오토마타 이론 HW1 보고서

2016-11464 정재훈

1. 사용한 파일들
1) java files(.java)
-Main.java
입출력과 여러 가지 함수를 사용하여 NFA를 설계하고 string을 돌려보는, 프로그램을 실행하는 주 무대가 되는 객체이다.
-state.java
NFA 설계에 사용되는 각 상태를 나타내는 객체로, 상태들끼리의 transition관련 정보들이 저장되어 있다.

2) java library
java.util.* : HashMap, Map, ArrayList를 사용하기 위해서 import하였다.
java.io.* : 파일 입출력을 하기 위해서 import하였다.

2. 구현에 관한 설명
이 프로그램은 크게 세 가지 과정으로 구현되어있다.

먼저 들어온 정규식을 (와)등의 기호들이 없는 postfix expression으로 바꿔서 string 변수에 저장한다. 다음으로, 이 postfix expression을 사용하여 NFA를 앞에서부터 기호에 따라 차례차례 건설해나간다. (이 과정이 모두 끝나면 최종적으로 큰 NFA 하나가 남는다.) 마지막으로, 이 NFA에 주어진 string을 넣어서 가능한 모든 상태 중에 NFA의 최종 상태가 포함되는지를 확인하여 yes 또는 no를 출력한다.

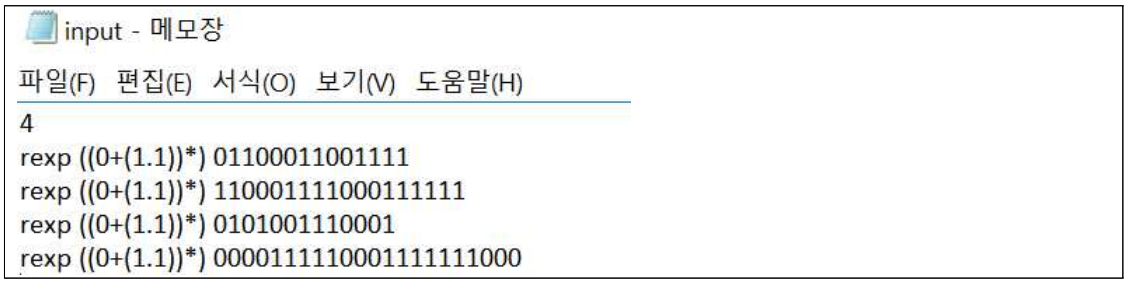
3. 자세한 동작 원리
3-1. 정규식 > postfix expression
숫자와 기호를 구분해서 받고 각각 stack에 저장한 뒤, 닫는 괄호())가 들어오면 pop해서 숫자와 기호를 postfix 형태로 붙인 후 다시 stack에 저장한다. 이를 반복하면 postfix expression이 숫자 stack에 마지막으로 남은 원소가 된다.
3-2. postfix expression > NFA
<오토마타 이론> 24, 25쪽을 참고하면, NFA들이 각 기호에 의해서 어떻게 합쳐지는지 나와 있는데, 이를 이용하여 구현하였다. 이 과정에서 각 NFA의 초기 state와 최종 state를 저장할 필요가 있으므로 이들을 stack으로 만들어 기호를 읽으면 pop해서 사용할 수 있게 하였다. (숫자를 읽으면 두 state를 만들어 한 state에서 다른 state로 그 숫자로 connect되게 하고, +를 읽으면 새로운 state 두 개 (a,b) 를 만들어 a state에서 연산할 NFA 두 개의 초기 state에 epsilon으로 연결하고, 연산할 NFA 두 개의 최종 state에서 b state에 epsilon으로 연결하는 등...)

3-3 NFA + string > boolean value

해당 string이 NFA를 통과하는지 보는 데는 두 가지 과정을 반복적으로 사용하였다. 먼저, epsilon transition으로 갈 수 있는 모든 곳을 체크하고, 다음으로, 거기에서 string의 특정 문자로 갈 수 있는(transition 가능한) 곳을 체크한다. 이를 string의 모든 문자에 대하여 반복하고 마지막으로 epsilon transition으로 갈 수 있는 곳을 체크하면, string이 가질 수 있는 모든 state의 집합을 얻는다. 이것이 미리 저장해둔 NFA의 최종 상태를 포함하는지 확인하여 yes 또는 no를 출력하였다. 한 번 체크한 곳은 ArrayList에 저장하여 contains를 통해 다시 확인되는 것을 막았다.

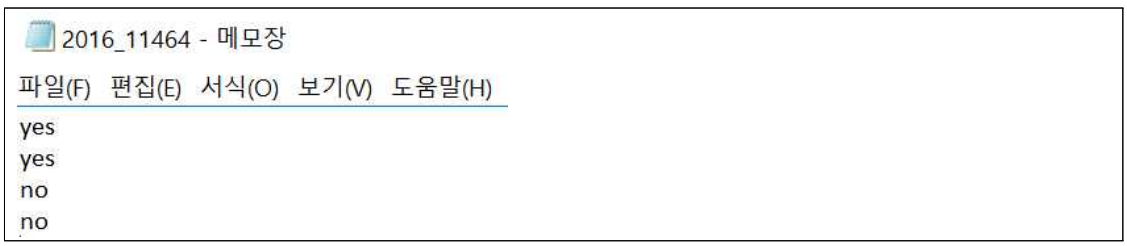
4. 입출력 예시

<입력 1>



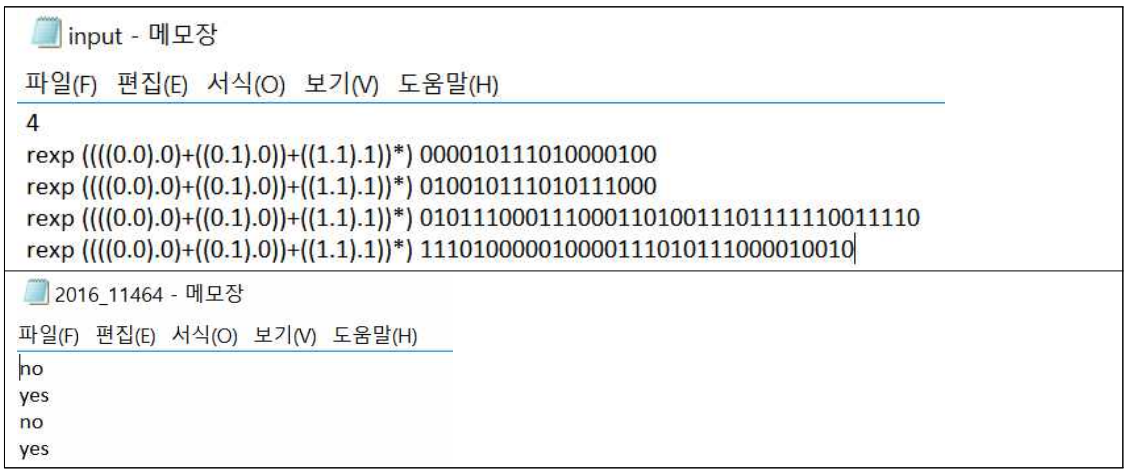
(0+11)* , 0또는 11로만 이루어진 정규식을 생각해보자. 위의 두 개는 연속한 1이 짝수 개이므로 yes, 밑의 두 개는 '111'과 '11111'을 포함하므로 no를 출력해야 한다.

<출력 1>



잘 실행됨을 알 수 있었다.

<입력 2 & 출력 2>



좀 더 복잡한 식에 관해서 확인해보아도 정확한 결과가 출력됨을 알 수 있었다.