NFA-DFA 변환기 구현

2020112082 오현석

형식언어

송수환 교수님

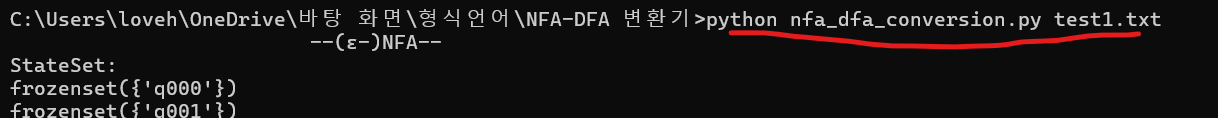
2024.05.19

**1. 실행 방법 (윈도우 기준)**

a) 실행 코드와 테스트 파일이 든 폴더를 압축 해제

b) 해당 폴더에서 cmd실행

c) python nfa\_dfa\_conversion.py test1.txt 명령어로 실행 (테스트 케이스는 1부터 10까지 존재)



**2. 중요 코드**

변환기에서 중요한 코드는 nfa에서 dfa로 변환하는 함수(a)와 그 중 epsilon-closure을 구하는 함수(b), 그리고 dfa를 최소화시켜주는 함수(c)이다.

(a) dfa\_conversion 함수

파일에서 읽어온 NFA 정보를 이용하여 DFA로 변환하는 함수이다. 스택을 사용하고 있으며 stack을 startstate의 epsilon-closure로 초기화한다. while문을 통해 스택의 상태들을 가져와 해당 상태가 갈 수 있는 모든 상태를 고려하며 이것을 하나의 상태로 분류한다. epsilon-closure까지 합집합으로 합쳐 하나의 새로운 DFA 상태를 만든다. 마지막으로 final\_state가 포함된 새로운 dfa상태들을 dfa\_final\_state로 저장하여 새로운 final\_state를 만든다.

   def dfa\_conversion(self):

        dfa\_states = []  # DFA 상태 저장 리스트

        dfa\_delta = {}   # DFA 함수 저장 딕셔너리

        dfa\_start\_state = self.epsilon\_closure(self.StartState)  # 시작 상태의 입실론-closure를 구함

        # print(frozenset(dfa\_start\_state))

        stack = [frozenset(dfa\_start\_state)]  # 스택으로 DFA 상태를 구함, 시작 상태로 초기화

        # print(stack)

        while stack:

            current\_dfa\_state = stack.pop()  # 스택에서 DFA 상태를 pop

            if current\_dfa\_state not in dfa\_states:

                dfa\_states.append(current\_dfa\_state)  # dfa\_states리스트에 저장

                # 모든 가능한 DFA 상태를 탐색

                for symbol in self.TerminalSet:

                    next\_dfa\_state = set()

                    # 입실론-closure까지 고려하여 상태를 저장

                    for nfa\_state in current\_dfa\_state:#현재 상태에서 갈 수 있는 모든 상태

                        transitions = self.DeltaFunctions.get((frozenset({nfa\_state}), symbol), set())

                        for transition\_state in transitions:#그 중 입실론-closure까지 고려하여 합집합

                            next\_dfa\_state |= self.epsilon\_closure(transition\_state)

                    if next\_dfa\_state:

                        next\_dfa\_state = frozenset(next\_dfa\_state)

                        if next\_dfa\_state not in dfa\_states:

                            stack.append(next\_dfa\_state)  #이렇게 만들어진 새로운 상태가 dfa\_state리스트에 없다면 스택에 저장하고 다시 loop

                        dfa\_delta[(current\_dfa\_state, symbol)] = next\_dfa\_state  # DFA 상태 전이 함수 업데이트

        # dfa\_state에서 finalState를 포함한 상태를 dfa\_final\_state로 저장

        dfa\_final\_states = [state for state in dfa\_states if any(s in self.FinalState for s in state)]

(b) epsilon\_closure 함수

특정 상태의 epsilon-closure을 구하는 함수이다. 스택을 사용하며 while문으로 반복한다. 현재 상태를 스택에서 받아와 현재 상태가 epsilon을 보고 다음 상태가 존재한다면 스택에 넣고 반복하여 epsilon-closure을 구할 수 있다.

    def epsilon\_closure(self, state):

            visited = set()  # 방문한 상태를 기록

            stack = [state]  # 스택으로 입실론-closure를 찾아냄

            while stack:

                current\_state = stack.pop()  #스택에서 상태를 받아옴

                if current\_state not in visited:

                    visited.add(current\_state)  #해당 상태를 visited로

                    #입실론으로 갈 수 있는 모든 상태 탐색

                    epsilon\_transitions = self.DeltaFunctions.get((frozenset({current\_state}), 'ε'), set())

                    for next\_state in epsilon\_transitions:

