Ministerul Educaţiei și Cercetării al Republicii Moldova Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică și Microelectronică

Laboratory work 2:

Determinism in Finite Automata. Conversion from NDFA to DFA. Chomsky Hierarchy.

Elaborated:

st. gr. FAF-223 Ciornii Alexandr

Verified:

asist. univ. Dumitru Cretu

Chișinău - 2024

Objectives:

Understand what an automaton is and what it can be used for.

Continuing the work in the same repository and the same project, the following need to be added:

a. Provide a function in your grammar type/class that could classify the grammar based on Chomsky hierarchy.

b. For this you can use the variant from the previous lab.

According to your variant number (by universal convention it is register ID), get the finite automaton definition and do the following tasks:

a. Implement conversion of a finite automaton to a regular grammar.

b. Determine whether your FA is deterministic or non-deterministic.

c. Implement some functionality that would convert an NDFA to a DFA.

d. Represent the finite automaton graphically (Optional, and can be considered as a bonus point):

You can use external libraries, tools or APIs to generate the figures/diagrams.

Your program needs to gather and send the data about the automaton and the lib/tool/API return the visual representation.

Please consider that all elements of the task 3 can be done manually, writing a detailed report about how you've done the conversion and what changes have you introduced. In case if you'll be able to write a complete program that will take some finite automata and then convert it to the regular grammar - this will be a good bonus point.

Implementation description

Here's method in Grammar class that identifies type of the grammar

```
public int GetChomskyType()
{
    var list = new List<Char>();
    foreach (var hash in productionRules.Values)
    {
        foreach (var VARIABLE in hash)
        {
            list.Add(IsThird(VARIABLE));
        }
    }
    var left = new List<Char>() { 'l', 'b' };
    var right = new List<Char>() { 'r', 'b' };
  //  if (productionRules.Values.All(set => (set.All(s => left.Contains(IsThird(s))) || set.All(s => right.Contains(IsThird(s))))))
  if(!(list.Contains('l') && list.Contains('r')))
```

```
            return 3;
        if (productionRules.Keys.All(key => nonterminals.Contains(key[0])))
            return 2;
        if (productionRules.All(kv => kv.Value.All(str => str.Length < kv.Key.Length)))
            return 1;
        if (productionRules.Keys.All(key => !string.IsNullOrEmpty(key)))
            return 0;
        return -1;
    }
```

Here's method in NFA class that create object of Grammar type

```
public Grammar CreateGrammar()
{
    var terms = states.SelectMany(s => s.Transitions.Select(t => t.Symbol).Select(symbol =>symbol)).Distinct().ToList();

    List<char> nonterms = new List<char>(){'S'};
    for (int i = 0; i < states.Count() - 1; i++)
    {
        nonterms.Add((char)((char)65+i));
    }
    Grammar grammar = new Grammar(terms, nonterms);
    foreach (var state in states)
    {
      //  if(!state.Transitions.Any())
        //    grammar.AddRule(nonterms[state.StateId].ToString(), "");
        foreach (var trans in state.Transitions)
        {
            grammar.AddRule(nonterms[state.StateId].ToString(), trans.Symbol.ToString() + nonterms[trans.TargetState].ToString());
        }
    }
    return grammar;
}
```

