Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики

Кафедра информатики и прикладной математики

**Домашняя работа №2**

**Дисциплина «Прикладная математика»**

**Вариант 21**

**Выполнил:**

Съестов Дмитрий Вячеславович

Группа P3317

**Преподаватель:**

Лаздин Артур Вячеславович

Санкт-Петербург

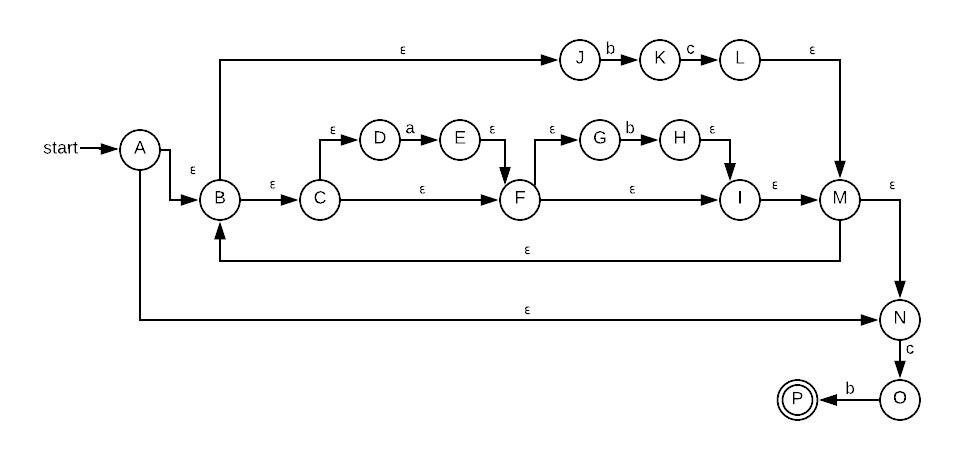
-2018

Регулярное выражение: ((a?b?)|(bc))\*cb

Задание:

По заданному регулярному выражению  
• Определить язык (регулярное множество), порождаемый регулярным  
выражением  
• Построить НКА  
• По полученному НКА построить ДКА  
• Минимизировать полученный ДКА  
• Для мин. ДКА написать программу-распознаватель предложений  
языка, порождаемого регулярным выражением

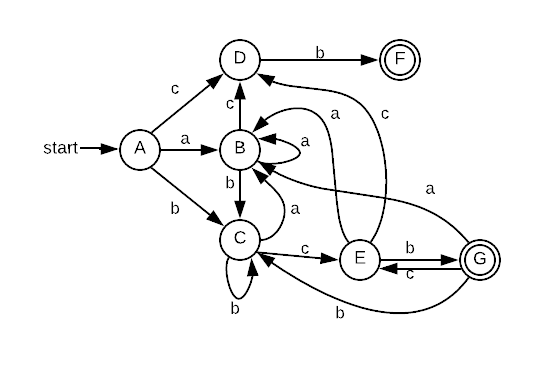
Регулярное множество: { Ancb | A ∈ {a, b, ab, bc}, n ≥ 0 }

НКА:

Преобразование в ДКА (терминалы обозначены красным)

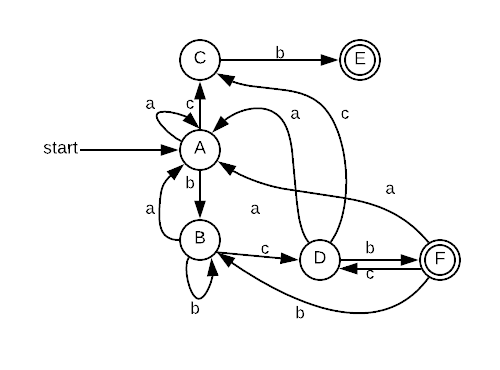
|  |  |  |  |
| --- | --- | --- | --- |
| **State** | **a** | **b** | **c** |
| A’ = {A,B,C,D,F,  G,I,M,N,J} | B’ = {E,F,G,I,M,  N,B,C,D,J} | C’ = {H,I,M,N,B,  C,D,F,G,J,K} | D’ = {O} |
| B’ | B’ | C’ | D’ |
| C’ | B’ | C’ | E’ = {O,L,M,B,J,  C,D,F,G,I,N} |
| D’ | - | F’ = {P} | - |
| E’ | B’ | G’ = {P, C’} | D’ |
| F’ | - | - | - |
| G’ | B’ | C’ | E’ |

Полученный ДКА:



Для минимизации объединяем состояния А и В, которые имеют одинаковые таблицы переходов.

Минимизированный ДКА:



Программа-распознаватель (Python):

class State:

transitions = {}

def \_\_init\_\_(self, transitions):

self.transitions = transitions

class InitState(State):

pass

class TerminalState(State):

pass

class FSM:

states = {}

initState = InitState({})

def \_\_init\_\_(self, states):

self.states = states

state\_values = dict.values(states)

initStates = [s for s in state\_values if isinstance(s, InitState)]

terminalStates = [s for s in state\_values if isinstance(s, TerminalState)]

assert len(initStates) == 1

assert any(terminalStates)

self.initState = initStates[0]

def doTransition(self, state, char):

if state is not None:

next\_state = state.transitions.get(char)

return self.states.get(next\_state)

else:

return None

def parse(self, sentence):

state = reduce(self.doTransition, sentence, self.initState)

return isinstance(state, TerminalState) if state is not None else False

fsm = FSM({

'A': InitState({'a': 'A', 'b': 'B', 'c': 'D'}),

'B': State({'a': 'A', 'b': 'B', 'c': 'C'}),

'C': State({'a': 'A', 'b': 'F', 'c': 'D'}),

'D': State({'b': 'E'}),

'E': TerminalState({}),

'F': TerminalState({'a': 'A', 'b': 'B', 'c': 'C'})

})

Модульные тесты:

assert fsm.parse("cb")

assert fsm.parse("bccb")

assert fsm.parse("bcbccb")

assert fsm.parse("acb")

assert fsm.parse("bcb")

assert fsm.parse("abcb")

assert fsm.parse("aabcb")

assert fsm.parse("bbbcb")

assert fsm.parse("aaaaaaababababbcbcabababcbbbbbcb")

assert not fsm.parse("c")

assert not fsm.parse("bcccb")

assert not fsm.parse("whatever")