본 프로젝트 2 정규식 to m-DFA 변환기

제 10주 목요일 (11/03) 출제 제13주 수요일 (11/23)까지 제출

(1) lex와 yacc[1]을 이용하여 정규식을 **파싱**(parsing)하며 Abstract Syntax Tree(AST)[2]를 만든다.
(2) AST를 지나며(traverse) ε-NFA를 만든다.
(3) 예비프로젝트 2.1의 ε-NFA to m-DFA 변환기 결과를 이용하여, m-DFA로 변환한다.

제출 시 유의사항

작성할 프로그래밍 언어는 다음으로 제한한다. **C, C++, Python, Java**. 코드와 함께, 작성한 프로그램을 실행시키기 위한 **환경, 컴파일 및 실행 방법**을 명시하여 **README**를 첨부해야 한다. 프로그램의 자세한 세부사항은 아래 제출파일 및 입출력 예시를 참고하라.

지정한 파서 혹은 파서 제너레이터를 사용하고, 각각 입력/출력/사용방법을 반드시 함께 첨부하라.

- C/C++의 경우
  - lex/yacc 혹은 flex/bison을 사용하라.
  - 각 소스코드 (*.lex, *.y)과 출력파일 (lex.yy.c, y.tab.*)을 첨부하라.
  - 각 파서 제너레이터를 실행한 방법을 **README**에 명시하라.

- Python의 경우
  - PLY를 사용하라.
  - 출력파일 (parsetab.py)을 첨부하라.

- Java의 경우
  - ANTLR4를 사용하라.
  - 각 소스코드 (*.g4)과 출력파일 (*.java)을 첨부하라.
  - 각 파서 제너레이터를 실행한 방법을 **README**에 명시하라.

---

1) lex와 yacc은 UNIX에서 제공된다. 비슷한 일을 하는 flex, bison등을 사용하여도 좋다.
2) AST는 parse tree의 부분집합이다.

입력 방식 및 입출력 테스트 케이스 설명

0. Readme

아래와 같이 정규식 입력 방식과 input 문자열 입력 방식을 제한합니다. 다음 포맷을 지키지 않을 경우에 불이익이 있을 수 있습니다. 첨부한 're.txt' 및 'm-dfa.txt' 파일은 프로젝트 수행에 있어서 자신의 코드를 테스트하기 위한 간단한 test case로 사용할 수 있습니다.

1. ε-NFA 입력 방식

ε-NFA를 입력하는 방식은 're.txt' 파일을 읽어 결정합니다.

re.txt	

ba*(a+b)	// 정규식 한 줄을 입력받음.
	// 주의: 입력 문자는 a, b로 제한되지 않습니다.
	// 정규식 내 입력 문자의 종류를 보고
	// m-DFA의 input symbol 목록을 정하십시오.

2. m-DFA 출력 방식

m-DFA로의 변환 결과는 다음과 같이 'm-dfa.txt' 텍스트 파일 형식으로 출력하여야 합니다. 예비프로젝트 1-1에서의 DFA 입력 방식과 유사합니다.

m-dfa.txt	
	// DFA가 $M_D = (Q_D, \Sigma, \delta_D, q_{0D}, F_D)$로 표현될 때
State	// Q는 **상태**(State)를 의미
q0,q1,q2,q3,q4	// 각 상태 q0,q3 ... 는 ","로 구분됨.
Input symbol	// Σ는 **입력문자**(input symbol)를 의미
a,b	// 각 입력문자 a,b는 ","로 구분됨.
State transition function	// δ는 **상태변화함수**를 의미
q0,a,q4	// 현 상태 (q0)에서 입력문자(b)를 보고 다음 상태(q1)을
q0,b,q1	의미하며 ,로 구분됨
q1,a,q3	
q1,b,q2	
q2,a,q4	
q2,b,q4	
q3,a,q3	
q3,b,q2	
q4,a,q4	
q4,b,q4	
Initial state	// $q_0 \in Q$는 **처음상태**를 의미
q0	
Final state	// $F \subseteq Q$는 **끝나는 상태**를 의미
q2,q3	// 다수의 끝나는 상태가 존재하면 ","로 구분