Here's method that check if FA is deterministic

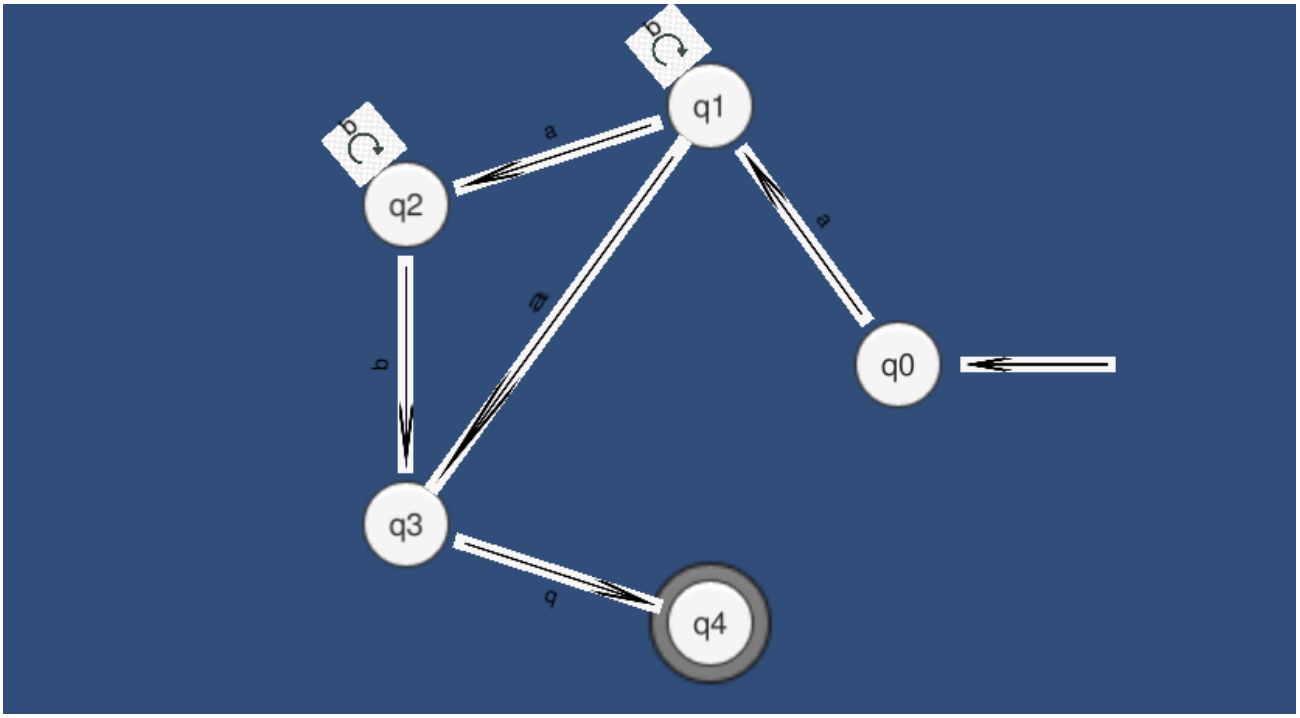
```
public bool IsDeterministic()
{
   // return states.All(state => state.Transitions.Count == 1);
    return states.All((state => state.Transitions.GroupBy(t => t.Symbol).All(g => g.Count() == 1)));
}
```

Transition nfa → dfa I did manually

```
dfa.AddState(0, isAccepting: false); // Initial state
dfa.AddState(1, isAccepting: false); // Accepting state
dfa.AddState(2, isAccepting: false); // Accepting state
dfa.AddState(3, isAccepting: false); // Accepting state
dfa.AddState(4, isAccepting: true); // Accepting state
//dfa.AddState(5, isAccepting: true); // Accepting state


// Transitions
dfa.AddTransition(0, 'a', 1);
dfa.AddTransition(1, 'b', 1);
dfa.AddTransition(1, 'a', 2);
dfa.AddTransition(2, 'b', 3);
dfa.AddTransition(3, 'b', 4);
dfa.AddTransition(3, 'a', 1);
dfa.AddTransition(4, 'a', 1);
dfa.AddTransition(4, 'b', 4);
```

Here’s visual representation of the Nfa



Note: it accepts any object of class NFA

Conclusions and Results

In the course of this laboratory work, substantial progress was made towards achieving the set objectives. The introduction of a function within the grammar type/class to classify the grammar based on the Chomsky hierarchy was a pivotal development. This addition empowers the system with a capacity to categorize grammars, providing a valuable analytical tool for understanding their structural complexities.

A significant milestone was the successful implementation of the conversion process from a finite automaton to a regular grammar. This accomplishment signifies a crucial link between automata and formal language structures, enhancing our ability to analyze and manipulate different types of languages.

Addressing the deterministic or non-deterministic nature of the finite automaton adds depth to our understanding of its behavior. This distinction holds practical significance in various applications, from language recognition to algorithm design, as it informs us about the predictability and precision of the automaton's responses.

The implementation of functionality to convert a non-deterministic finite automaton (NDFA) to a deterministic finite automaton (DFA) represents a substantial step forward. This process not only streamlines the automaton but also contributes to the efficiency of language recognition systems. The ability to automate this conversion process enhances our toolkit for handling diverse automata types.

Delving into the optional graphical representation of the finite automaton added an extra layer of sophistication to the project. The use of external libraries, tools, or APIs for generating visual representations offers a tangible and intuitive way to comprehend the

intricacies of the automaton's structure. This visual representation provides a valuable aid in explaining complex concepts and can be particularly beneficial for educational purposes.

In summary, the laboratory work has resulted in a well-rounded achievement, encompassing both theoretical understanding and practical implementation. The developed functionalities contribute to the versatility of the system, allowing for a more thorough exploration of automata and formal language theory. The successful completion of these tasks not only fulfills the objectives but also positions the project as a robust tool for further studies in the field.