Документация и теоретическое обоснование кода детектора подстроки abba

Супрун Артём Сергеевич

14 апреля 2025 г.

Аннотация

Данный документ демонстрирует применение библиотеки transitions для создания детектора подстроки abba с визуализацией работы конечного автомата посредством GraphMachine. Документация включает краткий обзор теории детерминированных конечных автоматов, описание логики переходов и подробное пояснение работы исходного кода на Python.

Содержание

1 Введение

Конечные автоматы являются важным инструментом в теории вычислений, их широко применяют в лексическом анализе, распознавании текстовых шаблонов и в реализации протоколов связи. В данном примере с помощью библиотеки transitions создаётся автомат, проходящий по входной строке для определения наличия подстроки abba. Для удобства анализа работы автомата используется GraphMachine, который генерирует диаграмму состояний и переходов.

2 Теоретическая основа

2.1 Детерминированный конечный автомат (ДКА)

Детерминированный конечный автомат (ДКА) описывается кортежем:

$$(Q, \Sigma, \delta, q_0, F)$$

где:

- Q — конечное множество состояний;
- Σ — алфавит входных символов;
- $\delta : Q \times \Sigma \to Q$ — функция перехода, задающая правило смены состояний;
- $q_0 \in Q$ — начальное состояние;
- $F \subseteq Q$ — множество принимающих (финальных) состояний.

Автомат последовательно считывает символы строки, переходя из одного состояния в другое согласно функции δ. Если после обработки входных символов автомат оказывается в одном из состояний F, строка считается распознанной. В рассматриваемом примере подстрока abba обнаруживается при достижении состояния q4.

2.2 Библиотека transitions

Пакет transitions позволяет декларативно описывать конечные автоматы, задавая список состояний, переходов и условия для каждого перехода. Расширение GraphMachine, в частности, обеспечивает автоматическую генерацию визуальной схемы автомата, что значительно облегчает отладку и анализ работы системы.

3 Описание реализации

3.1 Структура кода и логика переходов

Код написан на Python. В нём определяется класс Detector, объект которого используется для обработки строки символ за символом с вызовом метода advance. При этом переходы между состояниями происходят согласно заданным правилам:

- Из начального состояния q0 при встрече символа 'a' происходит переход в состояние q1.
- В состоянии q1 переход в состояние q2 осуществляется при символе 'b', а повторное появление 'a' остаётся в q1.

- Переходы для состояний q2 и q3 аналогичным образом обеспечивают постепенное распознавание последовательности.
- Достижение состояния q4 означает, что подстрока abba обнаружена, и автомат остаётся в этом состоянии независимо от дальнейших символов.

3.2 Условия переходов

Для каждого перехода определены функции, анализирующие текущий символ:

- is_a — возвращает True, если символ равен 'a'.
- is_not_a — возвращает True, если символ не равен 'a'.
- is_b — возвращает True, если символ равен 'b'.
- not_a_or_b — возвращает True, если символ не является ни 'a', ни 'b'.

3.3 Исходный код детектора

Ниже приведён полный исходный код с подробными комментариями:

Листинг 1: Исходный код детектора подстроки abba

```
from transitions.extensions import GraphMachine

def is_a(**kwargs):
    """
                    ,                                    'a'.
    """
    return kwargs.get('char') == 'a'

def is_not_a(**kwargs):
    """
                    ,                                     'a'.
    """
    return kwargs.get('char') != 'a'

def is_b(**kwargs):
    """
                    ,                                    'b'.
    """
    return kwargs.get('char') == 'b'

def not_a_or_b(**kwargs):
    """
                        True,
                'a'                        'b'.
    """
    return kwargs.get('char') not in ['a', 'b']

class Detector:
    """
            -                                                'abba'.
    """
    pass

#
detector = Detector()
```

```
#
states = ['q0', 'q1', 'q2', 'q3', 'q4']

#                                          ,
                         'q0'
machine = GraphMachine(model=detector, states=states,
                       initial='q0', show_conditions=True)

#                                                    :
# (                            ,                 ,
                  ,                 )
transitions = [
    ('q0', 'advance', 'q1', is_a),        #          'a': q0 -> q1
    ('q0', 'advance', 'q0', is_not_a),    #
        q0

    ('q1', 'advance', 'q2', is_b),        #       q1: 'b'
          q2
    ('q1', 'advance', 'q1', is_a),        # 'a'                       q1 (
                                   )
    ('q1', 'advance', 'q0', not_a_or_b),  #
                                            q0

    ('q2', 'advance', 'q3', is_b),        #
                                   'b': q2 -> q3
    ('q2', 'advance', 'q1', is_a),        # 'a'

    ('q2', 'advance', 'q0', not_a_or_b),  #
        ---                 q0

    ('q3', 'advance', 'q4', is_a),        #                           'a':
                        'abba'            (q3 -> q4)
    ('q3', 'advance', 'q0', is_not_a),    #                            q0

    ('q4', 'advance', 'q4', True)         #

]

#
for source, trigger, dest, cond in transitions:
    machine.add_transition(trigger, source, dest, conditions=cond)

if __name__ == "__main__":
    #                                                                     ,
        abba'
    test_string = "xxabba123"
    print(f"                          : {test_string}")

    #
    for ch in test_string:
        detector.advance(char=ch)
        if detector.state == 'q4':
            print(f"              '{test_string}'
                             'abba'.")
            break

    #
```

```
    graph = detector.get_graph()
    graph.layout(prog="dot")
    graph.draw("state_diagram.png")
    print("
        'state_diagram.png'.")
```

4 Заключение

Представленный пример демонстрирует, как можно с помощью библиотеки transitions реализовать конечный автомат для поиска конкретного паттерна (подстроки abba) в строке. Тщательно продуманная схема переходов и последующая визуализация диаграммы помогают глубже понять логику работы автомата. Можно расширить данный подход для более сложных алгоритмов распознавания, обработки ошибок и динамического управления состояниями.

5 Дополнительное чтение

Рекомендуем для дальнейшего углубления ознакомиться с:

ions/transitions Документацией библиотеки transitions.

- Литературой по теории конечных автоматов и реализации алгоритмов распознавания.
- Инструментами визуализации графовых структур, например, Graphviz.