Санкт-Петербургский государственный университет

Факультет прикладной математики – процессов управления

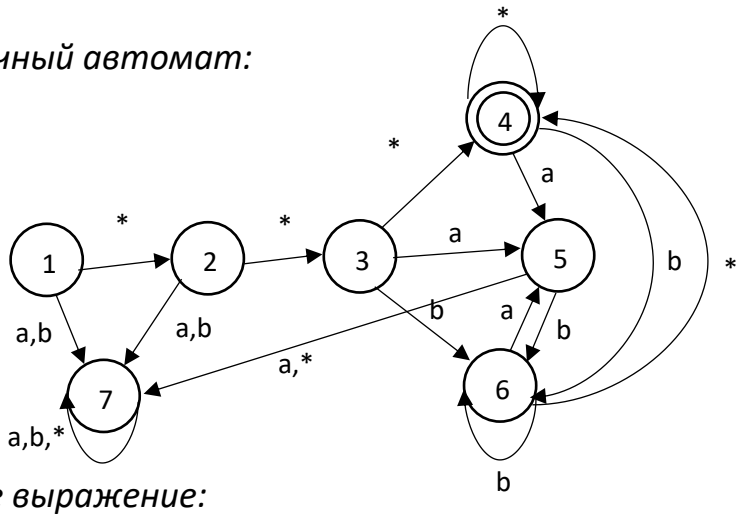
# Реализация конечного автомата для регулярного языка

Работу выполнил Панюшин Даниил Васильевич группа 19.Б12-пу

*Формулировка задания:*

Строка символов a, b, *,
начинающаяся с префикса ** и заканчивающаяся суффиксом *,
между которыми располагается последовательность символов a,b,
в которой после каждого символа a следует символ b (пример:*** или **bbabb*
или **ababbbab*).

*Конечный автомат:*




| | a | b | * |
|---|---|---|---|
| 1 | 7 | 7 | 2 |
| 2 | 7 | 7 | 3 |
| 3 | 5 | 6 | 4 |
| 4 | 5 | 6 | 4 |
| 5 | 7 | 6 | 7 |
| 6 | 5 | 6 | 4 |
| 7 | 7 | 7 | 7 |

*Регулярное выражение:*

**(**(b+ab+b)**)***

*Код программы (реализация без автомата):*

```
def recognize(string=None):
    if string is None: # пустая строка нам не подходит
        return False
    if string[0:2] != '**' or string[len(string) - 1] != '*': # проверяем начало и конец строки
        return False
    for char in string: # проверяем на наличие в строке только символов алфавита
        if char not in ('*', 'a', 'b',):
            return False
    if string != '***': # если строка не равна *** (что сразу подходит условию) происходит проверка
        previous = string[2] # запоминаем предыдущий символ (начиная с символа с индексои 2)
        for char in string[3:len(string) - 1]: # проходим по подстроке, которая не содержит начальных ** и конечной *
            if previous == 'a' and char != 'b': # если после a идёт не b, то такафя строка нам не подходит
                return False
            previous = char # обновляем предыдущий символ
    return True

def search(string):
    result = {}
    for i in range(0, len(string)): # проходим по символам строки
        if len(string[i:len(string)]) > 2 and string[i:i + 2] == '**': # если длина строки > 2 и первые 2 символа=**
            # индекс первого (после **) вхождения * в подстроку, начинающуюся с i-того символа входной строки
            j = string[i + 2:len(string)].find('*') + i + 2
            if recognize(string[i:j + 1]): # применяем распознающую функцию
                result[i] = string[i:j + 1] # если строка подходит, добавляем её в словарь
        # если в конце подстроки идут **, то это может быть началом новой подходящей подстроки=> двигаем соотвествующе маркер i
            if (string[j - 1] == '*'):
                i=j - 1
            else:
                i = j
    return result # возвращаем распознаныне строки
```

*Тесты:*

```
print(search('***'))
```

```
{0: '***'}
```

```
print(search('******'))
```

```
{0: '***', 1: '***', 2: '***', 3: '***'}
```

```
print(search('**ab*'))
```

```
{0: '**ab*'}
```

```
print(search('*ab*'))
```

```
{}
```

```
print(search('*ab**ab*'))
```

```
{3: '**ab*'}
```

```
print(search('**aaaabbbb*'))
```

```
{}
```

```
print(search('**hjhj*'))
```

```
{}
```

```
print(search('**ahab*'))
```

```
{}
```

```
print(search('**abghjghj*'))
```

```
{}
```

```
print(search('*****bbbbababababbbb***abababababbb*****bbbbbbbbabababbbbabbbbabbb****'))
```

```
{0: '***', 1: '***', 2: '***', 3: '**bbbbabababbbbb*', 19: '***', 20: '**abababababbb*', 33: '***', 34: '***', 35: '***', 36: '*
*bbbbbbbbabababbbbabbbbabbb*', 63: '***', 64: '***'}
```

*Код программы (реализация через автомат):*

```
class Position():
    # таблица состояний (первая строка - алфавит)
    states = [('a','b','*',),
              {'a':7, 'b':7, '*':2},
              {'a':7, 'b':7, '*':3},
              {'a':5, 'b':6, '*':4},
              {'a':5, 'b':6, '*':4},
              {'a':7, 'b':6, '*':7},
              {'a':5, 'b':6, '*':4},
              {'a':7, 'b':7, '*':7},
             ]
    recognized_strings = {} # строки, допустимые автоматом
    state = 1 # начальное состояние

    def __init__(self, final, states=None):
        self.final_state = final
        if states is not None:
            self.states = self.states

    # метод, реализующий смену состояния автомата при считывании символа
    def read(self, char=None):
        if char is None: # по пустому символу никуда не переходим
            return
        if char not in self.states[0]: # по символу не из алфавита никуда не переходим
            return
        sym = self.states[self.state][char] # переход в нужное состаяние реализован чеерез обращение к списку словарей
        if sym is None: # если симво None то остаёмся в прежнем состоянии
            return
        self.state = sym

    # метод, распознающий строки
    def recognize(self, string=None):
        if string is None: # пустая строка нам не подходит
            return False
        for char in string: # проходим по символам строки, применяя на нах функцию read(char)
            self.read(char)
        if self.state == self.final_state: # если текущее состояние равно финальному, то строка подходит
            return True
        return False

    # метод поиска допустимых подстрок во входной строке
    def search(self, string):
        self.recognized_strings={} # словарь найденных строк (ключ-номер первого символа подстроки)
        for i in range(0, len(string)): # проходим по символам строки
            if len(string[i:len(string)]) > 2 and string[i:i + 2] == '**': # если длина строки > 2 и первые 2 символа=**
                # индекс первого (после **) вхождения * в подстроку, начинающуюся с i-того символа входной строки
                j = string[i + 2:len(string)].find('*') + i + 2
                if self.recognize(string[i:j + 1]): # применяем распознающую функцию
                    self.recognized_strings[i] = string[i:j + 1] # если строка подходит, добавляем её в словарь
    # если в конце подстроки идут **, то это может быть началом новой подходящей подстроки=> двигаем соотвествующе маркер i
                if (string[j - 1] == '*'):
                    i=j - 1
                else:
                    i = j
        return self.recognized_strings # возвращаем распознаныне строки
```

*Результаты тестов идентичны реализации без использования абстракции автомата.*