형식언어 및 오토마타 프로젝트 2

개요

지난 보조 프로젝트 2-1에 이어, 정규 표현식을 m-DFA로 변환하는 프로그램을 작성하였습니다. 프로젝트 2-1에서 작성한, e-NFA를 m-DFA로 변환하는 프로그램을 사용할 수 있습니다. 따라 이번 프로젝트는 정규표현식을 e-NFA로 변환하는 데에 초점을 맞추고 있습니다.

정규식에 대응되는 Abstract Syntax Tree(AST)를 만들고, 트리를 순회하여 e-NFA를 만듭니다. 작성 언어는 Python 3.5로, 외부 모듈인 Python Lex & Yacc(PLY)를 사용했습니다. Ply 모듈은 python-pip를 통해 설치할 수 있습니다.

정규언어 표현식은 아래의 Backus-Naur Form으로 나타낼 수 있습니다.

```
<e>     := ϵ            // null symbol(base case)
        ¦ (<e>)         // parenthesis (우선순위 높음)
        ¦ <e>*          // Kleene star
        ¦ <sym><e>      // concatenation
        ¦ <e>+<e>       // union (우선순위 낮음)

<sym>   := <digit>¦<alpha>

<digit> := 0¦1¦2¦…¦9

<alpha> := a¦b¦c¦…¦z¦A¦B¦C¦…¦Z
```

편의를 위해 알파벳과 숫자만을 입력으로 받는다고 가정했습니다. 구현의 편의를 위해 ply 처리과정에서 _(underscore) 문자를 epsilon 대신 사용하였습니다.

파일 구조

1. re_to_ast.py

주어진 정규 표현식을 파싱하여 abstract syntax tree를 생성하는 코드입니다. Ply의 lex와 yacc 을 사용했으며,

2. ast_to_eNFA.py

주어진 abstract syntax tree를 바탕으로 e-NFA를 만들어내는 코드입니다.

3. eNFA.py, DFA.py, mDFA_convert

예비 프로젝트 2-1에서 작성한 프로그램입니다. 각각 e-NFA 클래스, DFA 클래스, 그리고 DFA를 m-DFA로

변환하는 기능을 담당합니다. 프로젝트 2-1과 다른 점은 empty string을 'E' 대신 '_'로 사용했다는 점입니다.

4. main.py

실행 프로그램입니다. 위의 모든 파일들을 불러온 뒤 re.txt를 읽어 대응되는 m-DFA를 출력, 알맞은 형식에 맞춰 m-dfa.txt에 저장합니다.

실행 예시

> python main.py

testcase/re.txt가 정상적으로 파싱될 경우에 한해 testcase/m-dfa.txt가 생성됩니다.

주의사항

출력되는 m-DFA의 initial state 이름이 q0가 아닐 수 있음에 주의하시기 바랍니다.

e-NFA를 DFA로 변환하는 과정에서 기하급수적으로 많은 상태가 생성됩니다. 문자 2개로 생성된 E-NFA의 상태가 20개를 넘어갈 경우 프로그램 실행 시간이 7분 가량 나옵니다. 문자와 상태의 수가 너무 많다면 기말고사가 끝날 때까지 답안이 나오지 않을 수 있습니다.