Parser

Github link:

https://github.com/CimpeanAndreea/FLCD/tree/master/Lab5

Problem statement

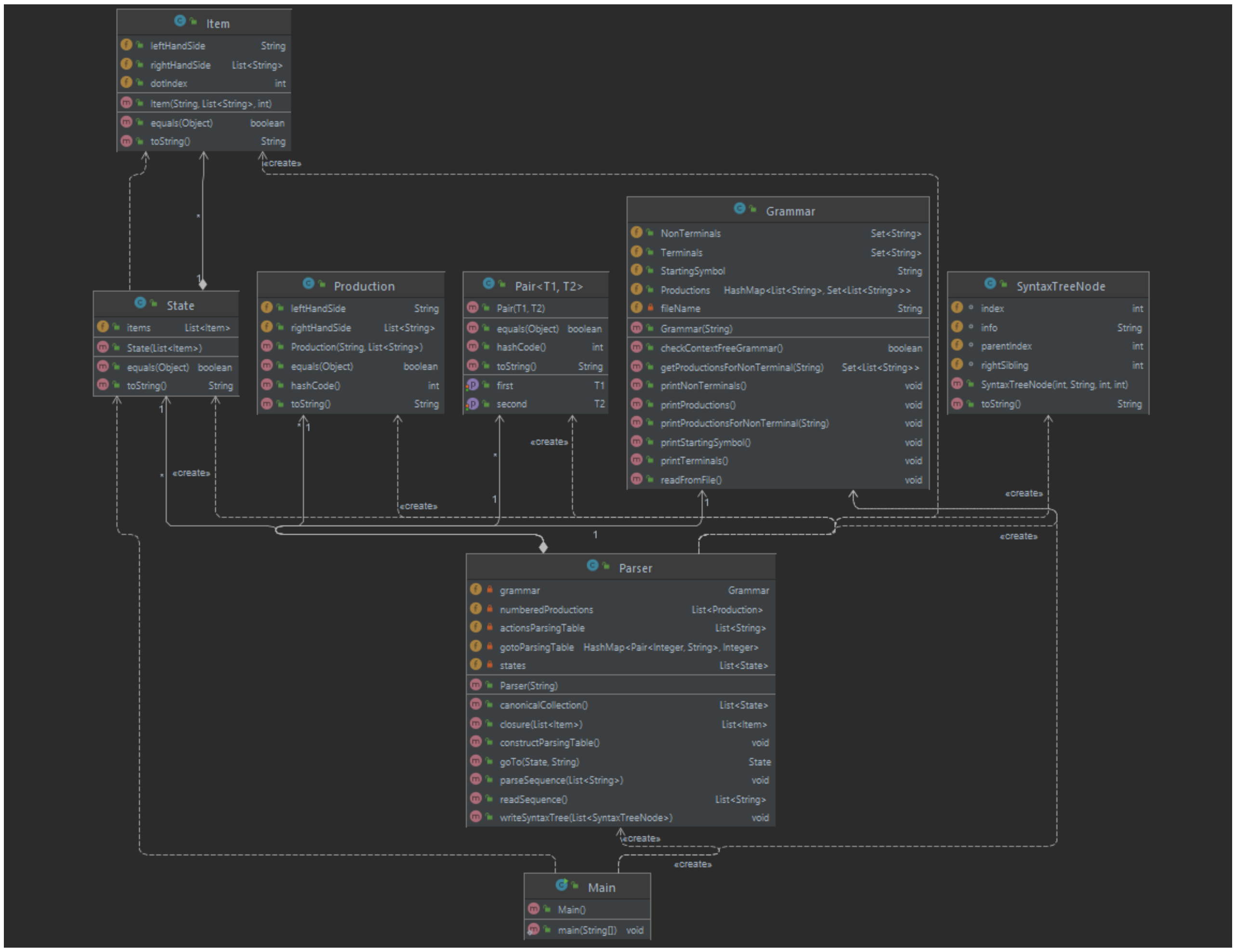
PART 3: Deliverables

1. Algorithms corresponding to *parsing table* (if needed) and *parsing strategy*

2. Class *ParserOutput* - DS and operations corresponding to choice 2.a/2.b/2.c (Lab 5) (required operations: transform parsing tree into representation; print DS to screen and to file)

Remark: If the table contains conflicts, you will be helped to solve them. It is important to print a message containing row (symbol in LL(1), respectively state in LR(0)) and column (symbol) where the conflict appears. For LL(1), values (α,i) might also help.

