고급 오토마타 이론 과제물 문제
Programming Minimization

최진영 <choi@formal.korea.ac.kr>

고려대학교 정형기법연구실

제출 기한: 2005년 6월 20일

1 문제

DFA를 최소화(minimize)시키는 프로그램을 작성하십시오.

2 기본 요구 사항

- 프로그램은 표준입력이나 파일로부터 DFA를 나타내는 문자열을 받아서, DFA를 나타내는 문자열을 표준출력이나 파일로 내보내야합니다.
- 또한, DFA를 나타내는 출력 문자열과 겹치지 않도록 따로 (예를 들어 표준 에러나 별도의 파일에), 프로그램의 수행 시간 측정치와 관련 정보를 출력해야합니다.
- 위 두 조건을 충족시킴으로써, 프로그램의 정상 (런타임 예외 상황이 발생하지 않았을 경우의) 출력은 또다시 프로그램의 입력으로 사용될 수 있어야합니다.
- 출력 결과는 언제나 최소화된 DFA를 나타내야합니다.

2.1 프로그램 입출력 문자열

프로그램의 출력 문자열들과 문법적으로 올바른 프로그램 입력 문자열들은 다음과 같은 POSIX 확장 정규표현식 *dfa*로 나타낼 수 있는 정규언어입니다 (단, 이탤릭체로 나타낸 단어는 그 자체가 다시 정규표현식이고, ε은 길이가 0인 빈 문자열, ␣는 한 칸의 공백 문자입니다):

$$\begin{aligned}
nat &= \texttt{0} \mid \texttt{[1–9][0–9]}^* \\
states &= \varepsilon \mid nat(\texttt{,}nat)^* \\
char &= \texttt{[a–z]} \\
alphabet &= \varepsilon \mid char(\texttt{,}char)^* \\
triple &= nat\texttt{:}char\texttt{:}nat \\
func &= \varepsilon \mid triple(\texttt{,}triple)^* \\
dfa &= states\texttt{\#}alphabet\texttt{\#}func\texttt{\#}nat\texttt{\#}states\texttt{[␣\textbackslash t\textbackslash r]}^*
\end{aligned}$$

위 정규표현식을 자연어로 설명하면, 예를 들어 다음과 같은 DFA

$$(\{q_0, q_1, q_2\}, \{a, b\}, \{((q_0, a), q_1), ((q_0, b), q_2), ((q_1, a), q_2), ((q_1, b), q_0), ((q_2, a), q_0), ((q_2, b), q_1)\}, q_0, \{q_1, q_2\})$$

를 프로그램의 입출력 문자열로 표현하는 데에 있어서:

- tuple을 둘러싸는 괄호는 쓰지 않습니다.
- DFA 5-tuple의 다섯 개 원소를 서로 구분짓는 기호로서 ‘,’ 대신 ‘`#`’를 씁니다.[1]
- 집합을 둘러싸는 중괄호는 쓰지 않습니다.
- 집합의 원소들을 서로 구분짓는 기호로서 ‘`,`’를 그대로 씁니다.
- 상태를 나타내는 ‘q_n’은 q를 빼고 아래첨자였던 자연수 ‘`n`’만 십진수로 씁니다. 선행 0은 있을 수 없습니다. (`17`은 옳지만 `0017`은 옳지 않습니다.)
- 전이기호를 나타내는 문자들은 로마자 소문자 `a`부터 `z`까지 26개 문자들 중 일부입니다.
- 전이함수 $\delta : Q \times \Sigma \to Q$는 tuple들의 집합입니다. 위에서 약속한 대로 tuple을 둘러싸는 괄호는 쓰지 않고, tuple의 각 원소를 구분짓는 기호로서 ‘,’ 대신 ‘`:`’를 사용합니다.
- DFA를 나타내는 프로그램 입력 문자열의 맨 마지막에는 임의의 갯수의 whitespace (공백, 탭, 개행) 문자들이 추가로 붙을 수 있습니다.[2]

즉, 수식으로 나타낸 위 DFA를 적법한 프로그램 입력 문자열로 나타내면

```
0,1,2#a,b#0:a:1,0:b:2,1:a:2,1:b:0,2:a:0,2:b:1#0#1,2
```

입니다.

2.2 프로그램의 예외 검사/처리

프로그램은 실행시의 환경과 인수에 따라 몇 가지 예외 상황에 대한 적절한 검사와 처리를 해야합니다. 그 중에서 필수적으로 구현해야하는 부분은 다음과 같습니다:

2.2.1 입력 문자열의 문법 오류 검사

추후 제공해드릴 공통 입력 파일들 중에는 문법적으로 적법하지 않은 (invalid) 문자열이 담겨있는 파일도 있을 수 있습니다. 문법적 적법성은 위 소절에서 제시한 확장 정규표현식을 기준으로 검사하시면 됩니다. 예를 들어

```
0,1,02#a,b # 0:a:1,0:b:2,1:a:2,1:b:0,2:a:0,2:b:1#0#1,2
```

는 문법적으로 옳지 않은 입력 문자열이며, 프로그램은 이를 감지할 수 있어야합니다.

이런 예외 상황을 제대로 감지했을 경우, 이에 대한 처리 결과 중 최소 요구 사항은:

- 프로그램의 메인 (결과 DFA 문자열) 출력은 없도록 하고
- 입력 문자열에 문법 오류가 있음을 표준에러 등에 적절히 표시하시면 됩니다.

[1] 이 숙제 문제지의 미공개 초안들에서는, 가독성을 위해 이 구분자로서 개행문자 시퀀스를 사용했었습니다. 그러나 각 플랫폼 마다 개행문자 처리방식이 서로 달라서 (MS-DOS는 LF/CR, UNIX는 LF, MacOS는 CR) 혼란과 오동작을 초래할 여지가 있으므로, 가독성을 일부 포기하고 부득이 이러한 문자열을 사용합니다.

[2] 엄밀한 의미에서는 이런 부가 조건이 필요 없지만, 실제로 많이 사용되는 텍스트 에디터나 문자열 저리기들 중 일부는 마지막 행이 개행문자로 끝나지 않을 경우 자동적으로 개행문자를 추가해버리는 만행을 저지릅니다. 이 조건은 이런 경우에 대비한 조치일 뿐입니다. 출력 문자열을 만드실 때에는 가능한 한 이런 trailing whitespace를 넣지 않도록 하십시오.

2.2.2 입력 문자열의 의미 오류 검사

위에서 제시한 프로그램 입력 문자열의 정규표현식을 유심히 보시면 아시겠지만, 정규표현식에 문법적으로 부합하는 적법한 문자열임에도 불구하고, 의미상으로는 올바른 DFA를 나타내지 못하는 경우도 있습니다. 예를 들어

```
0,1,2#a,b#0:a:1,0:c:2,1:a:2,2:a:0,2:b:1#0#2,3
```

는 문법적으로 올바른 문자열로서 정규표현식 dfa에 부합하지만, 그 의미상 DFA의 정의에 부합하지 않습니다. 추후 제공해드릴 공통 입력 파일들 중에는 이렇게 문법적으로는 옳지만 의미상 DFA가 아닌 문자열도 포함될 수 있으며, 프로그램은 이런 예외 상황을 적절히 감지하고 처리해야합니다. 이런 예외 상황을 제대로 감지했을 경우, 이에 대한 처리 결과 중 최소 요구 사항은:

- 프로그램의 메인 (결과 DFA 문자열) 출력은 없도록 하고
- 입력 문자열에 의미 오류가 있음을 표준에러 등에 적절히 표시하시면 됩니다.

단, 다음과 같은 경우는 의미 오류가 아닙니다:

- 상태들의 집합(DFA 5-tuple의 첫번째 원소)을 제외한 나머지 집합들은 공집합일 수 있습니다. 따라서

  ```
  0,1#a#0:a:1,1:a:1#0#
  ```

  는 종료 상태가 없는 올바른 DFA로서 의미상 옳습니다. (문법적으로도 당연히 옳습니다.)

- 집합은 문자열 내에서도 집합으로 간주합니다. 즉,

  ```
  2,2,5,5,2#c,c#2:c:5,5:c:2,2:c:5#2#5
  ```

  는 의미상 아무런 하자가 없는 DFA이며

  $$(\{q_2, q_5\}, \{c\}, \{((q_2, c), q_5), ((q_5, c), q_2)\}, q_2, \{q_5\})$$

  를 나타냅니다.

2.3 수행 시간 측정 루틴

프로그램에는 프로그램의 수행 시간을 측정하는 루틴이 들어있어야합니다. 여기서 말하는 '수행 시간'이란 순수하게 최소화 알고리즘이 수행되는 시간만을 뜻하며, 다음 항목들에 걸리는 시간은 포함하지 않습니다:[3]

- 프로그램 기동 시의 메모리 편성이나 인터프리터 적재
- 입력 파일 제어와 입력 문자열 분석
- 분석된 입력 결과로부터 적절한 자료 구조 구성
- 최소화된 결과 자료구조를 출력 문자열로 변환
- 출력 파일 제어와 출력 문자열 인쇄

[3]매우 적은 가능성으로나마, 이 부분에 약간의 변동이 생길 수 있습니다. 변경 사항이 생기면 즉시 재 공지하겠습니다.

2.4 결과 보고서

결과 보고서에는 다음 항목들을 필수적으로 기입해주셔야합니다:

- 수강생 학생 정보 (학과, 학번, 이름)
- 이 문서 5 쪽의 4.1 소절에서 말하고 있는 구현 언어 제한과 관련된 상세 정보들
- 자신의 소스 트리 전체 목록과 각 파일에 대한 간단한 설명
- 추후 제공할 공통 입력파일들과 이들을 각각 최소화한 결과 파일을 서로 대응시키는 파일명 대응표
- 추후 제공할 공통 입력파일 각각에 대한 최소화 수행 시간 측정값
- 위 시간 측정에 사용된 컴퓨터의 정확한 사양

3 추가 요구 사항

3.1 오류 처리 결과

문법 오류나 의미 오류가 발생했을 경우 윗 절에서 이야기한 오류 처리 이외에, 다음과 같은 처리를 부가적으로 해주시면 좋습니다:[4]

- 프로그램의 메인 출력이 표준출력이 아닌 실제 파일에 쓰여질 예정인 경우, 오류 상황에서는 파일을 생성하지 않거나 (이미 있는 파일이면) 덮어쓰지 않습니다.
- 프로그램은 오류 상황에서도 가능한 한 정상적으로 종료 과정을 거치도록 유도합니다. (메모리 반환 등)
- 프로그램을 호출한 부모 프로세스(예를 들면 유닉스 쉘)에 다음과 같은 exit code를 전달합니다:

  0: 오류 없음

  1: 입출력 오류 (입력파일 부재, 출력파일 권한 부족 등)

  2: 입력 문자열 문법 오류

  3: 입력 문자열 의미 오류

  4: 기타 오류

3.2 기타

- 프로그램 사용 방법을 설명해주십시오. 결과 보고서에 간단히 설명해주셔도 되고, UI에 도움말 항목으로 넣으시거나 명령행 옵션 중 '`-h`'와 '`--help`' (혹은 '`/h`') 등에 반응하도록 만드셔도 됩니다.

[4] Hint: `man 3 err` (BSD), `man 3 exit`

4 제한점

4.1 구현 언어

프로그램 구현 언어에는 제한이 없습니다. 단,

- 구현에 사용하신 프로그래밍 언어에 대하여 결과 보고서에 가능한 한 정확히 명시해주십시오. (예: 'C'보다는 'ISO C90 with GNU extensions'가 더 정확합니다.)
- 컴파일해야 실행 코드를 얻을 수 있는 언어일 경우:
  - 사용하신 컴파일러와 버전, 그리고 컴파일러가 이식된 환경을 정확히 명시하십시오. (예: 'gcc 2.95.1: SunOS 5.8 on Sun Blade 1000')
  - 자신의 프로그램을 컴파일할 때 사용한 컴파일 옵션들도 모두 정확하게 명시해주시기 바랍니다.
  - 만일 컴파일러의 표준 배포본이나 관련 환경에서 컴파일 스크립팅 도구를 지원한다면, 해당하는 컴파일 스크립트를 추가해주시기 바랍니다. (예를 들어 `Makefile` 등)
- 런타임 인터프리터가 필요한 언어일 경우, 자신의 프로그램을 실행하는 데에 필요한 인터프리터의 이름과 최소 버전, 이식된 환경들을 간단히 설명하시고, 그 인터프리터를 받고 설치하는 방법도 간단히 설명해주십시오. 또한, 인터프리터의 비호환 변종이나 언어 설계에 따른 버전, 이식 환경 등이 다양할 경우, 이들 중에서 자신이 프로그램을 작성하고 실행시킬 때 사용했던 버전과 플랫폼을 정확히 명시하십시오.
- 가상 기계가 필요한 언어일 경우, 필요한 가상 기계와 최소 버전, 배포 장소, 설치 방법 등을 간단히 설명하십시오. 같은 언어에 대한 가상 기계의 비호환 변종들이 다수 존재할 경우, 이들 중에서 자신이 사용한 가상 기계를 정확히 명시하셔야합니다.
- 자신이 선택하고 결과 보고서에 명시한 컴파일러나 인터프리터의 기본 배포판이 제공하는 표준 라이브러리 이외의 추가적인 별도 라이브러리(예: GMP, Haskell_FFI, Perl-like RegExp 등)를 사용하시는 경우에는 라이브러리의 이름, 버전, 배포 장소와 사용법 등을 함께 설명해주십시오. 이러한 추가 라이브러리와 함께 컴파일하는 경우에는 가능한 한 정적으로 링크해주시기 바랍니다.

4.2 파일명과 압축 도구

- 제출하시는 압축 파일에 포함된 어떤 파일에도 한글 파일명을 사용하지 말아주시기 바랍니다. 좀 더 정확히 제한하자면, 소스코드와 보고서를 포함하는 모든 파일들은 다음과 같은 정규표현식에 부합하는 파일명을 가져야만합니다.

  `[a–zA–Z][a–zA–Z0–9\._–]*`

  (조교가 UTF-8 문자셋을 기반으로하는 파일시스템을 사용하고 있어서 부득이 드리는 부탁입니다.)
- 특정 환경에서만 사용할 수 있는 압축 형식을 사용하지 말아주십시오. 특히 알집 전용 포맷으로 묶지 말아주십사 부탁드립니다. 권장하는 압축 포맷은 zip이며, tar로 묶으시거나 이를 다시 gzip으로 압축하셔도 상관 없습니다.

5 최종 제출물

아래 항목들을 하나의 파일로 묶어서 제출하십시오.

- 프로그램 소스 코드
- (컴파일해야하는 언어일 경우) 컴파일한 실행 코드
- 이 문제지와는 별도로 추후 제공해드릴 몇 개의 공통 입력파일 각각에 대한 출력파일들
- 결과 보고서

단, 만일 프로그램 소스 코드가 단일 파일이 아닐 경우에는 소스 트리 전체를 (디렉토리 구조를 유지하면서) 일단 먼저 묶어주시고, 그 결과를 다른 항목들과 다시 묶으시기 바랍니다.