

# 오토마타 이론

2024년 1월

서울대학교 컴퓨터공학부

박근수

# Contents

<b>1 서론</b>	<b>1</b>
1.1 관계 . . . . .	2
1.2 그래프 . . . . .	3
1.3 언어 . . . . .	4
1.4 증명 기법 . . . . .	6
1.5 연습 문제 . . . . .	7
<b>2 정규언어</b>	<b>9</b>
2.1 정규식 . . . . .	9
2.2 유한 오토마타 . . . . .	11
2.3 비결정 유한 오토마타 . . . . .	16
2.4 유한 오토마타의 동등성 . . . . .	19
2.5 정규식과 유한 오토마타 . . . . .	22
2.6 정규언어의 성질 . . . . .	25
2.7 유한 오토마타의 응용 . . . . .	29
2.8 유한 오토마타의 최소화 . . . . .	30
2.9 연습 문제 . . . . .	33
<b>3 문맥무관 언어</b>	<b>43</b>
3.1 문맥무관 문법 . . . . .	43
3.2 정규문법 . . . . .	49
3.3 파스 트리 . . . . .	52
3.4 표준형 . . . . .	55

3.5 내리누름 오토마타 . . . . .	58
3.6 내리누름 오토마타와 문맥무관 언어 . . . . .	61
3.7 문맥무관 언어의 성질 . . . . .	67
3.8 결정 내리누름 오토마타 . . . . .	72
3.9 하향 파싱 . . . . .	74
3.10 연습 문제 . . . . .	80
<b>4 튜링기계</b>	<b>89</b>
4.1 튜링기계 . . . . .	89
4.2 확장된 튜링기계 . . . . .	95
4.3 비결정 튜링기계 . . . . .	99
4.4 무제한 문법 . . . . .	101
4.5 Church-Turing의 명제 . . . . .	103
4.6 Chomsky 체계 . . . . .	105
4.7 연습 문제 . . . . .	108
<b>5 계산불가</b>	<b>111</b>
5.1 무한집합의 크기 . . . . .	111
5.2 범용 튜링기계 . . . . .	113
5.3 정지문제 . . . . .	115
5.4 계산불가 문제 . . . . .	118
5.5 연습 문제 . . . . .	126
<b>6 계산 복잡도</b>	<b>129</b>

# Chapter 1

## 서론

컴퓨터의 사용이 일반화되고, 이로 인해 컴퓨터에 의한 계산이 어디에서든지 이루어지고 있다. 이러한 컴퓨터의 발전은 컴퓨터의 기초 학문이라고 할 수 있는 컴퓨터 이론에 그 바탕을 두고 있다. 컴퓨터 이론에서 다음의 근본적인 질문에 답하고자 한다.

- 컴퓨터로 무엇을 계산할 수 있고 무엇을 계산할 수 없는가? 놀랍게도 우리가 자연스럽게 풀을 수 있는 문제 중에서 컴퓨터로 풀 수 없는 문제(unsolvable problem)가 많이 있다.
- 컴퓨터로 풀 수 있는 문제 중에서 컴퓨터가 빨리 풀 수 없는 문제(intractable problem)는 어떤 것이며 빨리 풀 수 있는 문제(tractable problem)는 어떤 것인가?

위의 질문에 답하기 위해서는 먼저 컴퓨터로 계산한다는 것이 무엇인가를 규정해야 한다. 이를 위해 컴퓨터(계산)의 이론적인 모델들을 고려한다. 계산능력이 커지는 순서로 유한 오토마타, 내리누름 오토마타, 튜링기계 등을 배우게 된다. 또한 이러한 모델들과 밀접한 관련을 가진 문법(grammar)을 다루게 된다.

## 1.1 관계

집합은 원소들을 모아놓은 것이다. 자연수의 집합  $N$ 은  $\{0, 1, 2, \dots\}$ 이다. 집합  $A$ 의멱집합(power set)  $2^A$ 은  $A$ 의 모든 부분집합들의 집합이다. 따라서  $|2^A| = 2^{|A|}$ 이다. 예를 들어,  $A = \{0, 1\}$ 이면  $2^A = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$ 이다.

순서쌍  $(a, b)$ 는 두 원소  $a, b$ 에 순서를 부여한 쌍이다. 여기에서  $a, b$ 는 같은 원소일 수 있다. 집합  $A, B$ 의 Cartesian(카티션) 곱  $A \times B$ 는  $a \in A, b \in B$ 인 모든 순서쌍  $(a, b)$ 의 집합이다. 원소가 3개 이상일 때, 순서열  $(a_1, \dots, a_n)$ 이라고 부른다.

- 두 집합  $A, B$ 에 대한 이원관계(binary relation)는  $A \times B$ 의 부분집합이다.
- 집합  $A$ 에서 집합  $B$ 로의 함수(function)는  $A, B$ 에 대한 이원 관계  $R$ 로서 각 원소  $a \in A$ 에 대해 순서쌍  $(a, b) \in R$ 이 하나만 존재하는 성질을 가진다.  $A$ 에서  $B$ 로의 함수를 일반적으로  $f : A \rightarrow B$ 로 적는다.
- 부분함수(partial function)는 각 원소  $a \in A$ 에 대해 순서쌍  $(a, b) \in R$ 이 하나 있거나 없는 경우이다.

**예제 1.1**  $A = \{a, b, c\}$ 이고  $B = \{0, 1\}$ 이라고 하자.  $\{(a, 0), (b, 0), (b, 1)\}$ 은 이원관계이고,  $\{(a, 0), (b, 1), (c, 0)\}$ 은 함수이고,  $\{(a, 1), (c, 0)\}$ 은 부분함수이다.

$R \subseteq A \times A$ 인 관계  $R$ 을 집합  $A$ 에서의 관계라고 부른다. 집합  $A$ 에서의 관계  $R$ 에 대하여 다음과 같은 성질을 고려하자.

- 모든  $a \in A$ 에 대하여  $(a, a) \in R$ 이면  $R$ 은 반사적(reflexive)이다.
- $(a, b) \in R$ 이면  $(b, a) \in R$ 을 만족하면  $R$ 은 대칭적(symmetric)이다.
- $(a, b) \in R$ 이고  $(b, c) \in R$ 이면  $(a, c) \in R$ 을 만족하면  $R$ 은 이행적(transitive)이다.

반사적이고 대칭적이고 이행적인 관계를 동치관계(equivalence relation)라고 부른다.  $R$ 을 집합  $A$ 에서의 동치관계라고 하자.  $R$ 에 의해 원소  $a \in A$ 와 동치인 원소들의 집합을 동치류(equivalence class)라고 부르고  $[a]$ 로 표시한다. 즉  $[a] = \{b : (a, b) \in R\}$ . 동치관계  $R$ 은 집합  $A$ 를 동치류들로 분할한다. 즉  $A_1, A_2, \dots$ 을  $R$ 의 동치류라고 하면

- $A_i \neq \emptyset$
- $i \neq j$ 에 대해  $A_i \cap A_j = \emptyset$
- $\bigcup_i A_i = A$

를 만족한다.

**예제 1.2** 자연수의 집합  $N$ 에서의 관계  $R = \{(a, b) : a \equiv b \pmod{4}\}$ 는 동치관계이고 다음 네 개의 동치류가 존재한다.

$$\begin{aligned}[0] &= \{0, 4, 8, 12, \dots\} \\ [1] &= \{1, 5, 9, 13, \dots\} \\ [2] &= \{2, 6, 10, 14, \dots\} \\ [3] &= \{3, 7, 11, 15, \dots\}\end{aligned}$$

## 1.2 그래프

그래프  $G = (V, E)$ 는 정점(vertex)의 집합  $V$ 와 간선(edge)의 집합  $E$ 로 구성된다. 방향그래프(directed graph)에서 간선은  $a, b \in V$ 인 순서쌍  $(a, b)$ 이다. 무방향그래프(undirected graph)에서 간선은  $a, b \in V$ 인 집합  $\{a, b\}$ 이다. 무방향그래프에서 간선을  $(a, b)$ 로 표시하기도 한다.

그래프  $G = (V, E)$  상의 경로(path)는  $(v_{i-1}, v_i) \in E$  ( $1 \leq i \leq k$ )를 만족하는 정점의 순서열  $(v_0, \dots, v_k)$ 이다 (또는 간선의 순서열). 이 경로의 길이는  $k$ 이다. 그래프에서 경로  $(v_0, \dots, v_k)$ 가  $k > 0$ 과

$v_0 = v_k$ 를 만족하면 사이클(cycle)이라 부른다. 또한  $v_1, \dots, v_k$ 가 서로 다르면 단순 사이클이다.

사이클이 없는 연결된 그래프를 트리(tree)라고 부른다. 루트 노드(root node)를 가지고 각 노드의 자식 노드(child node)가 2개 이하인 트리가 이진 트리(binary tree)이다. 자식이 없는 노드를 단말 노드라고 부른다. 이진트리에서 각 노드의 두 자식을 왼쪽 자식과 오른쪽 자식이라고 부른다. 루트 노드에서 임의의 노드  $x$ 까지 경로의 길이를  $x$ 의 깊이라고 한다. 트리에서 단말 노드의 깊이의 최대 값을 트리의 깊이(또는 높이)라고 부른다.

### 1.3 언어

언어를 정의하기 위하여 먼저 알파벳과 스트링을 정의한다.

- 알파벳은 글자들의 유한 집합이다. 일반적으로 알파벳을  $\Sigma$ 로 표시하고  $\{0, 1\}$ ,  $\{a, b, c, \dots, z\}$  등이 그 예이다.
- 알파벳  $\Sigma$ 상의 스트링은  $\Sigma$ 에 있는 글자들의 유한 열이다. 예를 들면, 1010과 00111은  $\Sigma = \{0, 1\}$ 상의 스트링들이고, abcab와 automata는  $\Sigma = \{a, b, c, \dots, z\}$ 상의 스트링들이다.
- 스트링  $w$ 의 길이는  $w$ 에 있는 글자들의 개수이고  $|w|$ 로 표시된다. 즉  $|abcab| = 5$ . 길이가 0인 스트링을 공스트링이라 부르고  $\epsilon$ 로 표시한다.
- 공스트링을 포함하여 알파벳  $\Sigma$ 상의 모든 스트링의 집합을  $\Sigma^*$ 로 표시한다. 즉,  $\{0, 1\}^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$ .

스트링에 대한 두 가지 연산을 정의한다.

- 두 스트링  $x, y$ 의 접합(concatenation)  $x \cdot y$  (또는  $xy$ )는  $x$  뒤에  $y$ 를 붙인 것이다. 즉  $x = 011$ 이고  $y = 01$ 이면,  $xy = 01101$ 이고  $yx = 01011$ 이다. 스트링  $w$ 에 대해서  $w^0 = \epsilon$ 이고,  $w^k = w^{k-1}w$  ( $k \geq 1$ )이다. 따라서  $w = 011$ 이면,  $w^2 = 011011$ 이다.

- 스트링  $w$ 의 역(reverse)  $w^R$ 은  $w$ 를 역순으로 나열한 스트링이다. 즉  $w = 011$ 이면,  $w^R = 110$ 이다. 두 스트링  $u, v$ 에 대하여  $(uv)^R = v^Ru^R$  이다.

스트링에 대한 몇 가지 용어를 정의한다.

- $v$ 가 스트링  $w$ 의 일부분일 때(즉  $w = xvy$ ),  $v$ 는  $w$ 의 부분스트링이다. 즉 011은 00111의 부분스트링이다.
- $v$ 가  $w$ 의 앞부분에 있는 부분스트링일 때(즉  $w = vy$ ),  $v$ 는  $w$ 의 어두(prefix)이다. 011의 어두는 0, 01, 011이다.
- $v$ 가  $w$ 의 뒷부분에 있는 부분스트링일 때(즉  $w = xv$ ),  $v$ 는  $w$ 의 어미(suffix)이다. 011의 어미는 1, 11, 011이다.

알파벳  $\Sigma$ 상의 스트링의 집합, 즉  $\Sigma^*$ 의 부분집합을 언어(language)라고 부른다. 이것은 언어에 대한 가장 추상적인 정의인데, 구체적인 예로 프로그래밍 언어를 들 수 있다. C 프로그램이란 C 언어의 문법에 맞는 스트링이다. 따라서 C 언어는 C 프로그램(즉 스트링)들의 집합이 된다. 언어들의 집합을 언어 종류(language class)라고 부른다.

언어는 집합이므로 언어들에 대하여 합집합, 교집합, 여집합 등의 연산을 적용할 수 있다. 이에 더하여 다음 두 가지 연산을 정의한다.

- 언어  $L_1, L_2$ 의 접합  $L_1 \cdot L_2$  (또는  $L_1L_2$ )는

$$\{uv : u \in L_1, v \in L_2\}$$

이다. 즉  $L_1 = \{0, 00, 000\}$ 이고  $L_2 = \{1, 11\}$ 이면,  $L_1L_2 = \{01, 001, 0001, 011, 0011, 00011\}$ 이다.  $L^0 = \{\epsilon\}$ 이고,  $L^k = L^{k-1}L$  ( $k \geq 1$ )이다. 즉  $L = \{1, 11\}$ 이면,  $L^2 = \{11, 111, 1111\}$ 이다.

- 언어  $L$ 의 Kleene(클리네) 곱  $L^*$ 는  $L$ 을 0번 이상 접합하여 만 들어지는 모든 스트링의 집합이다. 즉,

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots .$$

$L = \{0\}$ 이면,  $L^* = \{\epsilon, 0, 00, 000, \dots\}$ 이다. 따라서  $\Sigma^*$ 는  $\Sigma$ 의 Kleene 곱인 것을 알 수 있다.  $LL^*$ 를  $L^+$ 로 표기하기도 한다 (즉  $L^+$ 는  $L$ 을 1번 이상 접합하여 만들어지는 모든 스트링의 집합).