Implementation


Item
leftHandSide String
rightHandSide List<String>
dotIndex int
Item(String, List<String>, int)
equals(Object) boolean
toString() String
«create»
State
items List<Item>
State(List<Item>)
equals(Object) boolean
toString() String
Production
leftHandSide String
rightHandSide List<String>
Production(String, List<String>)
equals(Object) boolean
hashCode() int
toString() String
Pair<T1, T2>
Pair(T1, T2)
equals(Object) boolean
hashCode() int
toString() String
first T1
second T2
Grammar
NonTerminals Set<String>
Terminals Set<String>
StartingSymbol String
Productions HashMap<List<String>, Set<List<String>>>
fileName String
Grammar(String)
checkContextFreeGrammar() boolean
getProductionsForNonTerminal(String) Set<List<String>>
printNonTerminals() void
printProductions() void
printProductionsForNonTerminal(String) void
printStartingSymbol() void
printTerminals() void
readFromFile() void
SyntaxTreeNode
index int
info String
parentIndex int
rightSibling int
SyntaxTreeNode(int, String, int, int)
toString() String
Parser
grammar Grammar
numberedProductions List<Production>
actionsParsingTable List<String>
gotoParsingTable HashMap<Pair<Integer, String>, Integer>
states List<State>
Parser(String)
canonicalCollection() List<State>
closure(List<Item>) List<Item>
constructParsingTable() void
goTo(State, String) State
parseSequence(List<String>) void
readSequence() List<String>
writeSyntaxTree(List<SyntaxTreeNode>) void
Main
Main()
main(String[]) void
«create»
1
*

Continuing the previous work more functions were added to the Parser class and one more class **SyntaxTreeNode.** In order to parse a given sequence, after building the canonical collection of the LR(0) parser, there needs to be build a parsing table, with 2 parts: ACTION andGOTO, having one row for each obtained state, one column for the action part and one column for each symbol from the terminals and nonterminals of the grammar in the goto part.

<u>function constructParsingTable</u>()

description: construct the parsing table following the next steps:

1. If there is an item in the state containing dot somewhere besides the last position, then action(state) = shift
2. If there is an item [A->Beta.] in the state (that has dot at the end) then action(state) = reduce P, where P is the number of the production A->Beta from the grammar
3. If [S`->S.] belongs to the state then action(state)=accept
4. if goto(si, X) = sj then goto(si, X)=sj
   Conflicts:

If there are 2 items in a state that have the property from step 2, there is a reduce-reduce conflict, if there is an item with the property from 1 and another with a property from 2, there is a shift-reduce conflict. If there are conflicts print a message.

post: completes the 2 lists from the parser (**actionsParsingTable** and **gotoParsingTable**)

<u>function parseSequence(String[] sequence):</u>

pre: constructed the parsing table

description: parse a sequence to see if it is accepted by the grammar

Following the configuration (workingStack, inputStack, outputStack):

- add in the working stack: $0, which means initial state, inputStack is the sequence and outputStack is empty at the beginning
- repeat the following steps until no error occurred or parsed the entire sequence and it is accepted:
    1. get the number of the current state that is always on the top of the working stack and the action corresponding to it from the actionsParsingTable
    2. if action is shift and still have symbols to parse in the inputStack, move the symbol from the top of the inputStack to the workingStack and also add the result state of goto(current state, symbol) to the workingStack, if the action is shift and no more symbols in the inputStack or the result of mentioned goto is null => Error
    3. if action is accept and parsed all the sequence build the syntax tree from the outputStack, the outputStack contains a sequence of numbers representing the order of the numbered productions that need to be applied to obtain the input sequence, if not all the sequence was parsed => Error
    4. if action is reduce P, P is the number of a production and need to replace in the working stack this production in the following way:
        a. in the working stack, the top contains this exact production, the right hand side of it, only that after each symbol of the production there is a number for state
        b. replace all this right hand side of the production in the working stack with the left hand side and also add the result of goto(first state from top of working state, left hand side) and in the output stack add P, if the result of goto is null => Error

post: syntax tree or error

A syntax tree contains nodes of the form (index, info, parent, right sibling) and is constructed from the output stack in the following manner:

- Root is the starting symbol of the grammar, and has index 1

- Leaves are terminal symbols and construct the sequence
- The output stack gives the order of the productions
- The relation father -> children corresponds to lhs -> rhs of a production
- Rightmost derivations are applied, and nodes on the right have higher indexes
- Therefore, represent the tree as a list of nodes, at each step take one production from the stack, search in the list the node that has as info the lhs and no children, beginning from the end of the list (in case of same nodes having same symbol, the rightmost will be chosen), and add the rhs of the production as children

Tests

g1.txt

```
S L
x ( ) ,
S
S ::= ( L )
S ::= x
L ::= S
L ::= L , S
```

seq.txt

```
( x , ( x ) )
```

Output

CONSTRUCTING PARSER TABLE

NUMBERED PRODUCTIONS

1: S->(L)

2: S->x

3: L->S

4: L->L,S

ACTIONS

[shift, acc, reduce 2, shift, reduce 3, shift, reduce 1, shift, reduce 4]

GOTO

(state 0,symbol S) = production 1

(state 0,symbol x) = production 2

(state 0,symbol () = production 3

(state 3,symbol S) = production 4

```
(state 3,symbol L) = production 5
(state 3,symbol x) = production 2
(state 3,symbol () = production 3
(state 5,symbol )) = production 6
(state 5,symbol ,) = production 7
(state 7,symbol S) = production 8
(state 7,symbol x) = production 2
(state 7,symbol () = production 3
PARSING...
[$, 0]
shift
[$, 0, (, 3]
shift
[$, 0, (, 3, x, 2]
reduce 2
[$, 0, (, 3, S, 4]
reduce 3
[$, 0, (, 3, L, 5]
shift
[$, 0, (, 3, L, 5, ,, 7]
shift
[$, 0, (, 3, L, 5, ,, 7, (, 3]
shift
[$, 0, (, 3, L, 5, ,, 7, (, 3, x, 2]
reduce 2
[$, 0, (, 3, L, 5, ,, 7, (, 3, S, 4]
reduce 3
[$, 0, (, 3, L, 5, ,, 7, (, 3, L, 5]
shift
[$, 0, (, 3, L, 5, ,, 7, (, 3, L, 5, ), 6]
reduce 1
[$, 0, (, 3, L, 5, ,, 7, S, 8]
reduce 4
```

```
[$, 0, (, 3, L, 5]
shift
[$, 0, (, 3, L, 5, ), 6]
reduce 1
[$, 0, S, 1]
acc
```

