Санкт-Петербургский государственный университет

Факультет прикладной математики – процессов управления

Реализация магазинного автомата для контекстно-свободного языка

Работу выполнил Панюшин Даниил Васильевич группа 19.Б12-пу

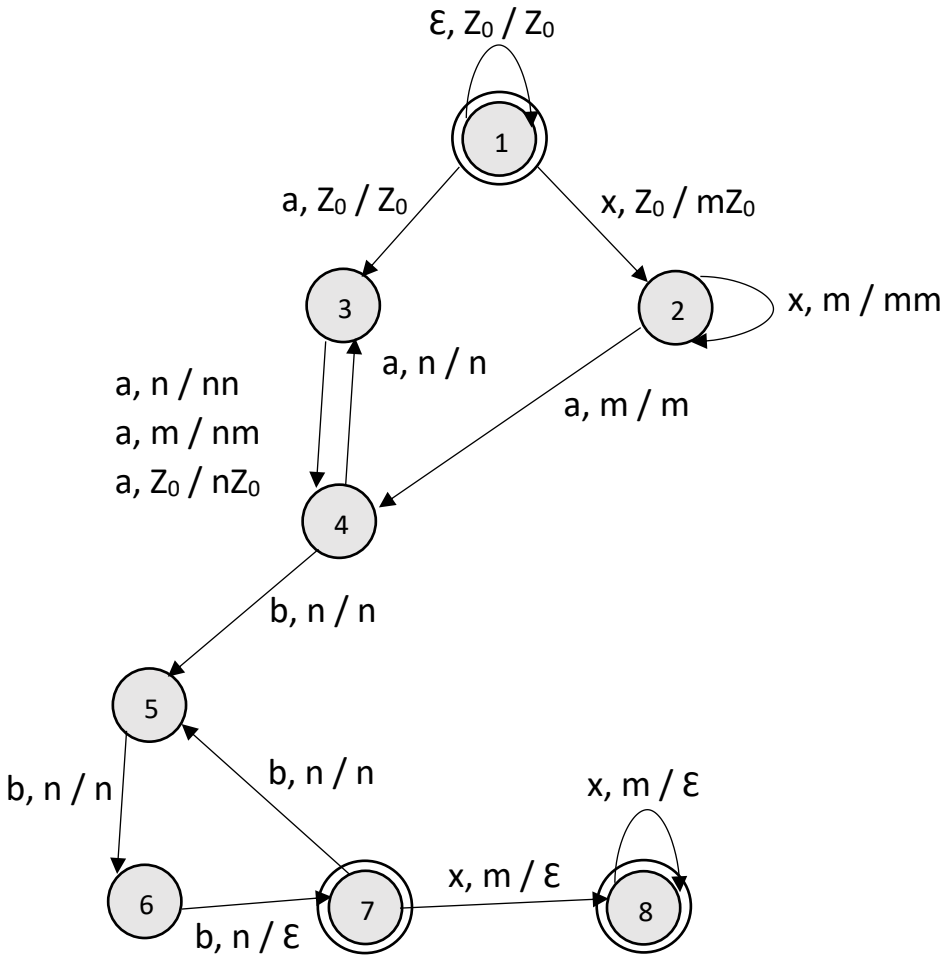
Формулировка задания:

1. Построить порождающую грамматику для заданного КС языка.
2. Разработать распознающий МП автомат для цепочек определенного в задании КС языка над заданным алфавитом и реализовать его в виде подпрограммы.
3. Разработать программу, распознающую цепочки этого языка на основе программной реализации МП автомата.

Порождающая КС грамматика:

- S → xSx | A | xx | ε
- A → aaAbbb | aabbb

МП – автомат (по завершающему состоянию):



Формальное задание МП автомата:

- $\delta(1, \varepsilon, Z_0) = (1, Z_0)$
- $\delta(1, x, Z_0) = (2, mZ_0)$
- $\delta(2, x, m) = (2, mm)$
- $\delta(1, a, Z_0) = (3, Z_0)$

- $\delta(3, a, Z_0) = (4, nZ_0)$
- $\delta(3, a, n) = (4, nn)$
- $\delta(3, a, m) = (4, nm)$
- $\delta(4, a, n) = (3, n)$
- $\delta(4, b, n) = (4, n)$
- $\delta(5, b, n) = (6, n)$
- $\delta(6, b, n) = (7, \varepsilon)$
- $\delta(7, b, n) = (5, n)$
- $\delta(7, x, m) = (8, \varepsilon)$
- $\delta(8, x, m) = (8, \varepsilon)$