## 1.4 증명 기법

증명 기법으로 수학적 귀납법과 비둘기집 원리를 많이 사용하게 된다.

수학적 귀납법(mathematical induction): 모든 자연수  $n$ 에 대하여  $A$ 가 성립함을 다음과 같이 증명할 수 있다.

1. 귀납 기초(basis):  $n = 0$ 일 때,  $A$ 가 성립함을 보인다.
2. 귀납 가설(induction hypothesis): 임의의  $n \geq 0$ 에 대하여 (또는 0부터  $n$ 까지 모든 수에 대하여)  $A$ 가 성립한다고 가정한다.
3. 귀납 과정(induction step):  $n + 1$ 에 대하여  $A$ 가 성립함을 보인다.

**예제 1.3** 높이가  $n$ 인 이진 트리에서 단말 노드의 개수는  $2^n$  이하임을 수학적 귀납법으로 증명하라.

- 귀납 기초: 높이가 0인 이진 트리의 단말 노드는 1 개이다.
- 귀납 가설: 높이가  $n \geq 0$  이하인 이진 트리의 단말 노드 개수가  $2^n$  이하라고 가정한다.
- 귀납 과정:  $T$ 는 높이가  $n+1$ 인 이진 트리라고 하자.  $T$ 의 루트 노드의 왼쪽 서브트리와 오른쪽 서브트리는 높이가  $n$  이하이다. 귀납 가설에 의해 왼쪽 서브트리의 단말 노드의 개수는  $2^n$  이하이고, 오른쪽 서브트리도 마찬가지이다. 따라서 이진 트리  $T$ 의 단말 노드 개수는  $2^{n+1}$  이하이다.

비둘기집 원리(pigeonhole principle):  $A$ 와  $B$ 가 유한집합이고  $|A| > |B|$ 이면,  $A$ 에서  $B$ 로의 일대일 함수가 존재하지 않는다. 즉, 비둘기가  $|A|$  개이고, 비둘기집이  $|B|$  개이면, 어떤 집에는 비둘기를 2개 이상 넣어야 한다.

비둘기집 원리의 일반적인 형태는 다음과 같다.  $n$  개의 비둘기를  $c$  개의 비둘기집에 넣으려면, 어떤 집에는  $\lceil n/c \rceil$  개 이상의 비둘기를 넣어야 한다.

**예제 1.4** 100 명의 사람이 있으면, 그 중 같은 달에 태어난 사람이 적어도  $\lceil 100/12 \rceil = 9$  명 있다.

## 1.5 연습 문제

1. 유한집합  $A$ 에 대하여  $|2^A| = 2^{|A|}$ 임을 수학적 귀납법으로 증명 하라.
2. 두 명 이상 모인 모임에서 아는 사람의 수가 같은 사람이 적어도 2명 있음을 증명하라. 여기에서 ‘안다’는 것은 ‘서로 안다’를 의미한다. (비둘기집 원리 사용)



# Chapter 2

## 정규언어

### 2.1 정규식

수식(arithmetic expression)과 유사하게 정규식(regular expression)을 정의한다.

**정의 2.1** 알파벳  $\Sigma$  상의 정규식과 그것이 표시하는 집합은 다음과 같다

1.  $\emptyset$ 는 정규식이고 공집합을 표시한다.
2.  $\epsilon$ 는 정규식이고  $\{\epsilon\}$ 을 표시한다.
3. 각  $a \in \Sigma$ 에 대하여  $a$ 는 정규식이고  $\{a\}$ 를 표시한다.
4.  $r$ 과  $s$ 가 정규식이고 각각  $R$ 과  $S$  집합을 표시한다면,  $(r+s)$ ,  $(rs)$ ,  $(r^*)$ 는 정규식이고 각각  $R \cup S$ ,  $RS$ ,  $R^*$  집합을 표시한다.

정규식  $r$ 의 언어  $L(r)$ 은  $r$ 이 표시하는 집합이다.

정규식  $a$ 와 글자  $a$ 는 서로 다른 것을 나타냄을 유의하라. 즉  $L(a) = \{a\}$ 이고  $L(\epsilon) = \{\epsilon\}$ 이다. 정규식  $a$ 와 글자  $a$ 는 서로 다르게 표시되어야 하나, 이후에는 편의상 정규식의 진한 표시를 생략하기로 한다.

정규식에서 연산의 순서를  $* \cdot +$ 로 정하면 많은 괄호를 생략할 수 있다. 즉  $((1(0^*)) + 0) = 10^* + 0$ 이고 이 정규식은  $\{0, 1, 10, 100, \dots\}$ 을 나타낸다. 또한 다음과 같은 연산의 법칙을 적용할 수 있다.

- $+$  연산은 합집합을 나타내므로, 교환법칙과 결합법칙이 성립한다. 즉  $r + s = s + r$ 이고  $(r + s) + t = r + (s + t)$ 이다.
- $\cdot$  연산은 접합을 나타내므로, 교환법칙은 성립하지 않고 결합법칙은 성립한다. 즉  $(rs)t = r(st)$ 이다.
- $+ \cdot$ 와  $\cdot \cdot$  사이에 분배법칙이 성립한다. 즉  $r(s + t) = rs + rt$ 이고  $(r + s)t = rt + st$ 이다.

**정의 2.2** 정규식으로 표시되는 언어를 정규언어라고 부른다.

### 예제 2.1 정규식

$$(00)^*(11)^*1$$

은 짹수 개의 0 다음에 홀수 개의 1이 나오는 스트링의 집합을 표시한다.

### 예제 2.2 $\{0^n 1^m : (n + m)\text{은 짹수}\}$ 를 나타내는 정규식을 구하라.

정규식의 연산  $+$ 는 “또는”의 의미이므로, 정규식을 구할 때 여러 경우로 나누어서 생각하는 것이 좋다. 0이 짹수 개 나오고 1이 짹수 개 나오든지 0이 홀수 개 나오고 1이 홀수 개 나와야 되므로, 답은  $(00)^*(11)^* + 0(00)^*1(11)^*$ 이다.

앞으로 예제에서 알파벳에 대한 별다른 언급이 없으면  $\Sigma = \{0, 1\}$ 이다.

### 예제 2.3 1이 한 개 또는 두 개 있는 스트링의 집합을 표시하는 정규식을 구하라.

1이 한 개 있는 스트링의 집합은  $0^*10^*$ 이고, 1이 두 개 있는 스트링의 집합은  $0^*10^*10^*$ 이므로 둘을 합치면  $0^*10^*(\epsilon + 10^*)$ 이다.

**예제 2.4** 길이가 3의 배수인 스트링의 집합을 표시하는 정규식을 구하라. 길이가 3인 스트링을 표시하는 정규식이  $(0+1)(0+1)(0+1)$  이므로 답은  $((0+1)(0+1)(0+1))^*$ 이다.  $(0+1)(0+1)(0+1)$ 을 줄여서  $(0+1)^3$ 으로 쓰기로 한다.

**예제 2.5** 11을 부분스트링으로 갖는 스트링의 집합을 표시하는 정규식은  $(0+1)^*11(0+1)^*$ 이다.

**예제 2.6**  $\Sigma = \{a, b, c\}$  일 때, 첫 글자가 다시 나타나지 않는 스트링의 집합을 표시하는 정규식을 구하라.

**예제 2.7** 111이 딱 한 번 나타나는 스트링의 집합을 표시하는 정규식을 구하라.

**예제 2.8** 11을 부분스트링으로 갖지 않는 스트링의 집합을 표시하는 정규식을 구하라.

언어를 표현하는 방식에는 정규식, 오토마타, 문법 등이 있다.

**정의 2.3** 언어의 표현 방식  $A, B$ 가  $L(A) = L(B)$ 를 만족하면,  $A$ 와  $B$ 는 동등(equivalent)하다.

## 2.2 유한 오토마타

컴퓨터는 일반적으로 중앙처리장치, 메모리, 입출력장치로 구성되어 있다. 앞으로 컴퓨터의 이론적인 모델들을 배우게 되는데, 그 중에서 가장 간단한 것이 유한 오토마타이다. 유한 오토마타는 중앙처리장치에 해당하는 유한제어기와 입력장치인 입력테입으로 구성된다. 유한 오토마타의 특징은 메모리를 가지고 있지 않다는 것이다. 입력테입은 칸들로 나누어지고 각 칸은 하나의 글자를 가질 수 있다. 입력테입에는 헤드(head)가 있어서 헤드가 가리키는 칸에 있는 글자를 읽을 수 있다. 유한제어기는 상태들로 구성된다. 그림 2.1를 보라.

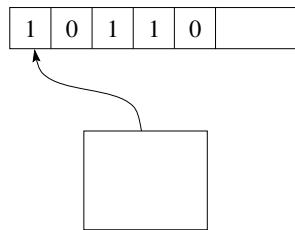


Figure 2.1:

초기에 헤드는 입력테입의 맨왼쪽 칸을 가리키고, 유한제어기는 초기 상태에 놓여 있다. 각 단계에서 현재 상태와 테입의 글자에 의해서 다음 상태가 결정되고, 헤드는 한 칸 오른쪽으로 진행한다.

**정의 2.4** 결정 유한 오토마타  $M$ 은 다섯 가지 요소  $(Q, \Sigma, \delta, q_0, F)$ 로 구성된다. 여기에서

1.  $Q$ 는 상태들의 유한 집합이고
2.  $\Sigma$ 는 알파벳이고
3. 전이함수  $\delta$ 는  $Q \times \Sigma$ 에서  $Q$ 로의 함수이고
4.  $q_0 \in Q$ 는 초기상태이고
5.  $F \subseteq Q$ 는 최종상태들의 집합이다.

결정 유한 오토마타(deterministic finite automata)를 줄여서 DFA로 부르기로 하자.

전이 함수  $\delta$ 를 확장하여  $Q \times \Sigma^*$ 에서  $Q$ 로의 함수  $\delta^*$ 를 다음과 같이 정의한다.

- $\delta^*(q, \epsilon) = q$
- 모든  $w \in \Sigma^*$ 와  $a \in \Sigma$ 에 대하여,  $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$

스트링  $w \in \Sigma^*$ 에 대하여  $\delta^*(q_0, w) \in F$  DFA  $M$ 의 최종상태이면,  $M$ 이  $w$ 를 받아들인다고 말한다.  $M$ 의 언어  $L(M)$ 은  $M$ 이 받아들이는 모든 스트링의 집합이다. 즉

$$L(M) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}.$$

DFA의 상황(configuration)은 현재 상태와 입력 스트링의 읽지 않은 부분으로 결정된다. 입력 스트링이  $w$ 일 때, DFA의 시작 상황은  $(q_0, w)$ 이고, 종료 상황은 어떤  $q \in Q$ 에 대하여  $(q, \epsilon)$ 이다.

한 번의 전이에 의해서 DFA  $M$ 의 상황이 변화하는 과정을  $\vdash_M$ 으로 표시한다. 즉 현재 상황이  $(q, 10110)$ 이고  $\delta(q, 1) = q'$ 이면,

$$(q, 10110) \vdash_M (q', 0110)$$

이다. DFA  $M$ 이 명확한 경우  $\vdash_M$  대신  $\vdash$ 를 사용할 수 있다. 0번 이상의 전이를  $\vdash^*$ 로 표시한다.

**예제 2.9**  $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_0\})$  이고,  $\delta$ 는 다음과 같다.

	0	1
$q_0$	$q_0$	$q_1$
$q_1$	$q_1$	$q_0$

$L(M)$ 은 짝수 개의 1을 가진 스트링의 집합이다.

유한 오토마타를 이해하기 쉽도록 그림으로 나타낸다. 상태는 정점으로, 전이  $\delta(p, a) = q$ 는 상태  $p$ 에서  $q$ 로 가는 라벨  $a$ 가 붙은 간선으로, 최종상태는 이중원으로, 초기상태는 화살표로 표시한다. 예제 2.9의 DFA를 그림으로 나타내면 그림 2.2와 같다.

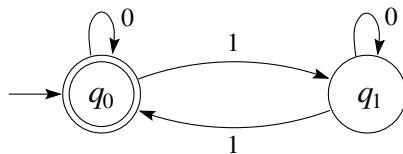


Figure 2.2:

일반적으로 다음 두 단계에 의해 DFA를 만든다.

1. 현재까지 읽은 부분에서 어떤 정보를 기억해야 하는지를 정하고 이 정보를 상태로 표시한다.

2. 새로운 글자를 읽었을 때 기억해야 하는 정보가 어떻게 바뀌는지를 보고 전이함수를 정한다.

이 중에서 중요한 것은 기억해야 하는 정보를 정하는 것이다. 예제 2.9에서는 1의 개수가 짝수인지 홀수인지를 기억해야 한다. 따라서 상태  $q_0$ 은 1의 개수가 짝수,  $q_1$ 은 1의 개수가 홀수임을 나타낸다.

**예제 2.10**  $L = \{w : N_0(w) \bmod 2 > N_1(w) \bmod 2\}$ 을 받아들이는 DFA를 구하라. 여기에서  $N_a(w)$ 는  $w$ 에 있는  $a$ 의 개수를 나타낸다. 즉, 이 DFA는 0의 개수는 홀수이고 1의 개수는 짝수인 스트링을 받아들여야 된다.

이 경우  $N_0(w) \bmod 2$ 와  $N_1(w) \bmod 2$ 를 기억하면 된다 (그림 2.3).