                        if next\_state not in visited:

                            stack.append(next\_state)  # 해당 상태가 visited되지 않았다면 stack에 추가 후 다시 loop

            return visited

(c) reduced\_dfa 함수

DFA의 상태수를 최소화하는 함수이다. 처음에는 리스트 P에 상태를 finalstate와 그렇지 않은 상태들로 나누었다. P는 새로운 최소화된 DFA 상태를 저장할 리스트이다. 이때 다른 리스트 stack에 같은 집합 내용을 저장하였다. 이는 반복문을 통해 상태들을 합치고 나누어 최소화하기 위함이다.

이렇게 초기화된 stack에서 A를 꺼내와 각 심볼에 대한 상태들의 값이 A에 존재한다면 즉, A(새로운 상태 집합)에 도달할 수 있는 상태들을 X 집합에 저장한다. P안의 집합을 Y로 순회하며 X와 Y의 교집합과 차집합을 구한다. X는 A에 도달할 수 있는 상태들이고 Y는 P의 집합이므로 둘의 교집합은 같은 상태집합이 되고 둘의 차집합은 다른 상태집합이 된다. 교집합과 차집합이 둘 다 존재한다면 서로 다른 두 집합이 생기는 것으로 P와 stack에 저장한다. 이렇게 계속 반복한다면 결론적으로 P에 최소화된 상태 집합 리스트가 저장된다. 이를 이용해 시작상태와 종료상태 그리고 델타 함수를 구할 수 있다.

def reduced\_dfa(self):

        #상태를 finalstate와 그렇지 않은 상태로 나누기

        P = [set(self.FinalState), set(self.StateSet) - set(self.FinalState)]#실제 최소화된 dfa상태가 저장될 리스트

        stack = [set(self.FinalState), set(self.StateSet) - set(self.FinalState)]#

        while stack:

            A = stack.pop()

            for symbol in self.TerminalSet:

                X = {state for state in self.StateSet if self.DeltaFunctions.get((state, symbol)) in A}# A(새로운 상태 집합) 에 도달할 수 있는 state들

                for Y in P.copy():#P의 세트 Y와 X의 차집합, 교집합을 이용해 그룹 분리

                    intersection = X & Y

                    difference = Y - X

                    if intersection and difference:#차집합과 교집합이 둘 다 존재 -> 서로 다른 두 그룹

                        P.remove(Y)

                        P.append(intersection)

                        P.append(difference)

                        if Y in stack:#계속해서 loop를 하기 위해 stack에도 차집합, 교집합 추가

                            stack.remove(Y)

                            stack.append(intersection)

                            stack.append(difference)

                        else:

                            if len(intersection) <= len(difference):

                                stack.append(intersection)

                            else:

                                stack.append(difference)

        new\_states = [tuple(s) for s in P]#minimized dfa states

        new\_start\_state = next(s for s in new\_states if self.StartState in s)#시작 상태

        new\_final\_states = [s for s in new\_states if any(fs in s for fs in self.FinalState)]#종료 상태

        new\_delta = {}

        #delta\_function 구하기

        for state in new\_states:

            tmp = next(iter(state))#minimized dfa 상태의 첫 번째 원소(대표 상태)

            for symbol in self.TerminalSet:#그 원소가 symbol을 바라봤을 때의 값

                next\_state = self.DeltaFunctions.get((tmp, symbol))

                if next\_state:

                    value = [s for s in new\_states if next\_state in s]#그 값이 minimized dfa 상태 중 어떤 상태에 속하는지 확인 후 해당되는 상태를 value에 저장

                    new\_delta[(state, symbol)] = value #minimized dfa function

**4. 실험 및 검증**

1) test1.txt

텍스트, 스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

(빨간 선이 하나의 그룹이다. 한번에 인식이 힘들어 추가하였다.)

2)test2.txt

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

3) test3.txt

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

4) test4.txt

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

5) test5.txt

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

6) test6.txt

텍스트, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

검증

텍스트, 스크린샷, 폰트, 도표이(가) 표시된 사진

자동 생성된 설명

텍스트, 폰트, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 도표, 번호이(가) 표시된 사진

자동 생성된 설명

**텍스트, 도표, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명**

7) test7.txt

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

8) test8.txt

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

9)test9.txt

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

10) test10.txt

텍스트, 스크린샷, 소프트웨어, 폰트이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

**5. 소감**