Код программы:

```
class Automat():
    # таблца состояний (мертвое состояние всегда последнее)
    states = [
        ('a', 'b', 'x'),
        {'x':(2, ('',), 'm'), 'a': (3, ('',), ''), 'b': (9, ('',), '')},
        {'x':(2, ('m',), 'm'), 'a': (3, ('m',), ''), 'b': (9, ('m',), '')},
        {'x':(9, ('m','n',''), ''), 'a': (4, ('m','n',''), 'n'), 'b': (9, ('m','n',''), '')},
        {'x':(9, ('m','n',''), ''), 'a': (3, ('n',), ''), 'b': (5, ('n',), '')},
        {'x':(9, ('n',), ''), 'a': (9, ('n',), ''), 'b': (6, ('n',), '')},
        {'x':(9, ('n',), ''), 'a': (9, ('n',), ''), 'b': (7, ('n',), 'd')},
        {'x':(8, ('m',), 'd'), 'a': (9, ('m','n'), ''), 'b': (5, ('n',), '')},
        {'x':(9, ('m',), 'd'), 'a': (9, ('m',), ''), 'b': (9, ('m',), '')},
        {'x':(9, ('n','m',''), 'd'), 'a': (9, ('n','m',''), 'd'), 'b': (9, ('n','m',''), 'd')},
    ]
    stack = ''  # начальный символ

    def __init__(self, final_states, current_state=1, states=None):  # конструктор
        self.final_states = final_states  # список финальных состояниц
        self.current_state = current_state  # текущее состояние
        if states is not None:
            self.set_states(states)

    def set_states(self, states):  # сеттер таблицы состояний
        self.states = states

    def add(self, string):  # добавить в конец стека символ
        self.stack = self.stack + string
    def pop(self):  # извлечь из стека последний символ
        self.stack = self.stack[0:-1]
    def check(self):  # посмотреть последниций символ в стеке
        if self.stack == '':
            return ''
        return self.stack[-1]

    def read(self, char):  # переход из одного состояний в другое основываясь на считанном символе и верхнем эементе стека
        if char not in self.states[0]:  # если символ не в алфавите переходим в мёртвое состояние
            self.current_state = self.states[-1][self.states[0][0]][0]
        state = self.states[self.current_state][char]
        if self.check() in state[1]:  # смотрим, что вверху стека
            if state[2]=='d':  # символ d означает удаление верхнего символа стека
                self.pop()
            else:  # иначе вносим символы в стек, основываясь на данных их таблицы состояний
                self.add(state[2])
            self.current_state = state[0]  # обновляем текущее состояние
        else:  # если нет совпадения верхнего символа стека с предполагаемыми на этом шаге
            self.current_state = self.states[-1][self.states[0][0]][0]  # обновляем текущее состояние на мёртвое
        return self.current_state  # возвращаем состояние, в которое перешли

    def recognize(self, string):  # распознавание цепочки, допустимой автоматом
        for char in string:  # проходим циклом по символам цепочки
            self.read(char)  # применяем функцию перехода к каждому символу строки
        # если текущее состояние после работы автомата - одно из допустимых и стек пуст, то цепочка допустима
        if self.current_state in self.final_states and self.stack=='':
            return True
        return False
```

Тесты:

```
In [144]: automat.recognize('')
Out[144]: True

In [145]: automat.recognize('xaabbbx')
Out[145]: True

In [146]: automat.recognize('xaabbb')
          Ошибка: в позиции 6 не хватает символа!
Out[146]: False

In [147]: automat.recognize('aabbbx')
          Ошибка: в позиции 5 неожидаемый символ x!
Out[147]: False

In [148]: automat.recognize('xxaabbbx')
          Ошибка: в позиции 8 не хватает символа!
Out[148]: False

In [149]: automat.recognize('xaabbbxx')
          Ошибка: в позиции 7 неожидаемый символ x!
Out[149]: False

In [150]: automat.recognize('aabbb')
Out[150]: True

In [151]: automat.recognize('aaabbb')
          Ошибка: в позиции 3 неожидаемый символ b!
Out[151]: False

In [152]: automat.recognize('aabbbb')
          Ошибка: в позиции 5 неожидаемый символ b!
Out[152]: False

In [153]: automat.recognize('xxxaabbbaaaxxx')
          Ошибка: в позиции 8 неожидаемый символ a!
Out[153]: False
```