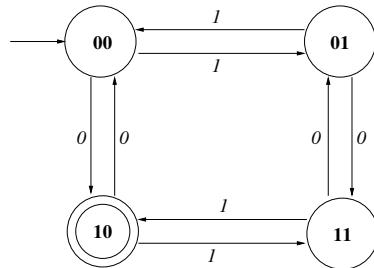


Figure 2.3:

**예제 2.11** 0101을 부분스트링으로 갖는 스트링을 받아들이는 DFA를 구하라.

이 문제에서는 0101의 어두 중에서 현재까지 읽은 부분을 기억해야 한다. 따라서 0101의 어두를 상태로 표시하는 DFA를 만들면 된다 (그림 2.4). 이제 전이함수를 정한다. 예를 들면, 현재까지 읽은 어두가 0이고, 그 다음에 1을 읽으면 읽은 어두가 01이 되고, 0을 읽으면 읽은 부분이 00이 되는데 0101의 어두에 해당되는 것은 0이다.

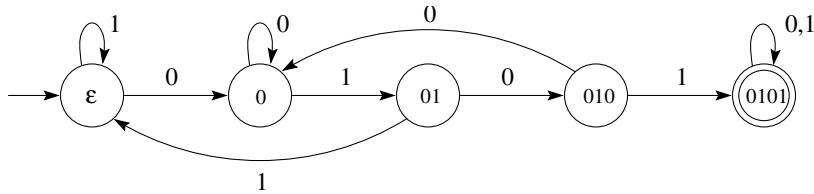


Figure 2.4:

**예제 2.12** 0101을 부분스트링으로 갖지 않는 스트링을 받아들이는 DFA를 구하라.

그림 2.4의 DFA에서 최종상태와 그 외의 상태를 맞바꾸면 된다. (이와 같이 여집합을 표시하는 문제가 정규식에서는 어려웠는데, DFA에서는 쉬운 것을 주목하라.)

**예제 2.13** 스트링에서 연속(run)이란 같은 글자로만 구성된 (더 이상 늘릴 수 없는) 부분스트링을 말한다. 예를 들면, 0110001은 길이가 2인 1의 연속과 길이가 3인 0의 연속을 가지고 있다. 길이가 2인 연속을 갖지 않는 스트링을 받아들이는 DFA를 구하라. 즉 스트링이 길이가 2인 연속은 갖지 않아야 하고 길이가 1이나 3이상인 연속은 가질 수 있다.

이 문제에서는 가장 최근의 연속을 기억해야 하고 그 중에서 길이가 3이하인 연속을 기억하면 된다.

**예제 2.14** 010을 부분스트링으로 가지고, 100을 부분스트링으로 갖지 않는 스트링을 받아들이는 DFA를 구하라.

이 문제는 약간 어렵다. 010이나 100이 부분스트링으로 있는지 없는지를 알기 위해서는 앞의 예제에서처럼 010과 100의 어두를 기억해야 한다. 동시에 현재 100을 읽고 있다면 이전에 010이 있었는지 없었는지를 알고 있어야 한다. 반대로 현재 010을 읽고 있을 때 이전에 100이 있었는지를 기억할 필요는 없다. 왜냐하면 100이 있었다면 이 스트링은 받아들여지지 않기 때문이다.

$\delta$ 가 부분함수일 때도 DFA로 볼 수 있다. 이 경우 위의 정의에 의한 DFA로 바꾸려면, 새로운 상태  $q' \notin F$ 을 도입하여

- 정의되지 않은 모든 전이는  $q'$ 으로 가도록 하고,
- 모든  $a \in \Sigma$ 에 대하여  $\delta(q', a) = q'$ 이 되도록 한다.

$q'$ 과 같이 한 번 들어가면 빠져 나올 수 없는 상태를 죽은 상태(dead state)라고 부른다. 예제 2.12에서 상태 0101도 죽은 상태이다.

### 2.3 비결정 유한 오토마타

비결정 유한 오토마타(nondeterministic finite automata)를 줄여서 NFA로 부르기로 하자. NFA는 다음 두 가지 면에서 DFA를 일반화한 것이다.

1. 각 상태와 글자가 주어졌을 때, 다음 상태가 없거나 하나 이상 일 수 있다.
2. 각 상태에서 공스트링  $\epsilon$ 에 의한 전이가 있을 수 있다.

**정의 2.5** 비결정 유한 오토마타  $M$ 은 다섯 가지 요소  $(Q, \Sigma, \Delta, q_0, F)$ 로 구성된다. 여기에서  $Q, \Sigma, q_0, F$ 는 DFA에서와 같이 정의된다.  $\Delta$ 는  $Q \times (\Sigma \cup \{\epsilon\}) \times Q$ 의 부분집합인 전이 관계로, 또는  $Q \times (\Sigma \cup \{\epsilon\})$ 에서  $2^Q$ 로의 전이 함수로 정의할 수 있다.

NFA를 정의할 때는  $\Delta$ 를 DFA의 전이 함수에서 일반화된 전이 관계로 파악하는 것이 자연스러우나, 설명에서는  $\Delta$ 를 전이 함수로 보는 것이 편리하므로 이후에는  $\Delta$ 를  $2^Q$ 으로의 함수로 간주한다. 또한 설명의 편의를 위해서  $\Delta$ 를  $2^Q \times (\Sigma \cup \{\epsilon\})$ 에서의 함수로 다음과 같이 확장한다.  $P \subseteq Q$ 와  $a \in \Sigma \cup \{\epsilon\}$ 에 대하여

$$\Delta(P, a) = \bigcup_{q \in P} \Delta(q, a).$$

임의의 상태  $q$ 에 대하여,  $E(q)$ 를  $q$ 에서부터 입력 글자를 읽지 않고 ( $\epsilon$  라벨이 붙은 간선을 따라) 도달할 수 있는 상태의 집합이라고 하자.  $E$ 도 역시 확장하여, 상태 집합  $P$ 에 대하여  $E(P) = \bigcup_{q \in P} E(q)$ .

$\Delta$ 를 확장하여  $Q \times \Sigma^*$ 에서  $2^Q$ 으로의 함수  $\Delta^*$ 를 다음과 같이 정의한다.

- $\Delta^*(q, \epsilon) = E(q)$
- 모든  $w \in \Sigma^*$ 와  $a \in \Sigma$ 에 대하여,  $\Delta^*(q, wa) = E(\Delta(\Delta^*(q, w), a))$

스트링  $w \in \Sigma^*$ 에 대하여  $\Delta^*(q_0, w)$ 이 NFA  $M$ 의 최종상태를 적어도 하나 포함하면,  $M$ 이  $w$ 를 받아들인다고 말한다.  $M$ 의 언어  $L(M)$ 은  $M$ 이 받아들이는 모든 스트링의 집합이다. 즉

$$L(M) = \{w \in \Sigma^* : \Delta^*(q_0, w) \cap F \neq \emptyset\}.$$

또는 전이과정을 사용하여  $L(M)$ 을 다음과 같이 정의할 수 있다.

$$L(M) = \{w \in \Sigma^* : (q_0, w) \vdash_M^* (q, \epsilon), q \in F\}.$$

일반적으로 NFA는 DFA보다 훨씬 쉽게 언어를 표현할 수 있다.

**예제 2.15** 정규식  $(010 + 01)^*$ 가 표시하는 언어를 받아들이는 NFA를 구하라.

그림 2.5을 보라. 입력 스트링이 010일 때,

- $\Delta^*(q_1, 0) = \{q_2\}$ ,
- $\Delta^*(q_1, 01) = \{q_3, q_1\}$ ,
- $\Delta^*(q_1, 010) = \{q_1, q_2\}$

이고, 여기에 최종상태  $q_1$ 이 포함되어 있으므로 이 NFA는 010을 받아들인다. 이것을 전이 과정으로 나타내면 다음과 같다.

$$(q_1, 010) \vdash (q_2, 10) \vdash (q_3, 0) \vdash (q_1, \epsilon)$$

즉  $(q_1, 010) \vdash^* (q_1, \epsilon)$ 이고  $q_1$ 이 최종상태이므로 이 NFA는 010을 받아들인다.

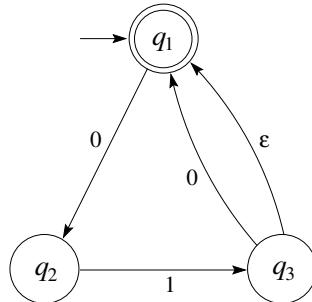


Figure 2.5:

**예제 2.16** 끝에서 두 번째 글자가 1인 스트링을 받아들이는 NFA를 구하라. 이 NFA는  $q_1$ 에 머물다가 끝에서 두 번째 글자를 추측하여 그 글자가 1이면  $q_2$ 로 이동한다.  $q_3$ 는 이 추측이 맞았는지 확인한다. 그림 2.6을 보라. 입력 스트링이 011일 때,

- $\Delta^*(q_1, 0) = \{q_1\}$ ,
- $\Delta^*(q_1, 01) = \{q_1, q_2\}$ ,
- $\Delta^*(q_1, 011) = \{q_1, q_2, q_3\}$

이므로, 이 NFA는 011을 받아들인다. 입력 스트링이 100일 때,

- $\Delta^*(q_1, 1) = \{q_1, q_2\}$ ,
- $\Delta^*(q_1, 10) = \{q_1, q_3\}$ ,
- $\Delta^*(q_1, 100) = \{q_1\}$

이므로, 이 NFA는 100을 받아들이지 않는다.

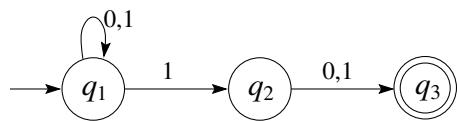


Figure 2.6:

NFA의 최종상태는 한 개라고 가정할 수 있다. NFA의  $Q$ 에 두 개 이상의 최종상태가 있으면, 새로운 상태  $q_f$ 만을 최종상태로 하고 모든  $q \in F$ 에 대해 전이  $(q, \epsilon, q_f)$ 를 더해준다.

왜 비결정성(nondeterminism)이 필요한가? 현재 사용되는 디지털 컴퓨터는 완전히 결정적으로 움직인다. 또 미래에 비결정적으로 작동되는 컴퓨터를 만들 가능성은 적다 (현재 quantum computer, DNA computing 등이 시도되고 있지만). 그럼에도 불구하고, 다음과 같은 이유로 비결정성은 중요한 역할을 한다.

- 비결정성을 이용하면 오토마타를 쉽게 만들 수 있다. 어떤 언어를 받아들이는 NFA를 만드는 것이 일반적으로 DFA를 만드는 것보다 쉽다.
- 문제의 세계와 문제를 푸는 도구의 세계를 생각해보자. 문제의 세계는 우리의 문제해결 능력과 관계없이 자연에 주어져 있는 것이다. 이에 비해 디지털 컴퓨터 등은 우리가 가지고 있는 문제해결도구이다. 현재 우리가 가지고 있는 문제해결도구는 결정적으로 움직이지만, 문제의 세계에는 비결정성이 이미 존재한다 (뒤에 나오는  $NP$ 완전 문제들). 따라서 문제의 세계를 이해하기 위해서는 비결정성이라는 개념이 필요하다.

## 2.4 유한 오토마타의 동등성

DFA가 받아들이는 언어 종류와 NFA가 받아들이는 언어 종류가 같음을 보이고자 한다. DFA는 NFA의 일종이므로 DFA의 언어 종류는 NFA의 언어 종류의 부분집합이다. 이제 반대의 경우를 증명하자.

**정리 2.1** 임의의 NFA에 대하여 이와 동등한 DFA가 존재한다.

**증명.**  $N = (Q_N, \Sigma, \Delta, q_0, F_N)$ 을 임의의 NFA라고 하자. 이제  $N$ 과 동등한 DFA  $D = (Q_D, \Sigma, \delta, q', F_D)$ 를 만들고자 한다. 증명의 핵심은 어느 시점에서 NFA의 현재 상태들의 집합을 DFA의 하나의 상태로 간주하는 것이다. 즉 DFA  $D$ 의 상태는  $Q_N$ 의멱집합의 원소이다.

DFA  $D$ 의 초기 상태  $q'$ 는  $E(q_0)$ 이다.  $D$ 의 상태  $P \subseteq Q_N$ 와 글자  $a \in \Sigma$ 에 대하여,  $D$ 의 전이 함수는  $\delta(P, a) = E(\Delta(P, a))$ 이다. 이를 이용하여 다음과 같이 DFA  $D$ 를 구한다.

```

 $Q_D \leftarrow \{E(q_0)\}$ 
mark  $E(q_0)$ 
while  $\exists$  marked state  $P \in Q_D$  do
    unmark  $P$ 
    for each  $a \in \Sigma$  do
         $R \leftarrow E(\Delta(P, a))$ 
        if  $R$  is not in  $Q_D$  then
            add  $R$  as marked state to  $Q_D$  fi
             $\delta(P, a) \leftarrow R$ 
        od
    od

```

마지막으로,  $D$ 의 상태  $P$ 가  $F_N$ 의 원소를 적어도 하나 포함하면  $P$ 는  $F_D$ 의 원소이다.

이제 NFA  $N$ 과 DFA  $D$ 가 동등함을 보이자. 입력 스트링의 길이에 대한 귀납법으로

$$\Delta^*(q_0, w) = \delta^*(E(q_0), w) \quad (2.1)$$

임을 증명한다.

1.  $|w| = 0$ 일 때,  $\Delta^*(q_0, \epsilon) = E(q_0)$  이고  $\delta^*(E(q_0), \epsilon) = E(q_0)$  이므로 (2.1)가 성립한다.
2.  $|w| < k$ 일 때, (2.1)가 성립한다고 가정하자.
3.  $|w| = k \geq 1$ 일 때,  $w = ua$  ( $u \in \Sigma^*, a \in \Sigma$ ) 라고 하자. 정의에 의해 NFA  $N$ 에서는  $\Delta^*(q_0, w) = E(\Delta(\Delta^*(q_0, u), a))$  이고, DFA  $D$ 에서는  $\delta^*(E(q_0), w) = \delta(\delta^*(E(q_0), u), a) = E(\Delta(\delta^*(E(q_0), u), a))$  이다. 귀납법 가정에 의해  $\Delta^*(q_0, u) = \delta^*(E(q_0), u)$  이므로 (2.1) 가 성립한다.

따라서  $\Delta^*(q_0, w)$ 가  $F_N$ 의 원소를 적어도 하나 포함하는 경우에만  $\delta^*(E(q_0), w) \in F_D$ 이다. 즉  $L(N) = L(D)$ .  $\square$

**예제 2.17** 예제 2.16의 NFA와 동등한 DFA를 구하라.

그림 2.7을 보라. 이 DFA를 직접 구하기 위해서는 DFA가 읽은 이전 두 글자를 기억해야 한다. 이전 두 글자는 00, 01, 10, 11 중 하나이므로 이들을 상태로 나타내면 된다. 그림 2.7에서 상태 {1}이 00, {1, 2}가 01, {1, 3}이 10, {1, 2, 3}이 11이 된다.

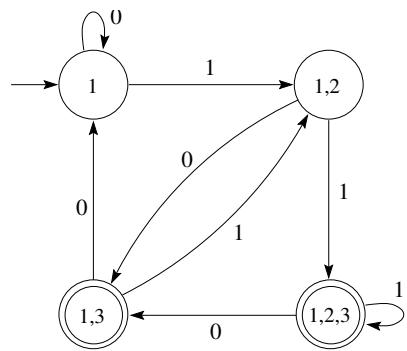


Figure 2.7:

**예제 2.18** 예제 2.15의 NFA와 동등한 DFA를 구하라.

그림 2.8을 보라. 이 DFA에서 공집합을 나타내는 상태는 (항상) 죽은 상태이다.

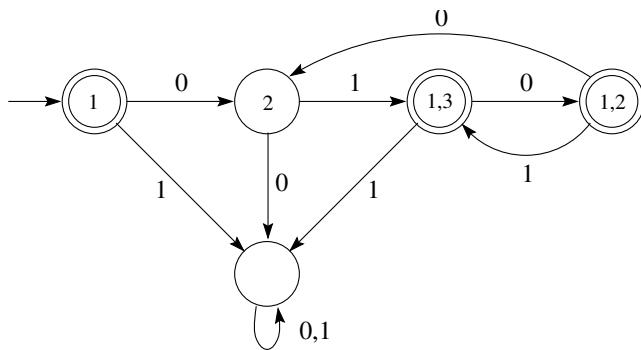


Figure 2.8:

## 2.5 정규식과 유한 오토마타

앞 장에서 DFA의 언어 종류와 NFA의 언어 종류가 같음을 보았는데, 이제 이 언어 종류가 정규언어 종류임을 보이고자 한다.

**정리 2.2**  $r$ 을 정규식이라고 하면,  $L(r)$ 을 받아들이는 NFA가 존재한다.

**증명.** 정규식은 정의 2.1에 의해 구성되므로 각 경우에 대하여 동등한 NFA가 있음을 보이면 된다. 이 과정에서 만들어지는 NFA는 항상 다음 조건을 만족한다.

- 최종상태는 하나이다.
- 초기상태로 들어가는 전이와 최종상태에서 나가는 전이가 없다.

정규식  $\emptyset, \epsilon, a \in \Sigma$ 와 동등한 NFA는 그림 2.9과 같다. 정규식  $r, s$  와 동등한 NFA를 각각  $R, S$  라고 하면  $(r + s), (rs), (r^*)$ 와 동등한 NFA는 그림 2.10, 2.11, 2.12과 같다.  $\square$

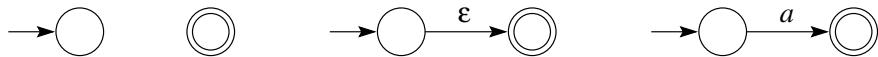


Figure 2.9:

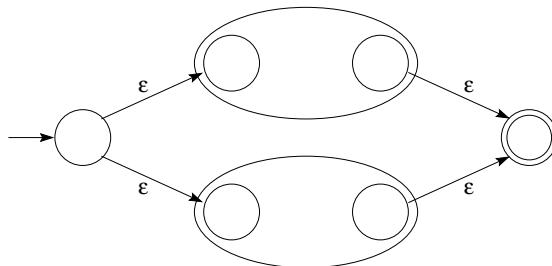


Figure 2.10:

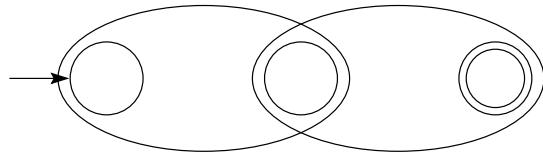


Figure 2.11:

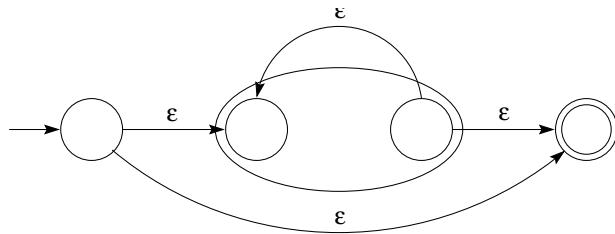


Figure 2.12:

**예제 2.19** 정규식  $(0 + 11)^*$ 와 동등한 NFA를 구하라. 그림 2.13을 보라.

**정리 2.3** DFA  $M$ 의 언어  $L(M)$ 에 대하여,  $L(M)$ 을 표시하는 정규식이 있다.

**증명.**  $M = (\{q_1, \dots, q_n\}, \Sigma, \delta, q_1, F)$ 라고 하자.  $R_{ij}^k$ 를  $k$  번호를 가진 상태만을 지나면서  $M$ 을  $q_i$ 에서  $q_j$ 로 전이시키는 스트링의 집합이라고 하자. 모든 상태는  $n$ 이하이므로,  $R_{ij}^n$ 은  $q_i$ 에서  $q_j$ 로 전이시키는 모든 스트링의 집합이다. 따라서

$$L(M) = \bigcup_{q_j \in F} R_{1j}^n.$$

$R_{ij}^k$ 는 다음과 같이 재귀적으로 구해질 수 있다.

$$\begin{aligned} R_{ij}^0 &= \begin{cases} \{a : \delta(q_i, a) = q_j\} & \text{if } i \neq j \\ \{a : \delta(q_i, a) = q_j\} \cup \{\epsilon\} & \text{if } i = j \end{cases} \\ R_{ij}^k &= R_{ij}^{k-1} \cup R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1} \end{aligned} \quad (2.2)$$

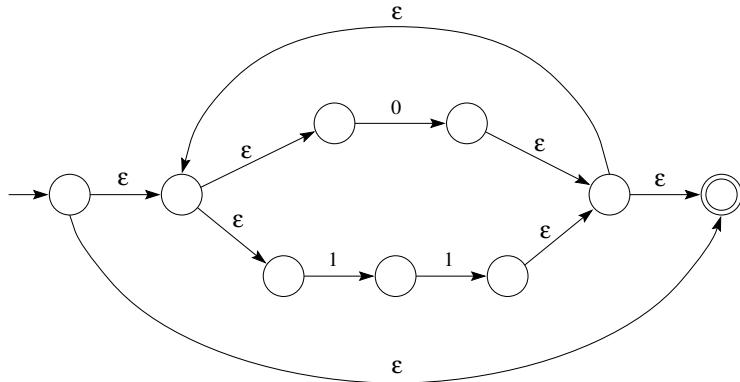


Figure 2.13:

식 2.2의 의미는  $k$ 이하의 상태를 지나면서  $q_i$ 에서  $q_j$ 로 가기 위해 서는

1.  $k - 1$ 이하의 상태를 지나면서  $q_i$ 에서  $q_j$ 로 가든지
2. 각 경우에  $k - 1$ 이하의 상태를 지나면서,  $q_i$ 에서  $q_k$ 로 가고  $q_k$ 에서  $q_k$ 로 0번 이상 순환하고  $q_k$ 에서  $q_j$ 로 가면 된다는 것이다.

식 2.2에서  $R_{ij}^k$ 는  $\ast, \cdot, \cup$  연산에 의해 표시되므로 정의 2.1에 의해  $R_{ij}^k$  집합을 표시하는 정규식이 있음을  $k$ 에 대한 귀납법으로 증명할 수 있다.  $L(M)$ 은 각  $q_j \in F$ 에 대하여  $R_{1j}^n$ 의 합집합이므로, 이에 대한 정규식도 역시 존재한다.  $\square$

**예제 2.20** 예제 2.18의 DFA  $M$ 과 동등한 정규식을 구하라.  $M$ 에서  $q_1 = \{1\}$ ,  $q_2 = \{2\}$ ,  $q_3 = \{1, 3\}$ ,  $q_4 = \{1, 2\}$  라고 하자.  $\emptyset$ 은 죽은 상태이므로 고려하지 않아도 된다. 이후의 전개에서 다음 식을 이용한다.

$$\begin{aligned}
 (st)^{\ast}s &= (\epsilon + st + stst + \dots)s \\
 &= s + sts + ststs + \dots \\
 &= s(\epsilon + ts + tsts + \dots) \\
 &= s(ts)^{\ast}
 \end{aligned}$$

집합  $R_{ij}^k$ 를 나타내는 정규식을  $r_{ij}^k$ 라고 하자 ( $\vdash R_{ij}^k = L(r_{ij}^k)$ ). 식 2.2에 의해, 구하는 정규식은

$$r = r_{11}^4 + r_{13}^4 + r_{14}^4$$

이때  $r_{11}^4 = \epsilon$ 임을 쉽게 알 수 있다.

$$\begin{aligned} r_{13}^4 &= r_{13}^3 + r_{14}^3(r_{44}^3)^*r_{43}^3 \\ &= 01 + 010(\epsilon + 010 + 10)^*(01 + 1) \\ &= 01 + 010((01 + 1)0)^*(01 + 1) \\ &= 01 + 01(001 + 01)^*(001 + 01) \\ &= 01(001 + 01)^* \end{aligned}$$

마찬가지로  $r_{14}^4 = 010(010 + 10)^*$  이다. 따라서

$$\begin{aligned} r &= \epsilon + 01(001 + 01)^* + 010(010 + 10)^* \\ &= \epsilon + (010 + 01)^*01 + (010 + 01)^*010 \\ &= (010 + 01)^* \end{aligned}$$

## 2.6 정규언어의 성질

**정리 2.4** 정규언어 종류는 다음 연산에 대하여 닫혀 있다: (1) 합집합, (2) 접합, (3) Kleene 곱, (4) 여집합, (5) 교집합.

**증명.** 정규식의 정의에 의해 정규언어는 합집합, 접합, Kleene 곱에 대해 닫혀 있다.

(4) 정규언어  $L$ 을 받아들이는 DFA를  $(Q, \Sigma, \delta, q, F)$ 라고 하자. 그러면 DFA  $(Q, \Sigma, \delta, q, Q - F)$ 가 여집합  $\overline{L} = \Sigma^* - L$ 을 받아들인다.

(5) 다음 식에 의해 교집합에 대하여 닫혀 있다.

$$L_1 \cap L_2 = \overline{\overline{L}_1 \cup \overline{L}_2}$$

□

정규언어에 대하여 여러 가지 질문을 던질 수 있다. 다음의 질문에 대하여 답하는 방법(알고리즘)이 있다.

**예제 2.21 (소속문제)** 정규언어  $L$ 과 스트링  $w$ 가 주어졌을 때,  $w$ 가  $L$ 에 속하는지 결정하라.

$L$ 이 정규식으로 주어지면 그와 동등한 유한 오토마타로 바꾼 후,  $w$ 를 읽어서 받아들이는지 결정한다.

**예제 2.22** 유한 오토마타  $M$ 이 주어졌을 때,  $L(M) = \emptyset$  인지 결정하라.

$M$ 을 DFA로 바꾼 후, 초기상태에서 최종상태로 가는 경로가 있는지 깊이 우선 탐색(depth-first search)이나 너비 우선 탐색(breadth-first search)를 이용하여 확인한다.

**예제 2.23** 두 정규언어  $L_1, L_2$ 가 주어졌을 때,  $L_1 = L_2$  인지 결정하라.

$L_1 = L_2$ 를 증명하기 위해서는  $L_1 \subseteq L_2$ 와  $L_2 \subseteq L_1$ 을 보이면 된다.  $L_1 \subseteq L_2$ 는  $L_1 \cap \overline{L_2} = \emptyset$ 과 같으므로,  $L_1 \cap \overline{L_2}$ 를 받아들이는 DFA  $M$ 을 만든 후  $L(M) = \emptyset$  인지 확인한다.

어떤 언어가 정규언어가 아님을 보이기 위해서는 다음 정리를 사용한다.

**정리 2.5 (펌프 정리)**  $L$ 을 무한 정규언어라고 하면, 다음을 만족하는 양의 정수  $t$ 가 존재한다. 길이가  $t$  이상인 임의의 스트링  $w \in L$ 은  $w = xyz$ 로 표현되며 여기서

1.  $|xy| \leq t$ ,
2.  $|y| \geq 1$ ,
3. 모든  $i \geq 0$ 에 대하여  $xy^i z \in L$ 이다.

