (tag) {  
 RuleTag.*Normal* -> "<" *to* ">"  
 RuleTag.*Token* -> "" *to* ""  
 RuleTag.*TokenStar* -> "" *to* ""  
 RuleTag.*NormalStar* -> "{" *to* "}"  
 else -> "ERROR" *to* "ERROR"  
}

Kotlin похож на Java и разрабатывался как язык программирования, который был бы лучше джавы, но все еще функционально совместим с ним.

# Вывод

В результате выполнения данной лабораторной работы был изученен генератор лексических анализаторов flex.