ACCEPT

OUTPUT STACK

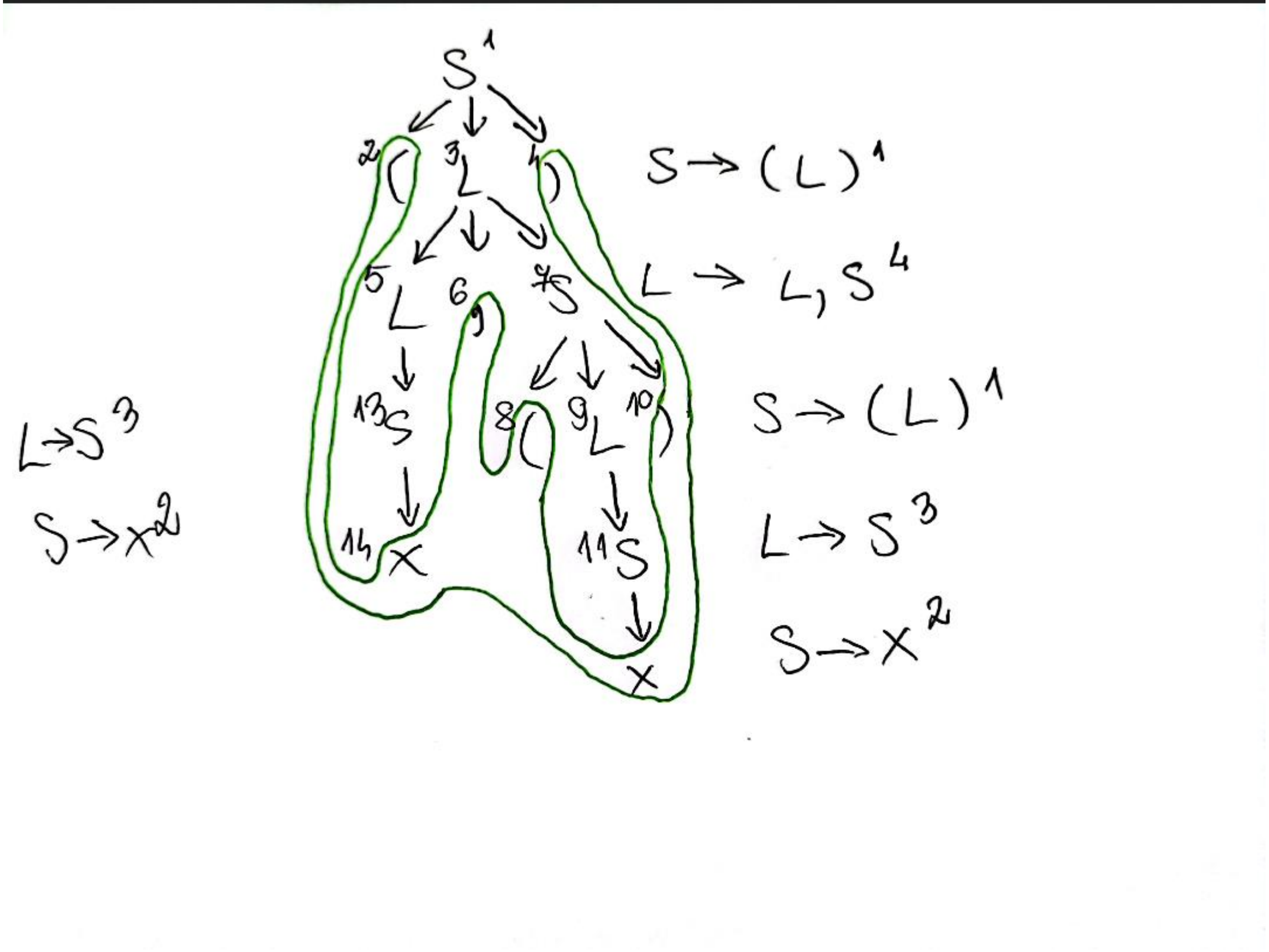
```
[1, 4, 1, 3, 2, 3, 2]
```

DERIVATIONS

```
S => ( L ) => ( L , S ) => ( L , ( L ) ) => ( L , ( S ) ) => ( L , ( x ) ) => ( S , ( x ) ) => ( x , ( x ) )
```

SYNTAX TREE

```
SyntaxTreeNode{index=1, info='S', parentIndex=0, rightSibling=0}
SyntaxTreeNode{index=2, info='(', parentIndex=1, rightSibling=0}
SyntaxTreeNode{index=3, info='L', parentIndex=1, rightSibling=2}
SyntaxTreeNode{index=4, info=')', parentIndex=1, rightSibling=3}
SyntaxTreeNode{index=5, info='L', parentIndex=3, rightSibling=0}
SyntaxTreeNode{index=6, info=',', parentIndex=3, rightSibling=5}
SyntaxTreeNode{index=7, info='S', parentIndex=3, rightSibling=6}
SyntaxTreeNode{index=8, info='(', parentIndex=7, rightSibling=0}
SyntaxTreeNode{index=9, info='L', parentIndex=7, rightSibling=8}
SyntaxTreeNode{index=10, info=')', parentIndex=7, rightSibling=9}
SyntaxTreeNode{index=11, info='S', parentIndex=9, rightSibling=0}
SyntaxTreeNode{index=12, info='x', parentIndex=11, rightSibling=0}
SyntaxTreeNode{index=13, info='S', parentIndex=5, rightSibling=0}
SyntaxTreeNode{index=14, info='x', parentIndex=13, rightSibling=0}
```



Seq2.txt

```
x , ( x ) )
```

PARSING...

[$, 0]

shift

[$, 0, x, 2]

reduce 2

[$, 0, S, 1]

acc

Error on symbol ,