**증명.**  $L$ 이 정규언어이므로  $L$ 을 받아들이는 DFA  $M = (Q, \Sigma, \delta, q_0, F)$ 가 존재한다.  $M$ 의 상태의 갯수를  $t$ 라 하자.  $w = a_1a_2 \cdots a_n$  ( $n \geq t$ )에 대하여  $\delta^*(q_0, a_1 \cdots a_i) = q_i$ 라고 하자.  $M$ 은  $t$  개의 상태만을 가지므로, 처음  $t + 1$  개의 상태  $q_0, q_1, \dots, q_t$ 가 전부 다를 수는 없다. 즉  $q_j = q_k$ 인 두 수  $j, k$  ( $0 \leq j < k \leq t$ )가 존재한다.

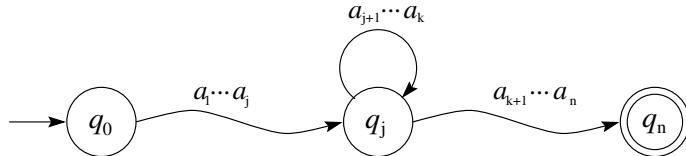


Figure 2.14:

$x = a_1 \cdots a_j$ ,  $y = a_{j+1} \cdots a_k$ ,  $z = a_{k+1} \cdots a_n$ 이라고 하면,  $|xy| \leq t$ 이고  $|y| \geq 1$ 이다.  $w \in L$ 이므로  $q_n \in F$ 이다. 따라서 모든  $i \geq 0$ 에 대해서  $\delta^*(q_0, xy^i z) = q_n$ 이므로  $xy^i z \in L$ 이다. 즉  $L$ 에 속한 충분히 긴 스트링은 세 부분으로 나눌 수 있는데, 중간 부분인  $y$ 를 몇 번 펌프질 하든지 그 결과는 역시  $L$ 에 속한다는 의미로 펌프 정리라는 용어를 사용한다.  $\square$

펌프 정리의 역은 성립하지 않는다.

펌프 정리를 적용하여 언어  $L$ 이 정규언어가 아님을 보이고자 할 때는 귀류법을 사용한다. 즉  $L$ 이 정규언어라고 가정하고, 펌프 정리에 모순됨을 보이면 된다. 펌프 정리 중  $t$ 와  $xyz$ 는 ‘존재한다’는 조건을 갖고,  $w$ 와  $i$ 는 ‘모든’의 조건을 가지므로 다음과 같이 증명을 전개한다. ( $t$ 와  $xyz$ 는 존재한다는 사실만 알고 실제 값을 모르므로 어떤 값에 대해서든지 증명할 수 있어야 되고,  $w$ 와  $i$ 는 모든 값에 대해서 성립하므로 증명에 가장 유리한 값을 선택하면 된다.)

1. 상대방(adversary)이  $t$ 를 선택한다.
2. 길이가  $t$  이상인  $w \in L$ 를 선택한다.
3. 상대방이  $w$ 를  $|xy| \leq t$ ,  $|y| \geq 1$  조건을 만족시키는  $x, y, z$ 로 나눈다.

4.  $i$ 를 선택하여  $xy^iz$ 가  $L$ 에 속하지 않음을 보인다.

이 과정 중에서 가장 중요한 것은 2 단계로서  $w$ 를 선택할 때, 3 단계에서 상대방이  $x, y, z$ 를 선택할 여지가 적어지도록 하여야 한다.

**예제 2.24**  $L = \{0^n 1^n : n \geq 0\}$ 은 정규언어가 아님을 증명하라.

$L$ 이 정규언어라고 가정하자. 그러면 정리 2.5를 만족하는 양의 정수  $t$ 가 존재한다. 스트링  $w = 0^t 1^t \in L$ 을 고려하자.  $|xy| \leq t$  이므로  $y$ 는  $0^k$  ( $1 \leq k \leq t$ ) 이어야 한다. 그러면  $i = 0$ 일 때  $0^{t-k} 1^t$ 은  $L$ 에 속하지 않으므로 정리 2.5에 모순된다.

**예제 2.25**  $L = \{uu : u \in \{0, 1\}^*\}$ 는 정규언어가 아님을 증명하라.

**예제 2.26**  $L = \{a^{n^2} : n \geq 1\}$ 는 정규언어가 아님을 증명하라.

**예제 2.27**  $L = \{0^m 1^n : m \neq n\}$ 은 정규언어가 아님을 증명하라.

$L$ 이 정규언어라고 가정하면, 정리 2.5를 만족하는  $t$ 가 존재한다. 이 경우  $w = 0^t 1^{t+1}$ 을 선택하면, 상대방은  $y = 00$ 를 선택하여 정리 2.5가 만족되므로 증명에 실패하게 된다. 따라서 상대방이 어떤  $y = 0^k$ 를 선택하든지 펌프질 하다보면 1의 개수와 같아지도록  $w$ 를 잡아야 된다. 즉  $w = 0^t 1^{(t+1)!}$ 을 선택한다.  $|xy| \leq t$  이므로  $y = 0^k$  ( $1 \leq k \leq t$ ) 이어야 한다. 그러면

$$i = \frac{tt!}{k} + 1$$

일 때,  $xy^iz = 0^t 0^{k(i-1)} 1^{(t+1)!} = 0^{(t+1)!} 1^{(t+1)!}$  이므로  $L$ 에 속하지 않는다.

보다 간단한 증명은 다음과 같다.  $L$ 이 정규언어라고 가정하자. 그러면  $\overline{L} \cap L(0^* 1^*)$ 는 정규언어이다. 그런데 이 언어가  $\{0^n 1^n : n \geq 0\}$  이므로 모순이다.

## 2.7 유한 오토마타의 응용

유한 오토마타는 논리회로 설계에 광범위하게 사용된다.

**예제 2.28** 패리티 검사기(*parity checker*)는 0, 1로 구성된 스트링을 읽어서 1의 개수를 검사하는 회로이다. 짝수(홀수) 패리티 검사기는 1의 개수가 짝수(홀수)일 때 받아들이는 회로이다. 예제 2.9의 DFA는 짝수 패리티 검사기를 나타낸다. 이 DFA에 대한 하드웨어 구현은 [Ka]를 참고하라.

**예제 2.29** 과자 한 봉지에 200원 받는 자동판매기가 있는데, 이것은 100원이나 50원 동전을 받아서 200원이 되면 과자 한 봉지를 내보낸다. 100원 동전을  $a$ 로, 50원 동전을  $b$ 로 표시할 때, 이 자동판매기의 제어기를 설계하라.

그림 2.15를 보라. 이 그림에서 150원 받은 상태에서 100원이 주어지는 경우가 없음에 유의하라. (즉 전이함수가 부분함수이다.) 이 제어기는 정확하게 200원이 주어졌을 때만 작동한다. 거스름돈을 내어주기 위해서는 제어기가 더 복잡해져야 된다.

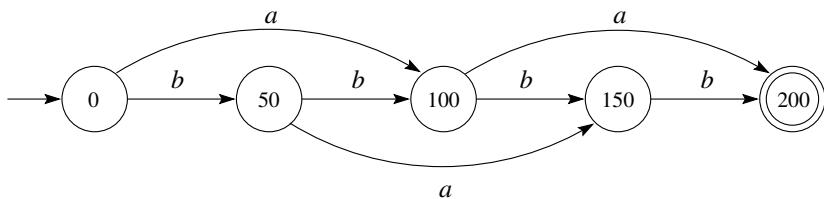


Figure 2.15:

정규식을 유한 오토마타로 변환하는 것도 여러 곳에 응용된다.

**예제 2.30** Pascal 언어에서 변수 이름은 한 개의 영문자 뒤에 영문자 또는 숫자를 반복적으로 붙인 것이므로,

$$(letter)(letter + digit)^*$$

로 표시할 수 있다. 여기에서 *letter*는  $A + B + \dots + Z$ 를 나타내고 *digit*은  $0 + 1 + \dots + 9$ 를 나타낸다. FORTRAN 언어에서 변수 이름은 Pascal 언어와 마찬가지이나 길이가 최대 6이므로

$$(letter)(\epsilon + letter + digit)^5$$

으로 표시할 수 있다. 컴파일러는 위의 정규식을 유한 오토마타로 바꾸어서 프로그램을 읽을 때 변수 이름을 인식한다.

**예제 2.31** Unix의 egrep 명령어는 다음과 같은 형태로 사용된다.

```
egrep 'exp' file
```

여기에서 *exp*는 정규식이고 *file*은 파일의 이름이다. Unix에서 정규식  $r + s, rs, r^*, r^+, r + \epsilon$ 는 각각  $r|s$ ,  $rs$ ,  $r^*$ ,  $r^+$ ,  $r?$ 로 표시된다. 이 명령어는 *file*의 줄들을 차례로 읽으면서 각 줄에 *exp*에 속한 스트링이 포함되어 있으면 그 줄을 출력한다. 예를 들면

```
egrep '(aa|bb) +' file
```

에서 *file*이 ababab, aaaa, ccbbaacc 세 줄로 구성되어 있으면 이 명령어는 aaaa와 ccbbaacc를 출력한다. 이 경우 *exp*에 속한 스트링 *w*가 각 줄의 부분스트링이 될 수 있으므로, *w* 앞에 어떤 것이든지 붙을 수 있도록 정규식  $\Sigma^*exp$ 를 고려하여야 한다. egrep은  $\Sigma^*exp$ 를 NFA로 바꾼 후 각 줄을 읽다가 최종상태에 도달하면 그 줄을 출력한다.

## 2.8 유한 오토마타의 최소화

DFA를 이용하여 논리회로를 설계할 때, 가능한 한 DFA의 상태 개수를 적게 하는 것이 필요하다. 본 절에서는 정규언어 *L*이 주어졌을 때 *L*을 받아들이는, 상태의 개수가 최소인 DFA를 찾는 방법을 기술

한다.

먼저  $L$ 을 받아들이는 임의의 DFA  $M = (Q, \Sigma, \delta, q_0, F)$ 를 만든다. 물론 초기상태에서 도달할 수 없는 상태들은 쉽게 제거할 수 있다. 따라서  $M$ 의 모든 상태들은 초기상태에서 도달할 수 있다고 가정한다.

**예제 2.32** 그림 2.16의 DFA를 고려하자.

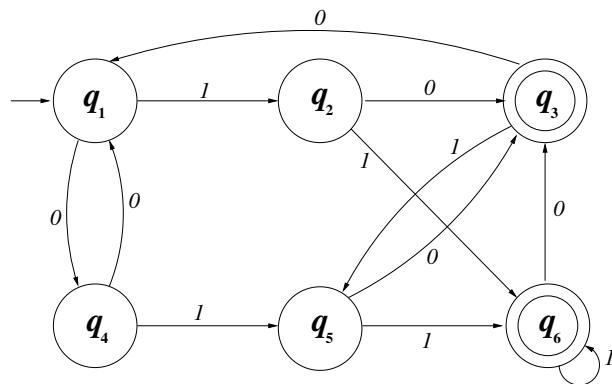


Figure 2.16:

이제  $M$ 의 어떤 상태들을 하나의 상태로 묶을 수 있는지 찾고자 한다. 이를 위해  $M$ 의 상태들  $p, q \in Q$ 에 대하여 다음의 관계를 정의한다.

모든 스트링  $x \in \Sigma^*$ 에 대하여  $\delta^*(p, x) \in F$ 인 경우에만  
 $\delta^*(q, x) \in F$ 일 때  $p \sim q$ 라고 정의한다.

즉  $p$ 와  $q$  이후에 어떤 스트링이 오든지 결국 두 경우 다  $M$ 이 받아들이든지 두 경우 다  $M$ 이 받아들이지 않기 때문에  $p$ 와  $q$ 는 한 상태로 묶을 수 있다는 것이다. 그림 2.16에서  $q_2 \sim q_5$ 임을 쉽게 확인할 수 있다.

먼저  $\sim$ 은 정의에 의해서 반사적이고 대칭적이고 이행적이므로 동치관계이다. 따라서  $M$ 의 상태들이  $\sim$ 에 의해 동치류들로 분할되고 한 동치류에 속한 상태들을 한 상태로 묶으려고 한다.

$\sim$ 의 동치류들을 단계적으로 구하기 위해  $\sim_n$ 을 정의한다. 길이가  $n$  이하인 모든 스트링  $x$ 에 대하여  $\delta^*(p, x) \in F$ 인 경우에만  $\delta^*(q, x) \in F$ 일 때  $p \sim_n q$ 이다. 이제  $\sim_0$ 의 동치류들,  $\sim_1$ 의 동치류들을 차례로 구하다가  $\sim_n$ 의 동치류들과  $\sim_{n+1}$ 의 동치류들이 동일하면 이것이 ( $\sim$ 과  $\sim_n$ 의 정의에 의해)  $\sim$ 의 동치류들이 된다.

$\sim_0$ 의 동치류는  $F$ 와  $Q - F$ 임을 쉽게 알 수 있다.  $\sim_n$ 의 동치류들이 주어졌을 때  $\sim_{n+1}$ 의 동치류들은 다음 식에 의해 구할 수 있다.

$p \sim_n q$ 이고 모든  $a \in \Sigma$ 에 대하여  $\delta(p, a) \sim_n \delta(q, a)$ 일 경우에만  $p \sim_{n+1} q$ 이다.

$\sim$ 의 동치류들이 구해지면,  $M = (Q, \Sigma, \delta, q_0, F)$ 으로부터 새로운 DFA  $M = (Q', \Sigma, \delta', q'_0, F')$ 을 만들 수 있다.

- $Q'$ 은  $\sim$ 의 동치류들의 집합이다.
- $M$ 에서  $\delta(p, a) = q$ 이면  $M'$ 에서  $\delta'([p], a) = [q]$ 이다.
- 초기상태  $q'$ 은  $[q_0]$ 이다.
- $q \in F$ 이면  $[q] \in F'$ 이다.

**예제 2.33** 그림 2.16의 DFA에 대하여  $\sim$ 의 동치류를 단계적으로 구하면 다음과 같다.

$$\begin{aligned}\sim_0 &: \{q_1, q_2, q_4, q_5\}, \{q_3, q_6\} \\ \sim_1 &: \{q_1, q_4\}, \{q_2, q_5\}, \{q_3\}, \{q_6\} \\ \sim_2 &: \{q_1, q_4\}, \{q_2, q_5\}, \{q_3\}, \{q_6\}\end{aligned}$$

$\sim$ 의 동치류는 네 개이고, 따라서 네 개의 상태를 가진 DFA를 만들 수 있다. 이렇게 만들어진 DFA가 그림 2.7의 DFA이다.

**정리 2.6 (Myhill-Nerode)** 정규언어  $L$ 을 받아들이는 최소 개의 상태를 가진 DFA는 유일하다.

Myhill-Nerode의 정리를 이용하여 위에서 만들어진  $M'$ 이 최소 개의 상태를 가진 DFA임을 증명할 수 있다 [HMU]. 그러므로 정규 언어  $L$ 을 받아들이는 최소 상태 DFA를 구하기 위해서는  $L$ 을 받아들이는 임의의 DFA  $M$ 을 만든 후  $M$ 으로부터  $\sim$ 을 이용하여 최소 개의 상태를 가진 DFA  $M'$ 을 만들면 된다.

## 2.9 연습 문제

### 1. 정규식

1.1. 알파벳  $\{0, 1\}$  상에서 다음 언어를 표시하는 정규식을 구하라.

- (a) 짝수 개의 0을 가진 스트링의 집합
- (b) 11이 한 번 이하 나타나는 스트링의 집합
- (c) 111을 부분스트링으로 갖지 않는 스트링의 집합
- (d) 1의 개수가 3의 배수인 스트링의 집합
- (e) 01로 끝나지 않는 스트링의 집합
- (f) 111이 정확히 한 번 나타나는 스트링의 집합

1.2.  $(00)^*(11)^*1$ 의 여집합을 표시하는 정규식을 구하라.

1.3. 알파벳  $\{a, b\}$  상에서 다음 언어를 표시하는 정규식을 구하라.

- (a)  $ab$ 가 나타나지 않는 스트링의 집합
- (b)  $\{a^n b^m : (n + m)\text{은 짝수}\}$
- (c) 짝수 개의  $a$ 와  $b$ 를 가진 스트링의 집합
- (d)  $\{a^n b^m : n\text{과 }m\text{은 짝수}\}$

1.4. 알파벳  $\{a, b\}$  상에서 다음 언어를 표시하는 정규식을 구하라.

- (a)  $L = \{w \in \{a, b\}^* : |w| \bmod 3 = 0\}$

- (b)  $L = \{w \in \{a, b\}^* : N_a(w) \bmod 3 = 0\}$ ,  $N_a(w)$ 는  
스트링  $w$ 에서  $a$ 의 개수를 나타낸다
- 1.5. 알파벳  $\{0, 1\}$  상에서 1이 정확히 한번 나타나는 스트링  
의 집합에 대하여 정규식을 구하라.
  - 1.6. 알파벳  $\{a, b, c\}$  상에서  $a$ 의 연속은 모두 그 길이가 3의  
배수인 스트링의 집합을 표시하는 정규식을 구하라.
2. 유한 오토마타
- 2.1. 0과 1의 개수가 둘 다 짝수인 스트링을 받아들이는 DFA  
를 구하라.
  - 2.2.  $\Sigma = \{0, 1\}$ 일 때, 맨 왼쪽 글자와 맨 오른쪽 글자가 다른  
스트링을 받아들이는 DFA를 구하라.
  - 2.3.  $\Sigma = \{0, 1\}$ 일 때, 1의 연속은 모두 그 길이가 3또는 4인  
스트링을 받아들이는 DFA를 구하라.
  - 2.4. 모든 00 다음에 바로 1이 나오는 스트링을 받아들이는  
DFA를 구하라. (즉, 011, 0010, 110011001은 받아들이고,  
0001, 00100은 받아들이지 않아야 한다)
  - 2.5.  $((0 + 1)1^*)0$ 의 여집합을 받아들이는 DFA를 구하라.
  - 2.6. 100을 부분스트링으로 갖지 않고 010으로 끝나는 스트링  
을 받아들이는 DFA를 구하라.
  - 2.7. 스트링을 정수의 이진표현으로 간주했을 때 5의 배수인  
스트링을 받아들이는 DFA를 구하라. 예를 들면, 00101  
과 1111은 각각 정수 5와 15를 나타내므로 받아들여져야  
된다.
  - 2.8.  $aab$ 를 부분스트링으로 갖는 스트링을 받아들이는 DFA를  
구하라.
  - 2.9.  $\{w \in \{a, b\}^* : N_a(w) \bmod 3 > N_b(w) \bmod 3\}$ 를 표시하  
는 DFA를 구하라.

2.10.  $\Sigma = \{0, 1\}$  일 때, 길이가 2인 0의 연속과 1의 연속이 없는 스트링을 받아들이는 DFA를 구하라.

2.11. 다음 언어를 받아들이는 DFA를 구하라.

$$L = \{w \in \{a, b\}^*: N_a(w) \bmod 2 > N_b(w) \bmod 2\}$$

2.12. 010 또는 101을 부분스트링을 갖고 있는 스트링을 받아들이는 DFA를 구하라.

2.13. 011로 끝나는 스트링을 받아들이는 DFA를 구하라.

2.14. 유한 오토마타  $M$ 이 주어졌을 때,  $L(M)$ 이 유한 집합인지 결정하라.

### 3. 비결정 유한 오토마타

3.1. 다음이 올바른지 아닌지 증명하라.

(i) NFA  $M = (Q, \Sigma, \delta, q_0, F)$ 에 대하여  $L(M)$ 의 여집합은  $\{w \in \Sigma^* : \delta^*(q_0, w) \cap F = \emptyset\}$ 이다

(ii) NFA  $M = (Q, \Sigma, \delta, q_0, F)$ 에 대하여  $L(M)$ 의 여집합은  $\{w \in \Sigma^* : \delta^*(q_0, w) \cap (Q - F) \neq \emptyset\}$ 이다

3.2. NFA  $M = (Q, \Sigma, \Delta, q_0, F)$ 에 대한  $L(M)$ 의 정의를 설명하고 정의를 이용하여 다음 NFA가 011과 100 각각을 받아들이는지 아닌지 증명하라.

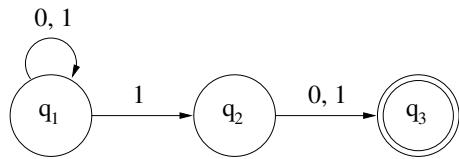


Figure 2.17:

### 4. 유한 오토마타의 동등성

4.1. 다음 NFA와 동등한 DFA를 구하라.

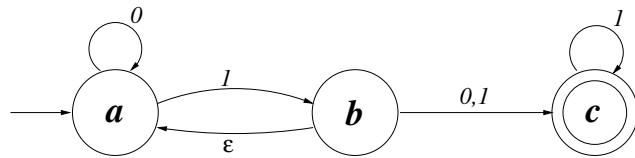


Figure 2.18:

4.2. 다음 NFA와 동등한 DFA를 구하라.

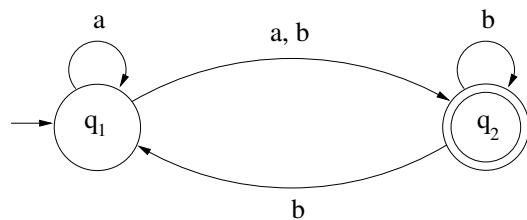


Figure 2.19:

4.3.  $(abc + bc)^*$ 의 NFA를 구하고 이 NFA를 DFA로 바꿔라.

4.4.  $\Sigma = \{0, 1\}$ 일 때, 끝에서 세 번째 글자가 1인 스트링을 받아들이는 NFA와 DFA를 구하고, 상태의 개수를 비교 하라.

4.5. 다음 NFA와 동등한 DFA를 구하라.

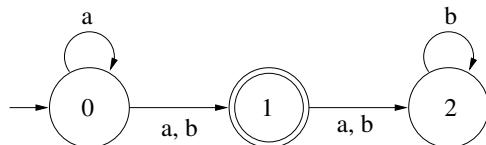


Figure 2.20:

4.6. 다음 NFA와 동등한 DFA를 구하라.

4.7. 다음 NFA와 동등한 DFA를 구하라.

5. 정규식과 유한 오토마타

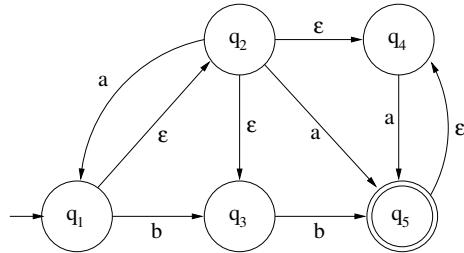


Figure 2.21:

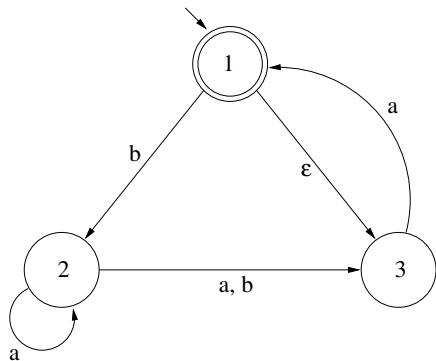


Figure 2.22:

5.1. 다음 정규식과 동등한 NFA를 구하라.

- (a)  $(0 + 1)^* 1 (0 + 11)^*$
- (b)  $(01)^* + 10^*$
- (c)  $((0 + 1)(0 + 1))^* + ((0 + 1)(0 + 1)(0 + 1))^*$
- (d)  $(01 + 001)^* + 10$
- (e)  $((0 + 1)(0 + 1))^* + (0 + 1)^* 11$

5.2. 다음 DFA와 동등한 정규식을 구하라.

5.3. 정규식  $(0 + 1)^* 01$ 과 동등한 DFA를 구하라.

6. 정규언어의 성질

6.1.  $L_1 \cup L_2$ 가 정규언어이고  $L_1$ 이 유한집합이라고 하자.  $L_2$ 가 정규언어인지 아닌지 증명하라.

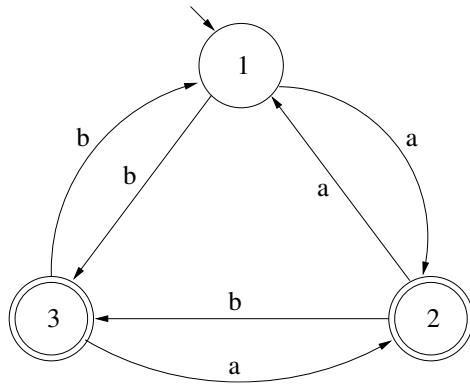


Figure 2.23:

- 6.2.  $L_1 \cup L_2$ 와  $L_1$ 이 정규언어라고 하자.  $L_2$ 가 정규언어인지 아닌지 증명하라.
- 6.3. 다음 문장이 참인지 거짓인지 증명하라.  $L_1$ 과  $L_2$ 가 정규언어가 아니면,  $L_1 \cup L_2$ 도 정규언어가 아니다.
- 6.4.  $L$ 이 정규언어라고 하자.  $\{uv : u \in L, v \in L^R\}$ 이 정규언어인지 아닌지 증명하라.
- 6.5. 언어  $L$ 의 헤드(head)는  $L$ 에 속하는 스트링의 모든 어두의 집합이다. 즉

$$\text{head}(L) = \{x : \text{어떤 } y \in \Sigma^* \text{에 대하여 } xy \in L\}$$

$L$ 이 정규언어이면  $\text{head}(L)$ 도 정규언어임을 증명하라.

- 6.6. 다음 언어가 정규언어인지 아닌지 증명하라.

$$\{a^n : n \text{은 소수}\}$$

- 6.7. 다음 언어가 정규언어인지 아닌지 증명하라.
- (a)  $\{a^n : n \text{은 } 3 \text{의 배수가 아니다}\}$
  - (b)  $\{0^n 1^m : n/m \text{은 정수}\}$

- (c)  $\{0^n 1^m : n \leq m\}$
- (d)  $\{0^n 1^m : 1 \leq n \leq m \leq 2n\}$
- (e)  $\{ww^R : w \in \{0, 1\}^*\}$
- (f)  $\{0^n 1^m 0^k : n + m + k \geq 3\}$
- (g)  $\{a^n b^m c^k : n, m, k \geq 1, n + m + k \geq 5\}$
- (h)  $\{a^n b^m c^k : n, m \geq 0, k \geq n + m\}$
- (i)  $L = \{w \in \{0, 1\}^* : w\text{에서 } 01\text{과 } 10\text{의 발생 횟수가 같다}\}$   
 (즉 101에는 01과 10이 한 번씩 나타나므로  $101 \in L$   
 이고, 1010에는 01은 한 번, 10은 두 번 나타나므로  
 $1010 \notin L$ 이다)

6.8. 다음 언어가 정규언어인지 아닌지 증명하라.

- (a)  $\{a^{n^2} \mid n \text{ 은 정수}, n \geq 1\}$
- (b)  $\{w \in I^* \mid w\text{는 } 5\text{의 배수인 숫자를 단항 표기법(unary notation)으로 나타낸 것이다}\}$

6.9.  $L$ 이 정규언어이면  $L^R = \{w \mid w^R \in L\}$ 도 정규언어임을 증명하라.

6.10. 다음 언어가 정규언어인지 아닌지 증명하라.

- (i)  $a$ 와  $b$ 의 개수가 같은 스트링 중, 어두(prefix)에  $b$ 의 개수보다 2개 이상 많이  $a$ 를 포함하지 않거나 혹은  $a$ 의 개수보다 2개 이상 많이  $b$ 를 포함하지 않는 스트링의 집합
- (ii)  $n$ 이 5자리 이하 소수인  $a^n$  스트링의 집합

6.11.  $\Sigma$ 와  $\Delta$ 가 알파벳이라고 하자. 다음과 같은 특징을 만족할 때  $h : \Sigma^* \rightarrow \Delta^*$  함수는 준동형(homomorphism)이라고 한다:

$$\begin{aligned} h(e) &= e \\ h(wa) &= h(w)h(a) \quad w \in \Sigma^*, a \in \Sigma \end{aligned}$$

$L \subseteq \Sigma^*$ 가 정규언어이면  $h(L) = \{h(w) : w \in L\}$ 도 정규언어인지 증명하라.

6.12.  $S_1$ 과  $S_2$ 의 대칭차집합(symmetric difference)은 다음과 같이 정의 된다.

$$\{x : (x \in S_1 \text{ and } x \notin S_2) \text{ or } (x \notin S_1 \text{ and } x \in S_2)\}$$

정규언어 종류는 대칭차집합에 대하여 닫혀 있는지 증명하라.

6.13. 다음 언어가 정규언어인지 아닌지 증명하라.

- (i)  $\{a^n : n = k^2 \text{ for some } k \geq 0\}$
- (ii)  $\{a^n b^m : n \geq 100, m \leq 100\}$
- (iii)  $\{uvw^Rv : u, v, w \in \{a, b\}^+\}$

6.14.  $L$ 이 정규언어이면  $L - \{\epsilon\}$ 도 정규언어임을 증명하라.

6.15. 다음 언어가 정규언어인지 아닌지 증명하라.

$$L = \{0^i 1^j 0^k : i + j + k \geq 4\}$$

6.16.  $L_1$ 과  $L_2$ 가 정규언어이면  $L_1 - L_2$ 가 정규언어인지 증명하라.

6.17. 다음 언어가 정규언어인지 아닌지 증명하라.

- (a)  $L_1 = \{a^n : n \geq 4\}$
- (b)  $L_2 = \{w \in \{0, 1\}^* : N_0(w) = N_1(w)\}$ ,  $N_a(w)$ 는  $w$ 에서  $a$ 의 개수를 나타낸다

6.18.  $\{0^n 1^m : n \leq m \leq 3n\}$ 이 정규언어인지 아닌지 증명하라.

## 7. 유한 오토마타의 응용

7.1.  $L$ 이  $\{0, 1\}$  상의 정규언어라고 하자.  $L$ 이 짹수 길이의 스트링을 포함하는지 결정하는 방법을 기술하라.

## 8. 유한 오토마타의 최소화

8.1. 아래의 DFA와 동등한 최소 상태 DFA를 구하라.

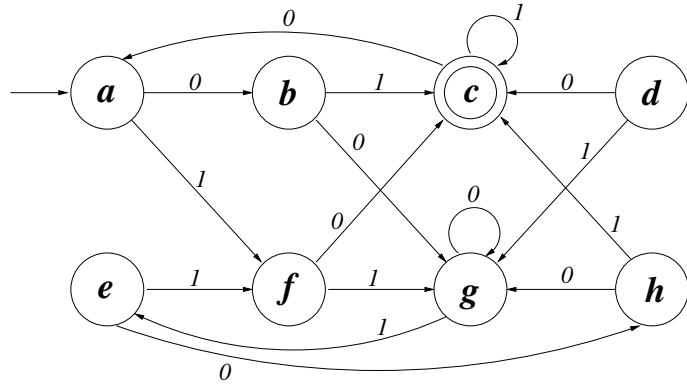


Figure 2.24:



# Chapter 3

## 문맥무관 언어

### 3.1 문맥무관 문법

**정의 3.1** 문법(grammar)  $G$ 는 네 가지 요소  $(V, \Sigma, S, P)$ 로 구성된다.

1.  $V$ 는 변수들의 유한 집합이고,
2.  $\Sigma$ 는 알파벳이고,
3.  $S \in V$ 는 시작변수이고,
4.  $P$ 는 생성규칙들의 유한 집합이다. 각 생성규칙은  $x \rightarrow y$ 의 형태를 가지는데, 여기에서  $x \in (V \cup \Sigma)^*V(V \cup \Sigma)^*$  이고  $y \in (V \cup \Sigma)^*$  이다.

예를 들면  $A, B \in V$ 와  $a, b \in \Sigma$ 에 대하여

$$aAb \rightarrow bBa$$

는 생성규칙이다.

**정의 3.2** 문법  $G = (V, \Sigma, S, P)$ 의 모든 생성규칙이  $A \in V$ 와  $x \in (V \cup \Sigma)^*$ 에 대하여

$$A \rightarrow x$$

형태이면  $G$ 는 문맥무관 문법(context-free grammar)이라고 부른다.

$A \rightarrow x$ 와  $A \rightarrow y$ 를 줄여서

$$A \rightarrow x \mid y$$

로 표시한다.

문법은 문장을 표현하는 데 광범위하게 사용된다. 변수는 일반적으로 문장 성분을 나타낸다. 예를 들어, “구름이 산을 넘는다”는 문장을 다음 문법을 이용하여 만들 수 있다.

$$\begin{aligned} \langle \text{문장} \rangle &\rightarrow \langle \text{주어} \rangle \langle \text{목적어} \rangle \langle \text{서술어} \rangle \\ \langle \text{주어} \rangle &\rightarrow \langle \text{명사} \rangle \langle \text{주격조사} \rangle \\ \langle \text{목적어} \rangle &\rightarrow \langle \text{명사} \rangle \langle \text{목적격조사} \rangle \\ \langle \text{서술어} \rangle &\rightarrow \langle \text{동사} \rangle \\ \langle \text{명사} \rangle &\rightarrow \text{구름} \mid \text{산} \\ \langle \text{주격조사} \rangle &\rightarrow \text{이} \\ \langle \text{목적격조사} \rangle &\rightarrow \text{을} \\ \langle \text{동사} \rangle &\rightarrow \text{넘는다} \end{aligned}$$

이제 문장을 만들기 위해서는 시작변수  $S$ 에서부터 생성규칙을 차례로 적용한다. 일반적으로  $u, v \in (V \cup \Sigma)^*$ 와  $A \in V$ 에 대하여  $uAv$ 에 생성규칙  $A \rightarrow w$ 를 적용하면  $uwv$ 를 얻는데, 이 과정을 유도(derivation)라고 부르고

$$uAv \Rightarrow uwv$$

로 표시한다. 따라서

$$\begin{aligned} \langle \text{문장} \rangle &\Rightarrow \langle \text{주어} \rangle \langle \text{목적어} \rangle \langle \text{서술어} \rangle \\ &\Rightarrow \langle \text{명사} \rangle \langle \text{주격조사} \rangle \langle \text{목적어} \rangle \langle \text{서술어} \rangle \\ &\Rightarrow \text{구름} \langle \text{주격조사} \rangle \langle \text{목적어} \rangle \langle \text{서술어} \rangle \end{aligned}$$

- $\Rightarrow$  구름이<목적어><서술어>
- $\Rightarrow$  구름이<명사><목적격조사><서술어>
- $\Rightarrow$  구름이 산<목적격조사><서술어>
- $\Rightarrow$  구름이 산을<서술어>
- $\Rightarrow$  구름이 산을<동사>
- $\Rightarrow$  구름이 산을 넘는다

$x$ 에 생성규칙을 0번 이상 적용하여  $y$ 를 얻으면, 이를  $x \xrightarrow{*} y$ 로 표시 한다.

- $S \xrightarrow{*} x$  ( $x \in (V \cup \Sigma)^*$ )이면  $x$ 를 문법  $G$ 의 문장형태라고 부른다.
- $S \xrightarrow{*} w$  ( $w \in \Sigma^*$ )이면  $w$ 를 문법  $G$ 의 문장(또는  $G$ 가 생성하는 스트링)이라고 부른다.

위의 예에서 “구름이<목적어><서술어>”는 문장형태이고 “구름이 산을 넘는다”는 문장이다. 그러나 이 문법이 “산이 구름을 넘는다”도 만들 수 있음을 유의하라.

문법  $G$ 의 언어  $L(G)$ 는  $G$ 가 생성하는 문장의 집합이다. 즉

$$L(G) = \{w \in \Sigma^* : S \xrightarrow{*} w\}.$$

**예제 3.1**  $G = (\{S\}, \{0, 1\}, S, P)$  이고  $P$ 가

$$S \rightarrow 0S1 \mid \epsilon$$

이면  $G$ 는 문맥무관 문법이다.

$$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 0011$$

에서 보는 것처럼  $L(G) = \{0^n 1^n : n \geq 0\}$ 임을 알 수 있다.

**정의 3.3** 문맥무관 문법이 생성하는 언어를 문맥무관 언어(*context-free language*)라고 부른다.

**예제 3.2**  $G = (\{S\}, \{((), )\}, S, P)$  이고  $P$ 가

$$S \rightarrow SS \mid (S) \mid \epsilon$$

이면,

$$\begin{aligned} S &\Rightarrow SS \xrightarrow{*} (S)(S) \xrightarrow{*} ()() \\ S &\Rightarrow SS \xrightarrow{*} ((S))(S) \xrightarrow{*} ((())()) \end{aligned}$$

이다. 즉  $G$ 는 여는 괄호와 닫는 괄호가 올바로 쌍을 이루는 스트링을 생성한다.

**예제 3.3** 다음 문맥무관 문법은 수식을 생성한다.

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

즉

$$\begin{aligned} E &\Rightarrow E + T \\ &\Rightarrow E + T * F \\ &\Rightarrow E + T * (E) \\ &\Rightarrow E + T * (E + T) \\ &\xrightarrow{*} F + F * (F + F) \\ &\xrightarrow{*} \text{id} + \text{id} * (\text{id} + \text{id}) \end{aligned}$$

**예제 3.4**  $L = \{0^m 1^n : m \neq n\}$ 을 생성하는 문맥무관 문법을 구하라.

$m > n$ 과  $m < n$ 의 경우로 나누어서 생각한다. 문맥무관 문법을 만들기 위해서는 어떤 변수가 무슨 스트링을 만들 것인지를 정해야 한다.  $m > n$ 인 경우, 변수  $A$ 가 0과 1의 개수가 같은 스트링을

만들도록 하고, 변수  $B$ 가 1개 이상의 0을 만들면 된다.

$$\begin{aligned} S &\rightarrow BA \\ A &\rightarrow 0A1 \mid \epsilon \\ B &\rightarrow 0B \mid 0 \end{aligned}$$

$m < n$ 인 경우, 변수  $A$ 는 그대로 이용하면 되고 변수  $C$ 가 1개 이상의 1을 만들면 된다.

$$\begin{aligned} S &\rightarrow AC \\ C &\rightarrow 1C \mid 1 \end{aligned}$$

**예제 3.5**  $L = \{a^n b^m c^k : n + m = k\}$ 를 생성하는 문맥무관 문법을 구하라.

**예제 3.6**  $L = \{0^n 1^m : n \leq m \leq 2n\}$ 을 생성하는 문맥무관 문법을 구하라.

**예제 3.7**  $L = \{a^m b^n c^u d^v : m + n = u + v\}$ 를 생성하는 문맥무관 문법을 구하라.

**예제 3.8**  $L_{ab} = \{w \in \{a, b\}^* : N_a(w) = N_b(w)\}$ 를 생성하는 문맥무관 문법을 구하라.

간단한 답은

$$S \rightarrow aSb \mid bSa \mid SS \mid \epsilon$$

이다. 또 다른 답은

$$\begin{aligned} S &\rightarrow aB \mid bA \mid \epsilon \\ A &\rightarrow a \mid aS \mid bAA \\ B &\rightarrow b \mid bS \mid aBB \end{aligned}$$

이다. 여기에서  $A$ 는  $a$ 가  $b$ 보다 하나 많은 스트링을 만들어내고  $B$ 는  $b$ 가  $a$ 보다 하나 많은 스트링을 만들어낸다.

문맥무관 문법이 어떤 언어를 만드는지 쉽게 확인할 수 없을 때 증명을 위해 이를 밝힐 수 있다.

**예제 3.9** 예제 3.8의 간단한 문법을  $G$ 라고 할 때,  $L(G) = L_{ab}$ 임을 증명하라.

**증명.**  $L(G) \subseteq L_{ab}$ :  $G$ 에서 알파벳 글자를 만드는 생성규칙은  $S \rightarrow aSb$ 와  $S \rightarrow bSa$ 이므로,  $G$ 가 생성하는 모든 스트링은 같은 수의  $a$ 와  $b$ 를 갖는다.

$L_{ab} \subseteq L(G)$ :  $w \in L_{ab}$ 의 길이에 대한 귀납법으로 증명하자.  $|w| = 0$ 일 때,  $G$ 가  $\epsilon$ 을 생성한다.  $|w| < k$ 일 때,  $G$ 가  $w$ 를 생성한다고 (즉  $S \xrightarrow{*} w$ ) 가정하자.

길이가  $k$ 인  $w \in L_{ab}$ 의 양쪽 끝 글자를 보자. 다음 네 가지 경우가 있다:  $w = axb$ ,  $w = bxa$ ,  $w = axa$ ,  $w = bxb$ .

