Lab 2 LFCT

The data representation of Grammar:

The nonterminals , terminals and starting symbols are stored in Character Sets: Set<Character>

The productions are stored as such:

* Each non-terminal has 0 or more modifying ways.
* Each way is composed of a Set of terminals and nonterminals
* All the non-terminal's ways form a list
* The non-terminal and it's way-list form a mapping in the grammar's productions map

The method to check whether the grammar is regular:

public boolean isRegular() throws MissingElemException {

if(isRightLinear() && passEpsilon())

return true;

return false;

}

public boolean isRightLinear(){

List<Character> aux;

for(Character c :productions.keySet()){

for(Set<Character> s: productions.get(c)){

aux = new ArrayList<>(s);

boolean isStart = aux.size()==1 && aux.get(0).equals('$');

if(!nonTerminals.contains(aux.get(aux.size()-1)) && !isStart)

return false; } }

return true;

}

public boolean passEpsilon() throws MissingElemException {

if(!onlyOneEpsilon && !S.isEmpty()){

return false;

}

for(Character c: productions.keySet()){

for(Set<Character> s: productions.get(c)){

List<Character> startingSet = new ArrayList<>(getS());

for (Character ch: startingSet){

if(!getS().isEmpty() && s.contains(ch)){

return false; } } }

} return true;

}

The method to convert the grammar to automaton:

public Automaton toAutomaton() throws GrammarException {

Automaton aut = new Automaton();

if(!isRegular())

throw new GrammarException("The chosen grammar is not regular.");

Set<Character> states = nonTerminals;

aut.getTransitions().put(new Character('K'),new ArrayList<>());

aut.setStates(states);

aut.setQ0(S);

aut.setAlphabet(terminals);

aut.setTransitions(productions);

Set<Character> finals = new LinkedHashSet<>();

finals.add(new Character('K'));

if(!S.isEmpty()){

finals.addAll(S); }

aut.setF(finals);

return aut;

}

The data representation of automaton:

The states, alphabet, starting set and final set are stored in Character Sets: Set<Character>

The transitions are set in a similar design to the grammar's productions:

There, the productions of one elem were sets inside a list A=[[a, D], [e, A]]

Here, (A,a) -> D and (A,e) -> A

The method to convert the automaton to grammar:

public Grammar toGrammar() throws MissingElemException {

Grammar g = new Grammar();

g.setNonTerminals(states);

g.setTerminals(alphabet);

if(!this.Q0.isEmpty()){

Q0.stream().forEach(q0 -> g.getS().add(q0));

}

g.setProductions(transitions);

if(!F.isEmpty()){

List<Character> aux = new ArrayList<>(F);

for(Character c: aux){

for(Set<Character> set: new Cloner().deepClone(transitions.get(c))){

List<Character> transAux = new ArrayList<>(set);

if(transAux.get(0).equals('$')) continue;

if(transAux.size()>1){

Set<Character> newSet1 = new LinkedHashSet<>();

newSet1.add(transAux.get(0));

g.getProductionsOfNonterminal(c).addAll(Arrays.asList(newSet1));

}

}

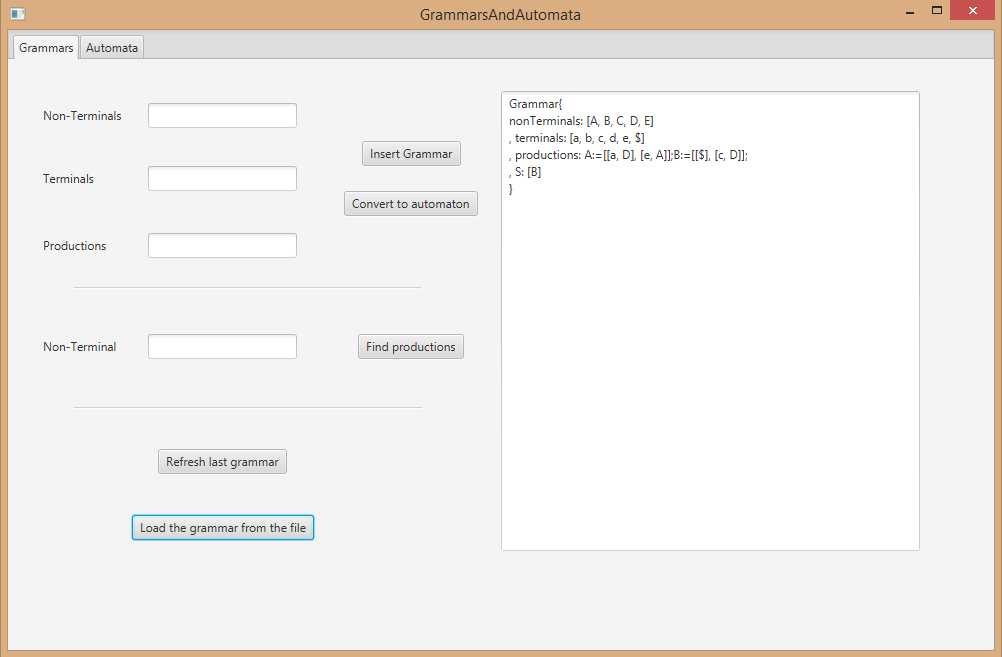
}

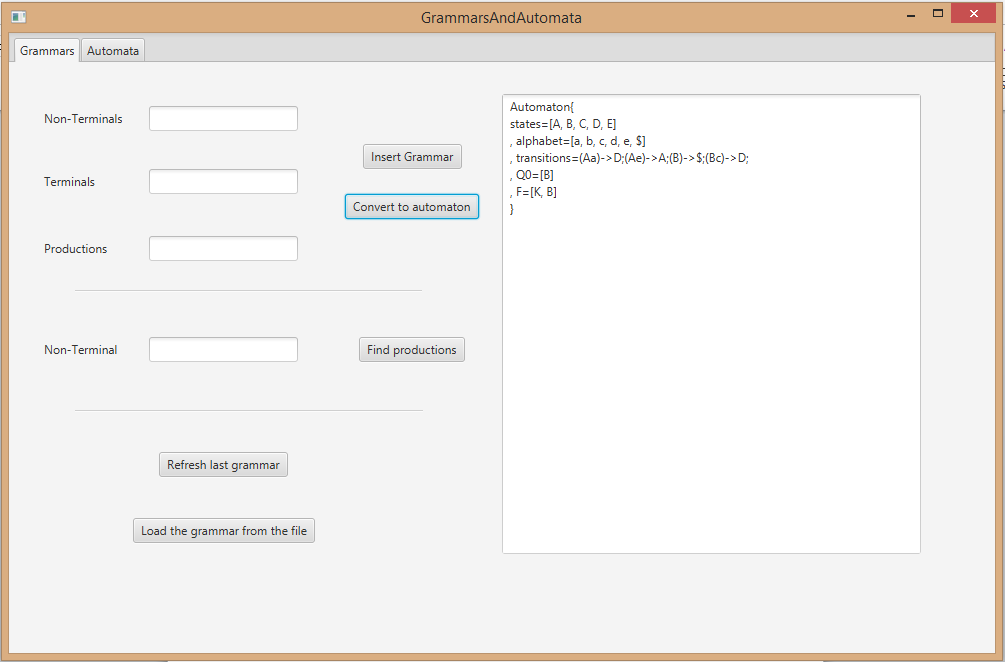
}

return g;

}

Example to convert grammar to automaton:





Example of conversion from aut to grammar:

