컴파일러: 3장

국민대학교 컴퓨터공학부
강 승 식

1

제3장 정규식과 유한자동

- 정규 문법
  - 우선형 문법: A → aA | a
  - 좌선형 문법: A → Aa | a
- 시작기호 S → ε 이면, S가 타 생성규칙의 RHS에 나타나지 않아야 함
- 아래 문법은 정규 문법이 아님

  A → aA | Bb | c

  nonterminal이 섞여있으면 정규문법이 아님 한쪽에만 있어야함

2

정규식(Regular Expression)

- L_{mn}은 정규 언어, L_{nn}는 정규 언어 아님
  - $L_{mn} = \{a^m b^n \mid m,n \geq 1\}$ a^+b^+
  - $L_{nn} = \{a^n b^n \mid n \geq 1\}$
- 정규식(Regular Expression)
  - 특정 스트링 유형(string pattern)을 기술하는 표현 방식
  - $(0+1)^*$ -- 0 또는 1로 이루어진 스트링 유형
  - $(ba)^*a$ -- ba가 0번 이상 반복된 후에 a로 끝나는 스트링 유형

3

- 정규식에 사용되는 메타 문자

  \+ : 또는
  · : 스트링 결합(string concatenation)
  ( ) : 괄호 연산자
  * : 윗첨자 Kleene closure, 0번 이상 반복
  \+ : 윗첨자 dagger, 1번 이상 반복

4

정규식 예제

- 식별자(identifier)에 대한 정규식
  <letter>(<letter>+<digit>)* digit(0~9)
- 식별자에 대한 BNF
  <id> ::= <letter> | <id><ld> id는 순환규칙
  <ld> ::= <letter> | <digit>
  <letter> ::= a | b | … | z
  <digit> ::= 0 | 1 | … | 9

  BNF는 지금까지 생성규칙과 똑같은데 차이점이라면 ::=을 쓴다는 점
- Nonterminal 기호 대문자 1개 → 최대 26개 제약
  - BNF의 nonterminal 기호: <> 안에 의미있는 이름을 부여
- 정수, 실수, 주석(comment)에 대한 정규식은?

(letter,digit은 nonterminal) (abcd~, 0123~ 는 터미널)

5

정규식의 등가성(equality)

- 정규식 형태가 다르더라도 정규식으로 표현되는 스트링 집합이 동일하면 동일한 언어에 대한 정규식이다.

  두개는 일치함
  동일한 정규언어인데
  정규표현은 두가지임
  즉같은거임 (a^+)로 표현가능

  a로시작해서 a^*번

  $aa^* \equiv a^*a$ a^*번 a로끝나는거

  $(ab)^*a \equiv a(ba)^*$ 이런 경우들 많음

- $(ab)^*a$ 과 $a(ba)^*$ 에 대한 정규 문법은?

  A->abA|a
  (우선형문법)
  A->Aba|a
  (좌선형)
  역시 저거 두개 모두 같다

6

정규식 계산 방법

- 정규 문법으로 기술되는 정규 언어는 정규식을 구할 수 있음
  - CFG 등 정규 언어가 아닌 언어는 정규식으로 기술할 수 없음
- 유한 언어의 정규식은 모든 스트링 나열

S → a | b | aX | bX
X → a | b

X = a + b 을 S에 대입
S = a + b + aX + bX
= a + b + a(a+b) + b(a+b)
= a + b + aa +ab + ba + bb

- 무한 언어의 정규식은 반복되는 부분을 *(0번 이상 반복) 또는 $^+$(1번 이상 반복) 형태로 구해야 함

우순환 규칙의 정규식

- 우순환 규칙(right recursive rule)의 정규식

$X \rightarrow \alpha X \mid \beta$
$X = \alpha^* \beta$

- 무한 언어에 대한 정규식 구하기 예제

S → 0S | 1S | 0 | 1
S = (0+1)S + (0+1) 이므로 0+1이 알파
S = $(0+1)^*(0+1)$ = $(0+1)^+$

책 필기참고

- 아래 정규 문법 G_1에 대한 정규식은?

G_1 = ({O, E}, {a}, P, O)
P: O → a | bE
E → aO

O = a + bE = a + baO = baO + a 이므로
O = $(ba)^*a$

- 아래 정규 문법 G_2에 대한 정규식은?

G_2 = ({S, A, B, C}, {a, b, c, d}, P, S)
P: S → aA
A → aA | bB
B → bB | cC
C → cC | d

C = cC + d 로부터 C = c^*d

B = bB + cC
= bB + cc^*d
= bB + c^+d
B = b^*c^+d

A = aA + bB
= aA + bb^*c^+d
= aA + b^+c^+d
A = $a^*b^+c^+d$

S = aA = $aa^*b^+c^+d$ = $a^+b^+c^+d$

정규식 구하는 연습

- 정규문법-1

X → aX | bY | a
Y → bX | aY | b

- 정규문법-2

X → aX | bY | cZ | a
Y → bX | cY | aZ | b
Z → cX | aY | bZ | c

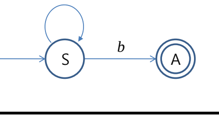
- 좌선형 문법에 대한 정규식을 구하는 방법은?

S → Sb | Aa | c
A → Aa | b

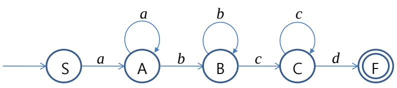
X → Xa | Yb | a
Y → Xb | Ya | b

유한 오토마타(Finite Automata)

- 유한 오토마타의 기술 방법
  - 상태(state 또는 node)와 레이블 있는 지시선(labelled arc)으로 구성
  - 시작 상태(start state)는 시작 지시선으로 표시 1개
  - 끝 상태(final state)는 이중 원으로 표시 1개 이상
- 정규식 a^*b 에 대한 유한 오토마타

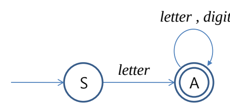


- 정규식 $a^+b^+c^+d$ 에 대한 유한 오토마타



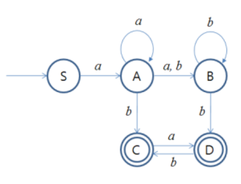
	a	b	c	d
S	A			
A	A	B		
B		B	C	
C			C	F
F				

- 식별자에 대한 유한 오토마타



- 아래 상태전이도(state transition diagram)로 기술된 유한 오토마타는?

	a	b
S	{A}	
A	{A,B}	{B,C}
B		{B,D}
C	{D}	
D		{C}



- 유한 오토마타의 구성 요소
  - state 집합: {S, A, B, C, D}
  - 입력 심볼 집합: {a, b}
  - 전이 함수(mapping function)
  - start state: S
  - final state 집합: {C, D}

DFA와 NFA

- 결정적 유한 오토마타(DFA)
  - 각 state에서 입력 심볼에 대해 next state가 항상 1개로 결정

A → aA | bB

- 비결정적 유한 오토마타(NFA)
  - 입력 심볼에 대해 next state가 1개로 결정되지 않는 것이 있는 경우

A → aA | aB

DFA와 NFA의 차이점

- DFA는 next state가 항상 유일하게 결정됨
  - 백트래킹(back tracking) 없이 O(n) 시간에 해당 언어를 인식하는 프로그램 구현 가능
- NFA는 next state가 유일하게 결정되지 않는 경우가 발생
  - 언어 인식 프로그램 구현이 어려움

NFA를 DFA로 변환

- 변환 방법
  - next state가 2개 이상인 state 집합을 새로운 1개의 state로 다시 정의
- 변환 예
  - NFA:

	a	b
S	{A}	
A	{A,B}	{B,C}
B		{B,D}
C	{D}	
D		{C}

19

NFA를 DFA로 변환 예제

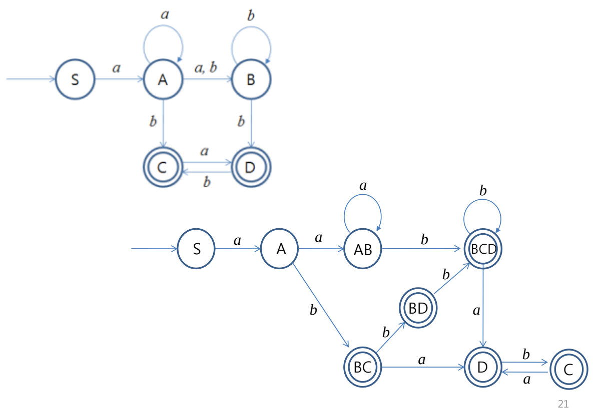
	a	b
S	{A}	
A	{A,B}	{B,C}
B		{B,D}
C	{D}	
D		{C}

Final states ={ C, D }

	a	b
[S]	[A]	
[A]	[A,B]	[B,C]
[A,B]	[A,B]	[B,C,D]
[B,C]	[D]	[B,D]
[B,C,D]	[D]	[B,C,D]
[D]		[C]
[B,D]		[B,C,D]
[C]	[D]	

Final states = { [B,C], [B,C,D], [D], [B,D], [C] }

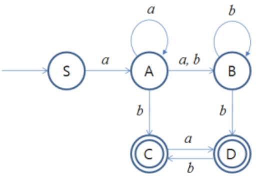
20



21

유한 오토마타를 정규문법으로

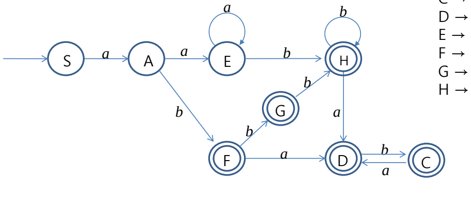
- NFA(DFA 변환 예제)



S → aA
A → aA | aB | bB | bC
B → bB | bD
C → aD | ε
D → bC | ε

22

- DFA 변환 후



S → aA
A → aE | bF
C → aD | ε
D → bC | ε
E → aE | bH
F → bG | aD | ε
G → bH | ε
H → aD | bH | ε

23

- NFA를 DFA로 변환할 때 주의할 점
  - NFA의 final state를 1개라도 포함하고 있는 DFA의 모든 state는 final state가 된다.
- 유한 오토마타를 정규문법으로 기술할 때 주의할 점
  - 모든 final state들에 대해 ε-생성규칙을 추가한다.
- 정규문법을 유한 오토마타로 구성하는 방법은 무엇인가?

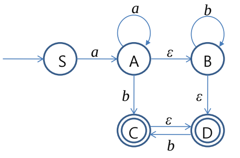
24

- 3.2절의 NFA를 DFA로 변환하시오.
  1) S → a | b | aX | bX
     X → a | b

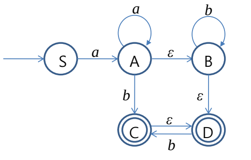
  2) S → 0S | 1S | 0 | 1
- DFA로 변환하기 전후의 정규문법들에 대해 정규식을 구하고, NFA와 DFA가 동일한 언어를 기술하는지 비교해 보시오.
- 일반적으로 NFA를 DFA로 변환하면 state 개수가 증가한다. NFA의 state 개수가 n일 때 DFA의 최대 state 개수는 몇 개인가?

ε-NFA를 DFA로 변환

- 변환 방법
  - NFA를 DFA로 변환하는 방법과 동일함
  - next state를 재정의할 때 ε-transition을 고려하여 도달 가능한 모든 state 집합이 next state가 됨
  - next state 집합을 구하기 위해 ε-CLOSURE 함수 사용
  - ε-CLOSURE 함수는 ε-transition으로 도달 가능한 state 집합



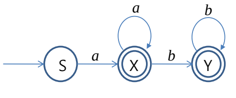
ε-CLOSURE(A) = { A, B, D }
ε-CLOSURE(B) = { B, D }
ε-CLOSURE(C) = { C, D }



ε-CLOSURE(A) = { A, B, D }
ε-CLOSURE(B) = { B, D }
ε-CLOSURE(C) = { C, D }

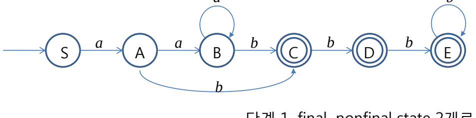
	a	b
[S]	[A,B,D]	
[A,B,D]	[A,B,D]	[B,C,D]
[B,C,D]		[B,C,D]

Final states = { [A,B,D], [B,C,D] }



가장 효율적인 DFA로 변환

- 가장 효율적인 DFA는 state 개수가 가장 적은 것
- DFA의 상태수 최소화 방법
  - 동일한 기능을 하는 state들이 있다면 이를 1개 state로 merge함
    임의의 2개 state가 동일한 기능을 하는지 판단이 어려움
  - 모든 state들을 final state 집합과 non-final state 집합으로 merge한 후에 원래 DFA와 동일한 기능인지 확인
  - 원래 DFA와 동일하지 않으면 문제가 되는 state들끼리 state 분할

- 

	a	b
S	A	
A	B	C
B	B	C
C		D
D		E
E		E

Final states = { C, D, E }

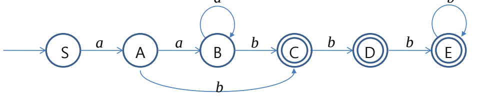
단계 1. final, nonfinal state 2개로 분할

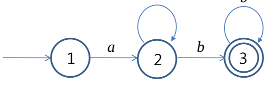
	1: {S, A, B}	2: {C, D, E}
a	1 1 1	φ φ φ
b	φ 2 2	2 2 2

단계 2. DFA 요건 불만족 state 분할

	1: {S}	2: {A, B}	3: {C, D, E}
a	2	2 2	φ φ φ
b	φ	3 3	3 3 3

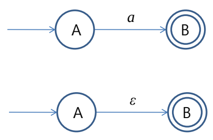
상태수 최소화 전후 비교



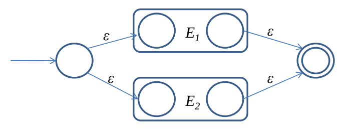


정규식을 유한 오토마타로 구성

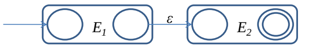
- a-transition, ε-transition을 인식하는 유한 오토마타



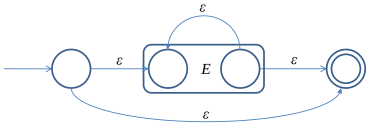
- 정규식 $E_1 + E_2$를 인식하는 유한 오토마타



- 정규식 $E_1 \cdot E_2$를 인식하는 유한 오토마타

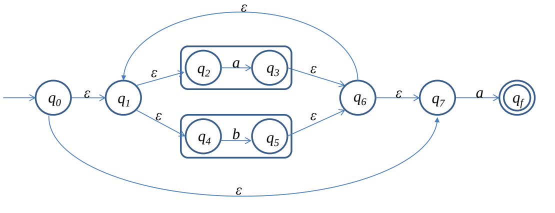


- 정규식 E^*를 인식하는 유한 오토마타



유한 오토마타 구성 예제

- $(a+b)^*a$

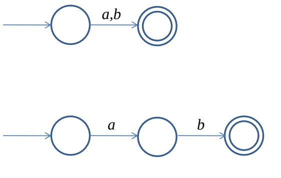


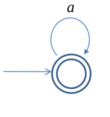
- 아래 정규식에 대해 유한 오토마타 구성

  $(aa^*bb^*)^*(aa+bb)$

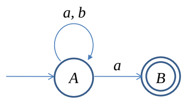
  $ab(aa+bb)^*a$

- $a+b$, ab, a^*에 대한 유한 오토마타





- (a+b)*a에 대한 간소화된 유한 오토마타



- 정규문법
  A → aA | bA | aB
  B → ε

- 상태전이표: NFA

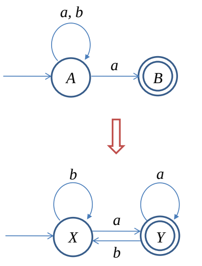
	a	b
A	A, B	A
B		

Final states = { B }

- NFA를 DFA로 변환

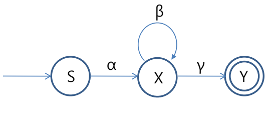
	a	b
[A]	[A,B]	[A]
[A,B]	[A,B]	[A]

Final states = { [A,B] }



정규언어의 속성: Pumping Lemma

- 어떤 언어가 정규언어가 아님을 증명하는데 활용
- 유한 오토마타(상태수가 유한 개)로 상태수보다 긴 스트링을 인식하려면 반복되는 부분이 있어야 한다.
- 길이가 상태수보다 긴 스트링 ω=αβγ의 반복되는 부분을 β라 하면 αβ*γ도 이 오토마타로 인식할 수 있다.



- $a^n b^n$ 은 정규언어가 아님을 증명하시오.