- $w = axb$ 인 경우:  $x$ 는 같은 수의  $a$ 와  $b$ 를 가지고 있고 그 길이가  $k$  미만이므로, 귀납 가설에 의해  $S \xrightarrow{*} x$ 이다. 따라서  $S \Rightarrow aSb \xrightarrow{*} axb$ 이므로,  $G$ 가  $w$ 를 생성한다. ( $w = bxa$ 인 경우도 마찬가지)
- $w = axa$ 인 경우: 그림 3.1과 같이  $w$ 의 왼쪽 끝에 0을 놓고 오른쪽으로 진행하면서  $a$ 가 나오면 +1을 하고  $b$ 가 나오면 -1을 하자.  $w$ 가 같은 수의  $a$ 와  $b$ 를 가지고 있으므로 오른쪽 끝의 값은 0이 되고, 그 바로 앞의 값은 -1이 된다.  $x$ 의 왼쪽 끝은 +1이고, 오른쪽 끝은 -1이므로  $x$ 의 중간에 0이 반드시 나타나야 한다.  $w$ 에서 이 0의 왼쪽 부분을  $u$ , 오른쪽 부분을  $v$ 라고 하면 (즉  $w = uv$ ),  $u$ 와  $v$ 는 각각 같은 수의  $a$ 와  $b$ 를 갖고, 그 길이는  $k$  미만이다. 귀납 가설에 의해  $S \xrightarrow{*} u$ 와  $S \xrightarrow{*} v$ 가 성립한다. 따라서  $S \Rightarrow SS \xrightarrow{*} uv$ 이므로,  $G$ 가  $w$ 를 생성한다. ( $w = bxb$ 인 경우도 마찬가지)

□

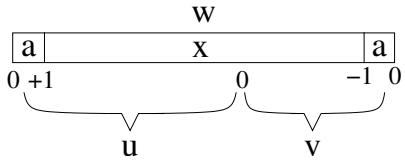


Figure 3.1:

## 3.2 정규문법

**정의 3.4** 문법  $G = (V, \Sigma, S, P)$ 의 모든 생성규칙이  $A, B \in V$ 와  $w \in \Sigma^*$ 에 대하여

$$\begin{aligned} A &\rightarrow wB \\ A &\rightarrow w \end{aligned}$$

형태이면  $G$ 는 정규문법이라고 부른다.

정규문법의 유도과정에서 문장형태는 항상 오른쪽 끝에 한 개의 변수만을 갖게 된다.

$$\begin{aligned} S &\Rightarrow w_1 A \\ &\Rightarrow w_1 w_2 B \\ &\Rightarrow w_1 w_2 w_3 C \\ &\Rightarrow w_1 w_2 w_3 w_4 \end{aligned}$$

정규문법이 생성하는 언어의 집합이 정규언어 종류임을 보이고자 한다.

**정리 3.1**  $L$ 이 정규언어이면,  $L = L(G)$ 인 정규문법  $G$ 가 존재한다.

**증명.**  $L$ 이 정규언어이므로  $L$ 을 받아들이는 DFA  $M = (Q, \Sigma, \delta, q_0, F)$ 이 존재한다.  $M$ 의 전이과정을 흡내내는 정규문법  $G = (Q, \Sigma, q_0, P)$ 를 만들고자 한다. (DFA의 상태가 문법의 변수가 되는 것에 유의) 즉

그림 3.2처럼 DFA가  $w$ 를 읽고 상태  $q$ 에 도달하면, 문법은  $q_0 \xrightarrow{*} wq$ 에 이르도록 문법  $G$ 를 만든다.

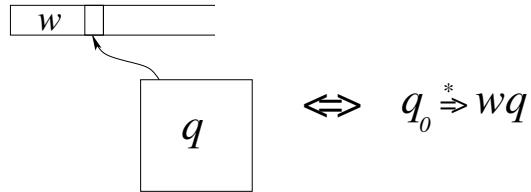


Figure 3.2:

$P$ 는 다음과 같이 정의된다.

1. 각 전이  $\delta(p, a) = q$ 에 대하여  $p \rightarrow aq$ 를  $P$ 에 더한다.
2. 각 최종 상태  $q \in F$ 에 대하여  $q \rightarrow \epsilon$ 를  $P$ 에 더한다.

$\delta^*(p, w) = q$ 인 경우에만  $p \xrightarrow{*} wq$ 임을 보일 수 있다. 따라서  $\delta^*(q_0, w) \in F$ 인 경우에만  $q_0 \xrightarrow{*} w$ 이다. 즉  $L(M) = L(G)$ .  $\square$

**예제 3.10** 예제 2.18(그림 2.8)의 DFA와 동등한 정규문법을 구하라. (죽은 상태는 생략)

$$\begin{aligned} q_1 &\rightarrow 0q_2 \mid \epsilon \\ q_2 &\rightarrow 1q_3 \\ q_3 &\rightarrow 0q_4 \mid \epsilon \\ q_4 &\rightarrow 0q_2 \mid 1q_3 \mid \epsilon \end{aligned}$$

**정리 3.2** 정규문법  $G$ 가 생성하는 언어  $L(G)$ 는 정규언어이다.

**증명.**  $G = (V, \Sigma, S, P)$ 라고 하자.  $G$ 의 유도과정을 흉내내는 NFA  $M = (Q, \Sigma, \Delta, S, \{q_f\})$ 를 만들고자 한다. 정규문법에 의해 만들어지는 문장형태는 항상  $wA$  ( $w \in \Sigma^*, A \in V$ )이므로, 이 때 NFA는  $w$ 를 읽고 상태  $A$ 에 도달해 있도록 만든다.

상태 집합  $Q$ 는  $V \cup \{q_f\}$ 와 중간 상태들로 구성된다.  $\Delta$ 는 다음과 같이 정의된다.

1. 각 생성규칙  $A \rightarrow wB$ 에 대하여,  $\Delta^*(A, w)$ 에  $B$ 를 더해 준다.
2. 각 생성규칙  $A \rightarrow w$ 에 대하여,  $\Delta^*(A, w)$ 에  $q_f$ 를 더해 준다.

이 때,  $w$ 의 길이가 2 이상이면  $\Delta^*(A, w)$ 를 표시하기 위하여 중간 상태를 필요로 한다.

$S \xrightarrow{*} w$ 인 경우에만  $q_f \in \Delta^*(S, w)$ 임을 귀납법으로 증명할 수 있다. 즉  $L(G) = L(M)$ .  $\square$

### 예제 3.11 정규문법

$$\begin{array}{lcl} S & \rightarrow & aA \\ & & A \rightarrow abS \mid a \end{array}$$

와 동등한 NFA를 구하라. 그림 3.3를 보라.

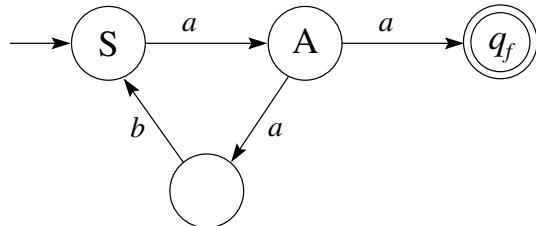


Figure 3.3:

정규문법은 문맥무관 문법의 일종이므로, 모든 정규언어는 문맥 무관언어이다. 즉, 정규언어 종류는 문맥무관언어 종류의 부분집합이다.

언어는 스트링의 집합이고, 언어종류는 언어의 집합임에 유의하라. 그림 3.4에서 언어  $\{0^n1^n\}$ 은 언어  $L(0^*1^*)$ 의 부분집합이지만,  $\{0^n1^n\}$ 은 정규언어가 아니고 문맥무관언어이다.

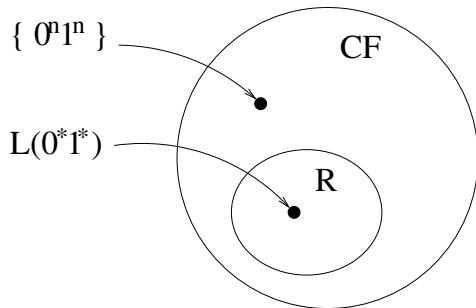


Figure 3.4:

### 3.3 파스 트리

$G = (V, \Sigma, S, P)$ 를 문맥무관 문법이라고 하자.  $G$ 에서의 유도과정을 파스 트리로 표시할 수 있다.  $G$ 에 대한 파스 트리(parse tree)는 다음과 같다.

1. 루트 노드의 이름은  $S$ 이다.
2. 내부 노드의 이름은  $V$ 의 원소이고,  $A$  이름의 내부 노드가  $u_1, \dots, u_r$  이름의 자식 노드들을 가지고 있으면  $A \rightarrow u_1 \dots u_r$ 은  $P$ 의 생성규칙이어야 한다.
3. 단말 노드의 이름은  $\Sigma \cup \{\epsilon\}$ 의 원소이고, 단말 노드의 이름이  $\epsilon$ 이면 이 노드는 부모 노드의 유일한 자식 노드이어야 한다.

단말 노드의 이름을 왼쪽부터 오른쪽으로 접합시킨 것을 파스 트리의 열매 스트링이라고 부른다.  $S \xrightarrow{*} w$ 를 파스 트리로 나타내면 그 열매 스트링이  $w$ 가 된다.

#### 예제 3.12 예제 3.2에서 유도과정

$$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ()S \Rightarrow ()(S) \Rightarrow ()()$$

는 다음 파스 트리로 나타내어진다.

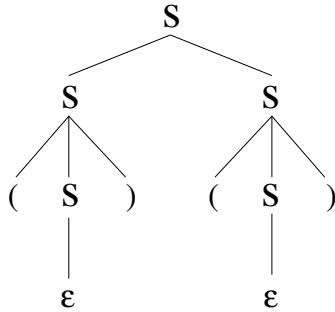


Figure 3.5:

유도과정에서 항상 맨 왼쪽의 변수에 생성규칙을 적용하는 것을 좌측 유도(leftmost derivation)라고 부른다. 한 개의 파스 트리에 대하여 여러 가지 유도 과정이 있을 수 있지만 좌측 유도는 한 가지만 존재한다. 즉 파스 트리와 좌측 유도는 1대1 대응관계를 갖는다.

### 예제 3.13 앞 예제에서 유도과정

$$S \Rightarrow SS \Rightarrow S(S) \Rightarrow S() \Rightarrow (S)() \Rightarrow ()()$$

도 파스 트리로 나타내면 위의 그림과 같아진다.

**정의 3.5** 문맥무관 문법  $G$ 가 어떤  $w \in L(G)$ 에 대하여 두 개 이상의 파스 트리를 가지면,  $G$ 는 애매하다(ambiguous)라고 말한다.

### 예제 3.14 다음 문법 $G$ 를 고려하자.

$$\begin{aligned} E &\rightarrow E + E \\ E &\rightarrow E * E \\ E &\rightarrow a \mid b \mid c \end{aligned}$$

$a + b * c$ 는 그림 3.6과 같이 두 개의 파스 트리를 가지므로  $G$ 는 애매하다. 그러나 이 문법은 예제 3.3처럼 애매하지 않은 문법으로 바꿀 수 있다.

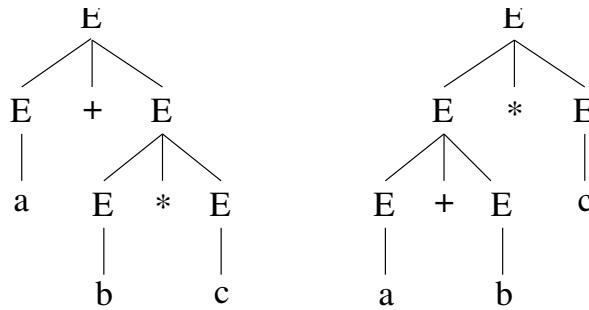


Figure 3.6:

## 예제 3.15

$$\begin{aligned} S &\rightarrow \text{if } C \text{ then } S \text{ else } S \\ S &\rightarrow \text{if } C \text{ then } S \\ S &\rightarrow x \mid y \\ C &\rightarrow a \mid b \end{aligned}$$

`if a then if b then x else y`는 두 개의 파스 트리를 가진다. 이 경우 일반적으로 문법은 그대로 둔 채 “`else`는 가장 가까운 `if`에 붙는다”는 의미(semantics)를 부여하여 애매성을 없앤다.

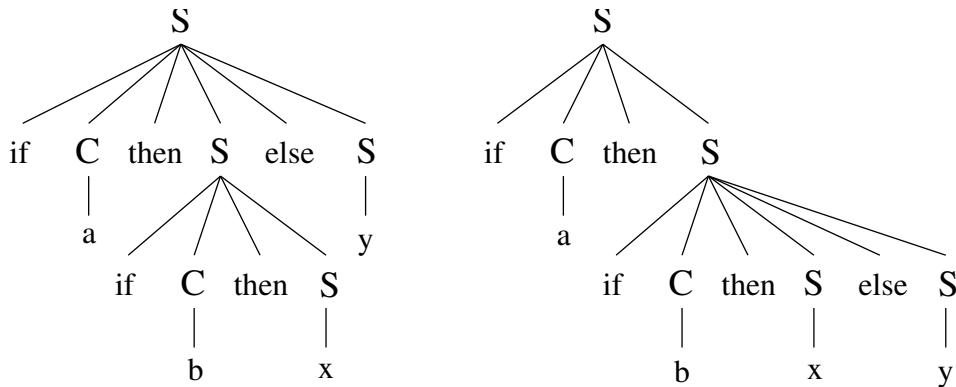


Figure 3.7:

**정의 3.6** 문맥무관 언어  $L$ 을 생성하는 애매하지 않은 문법  $G$ 가 존재하면,  $L$ 은 “애매하지 않다”라고 한다.  $L$ 을 생성하는 모든 문법이 애매하면,  $L$ 은 “본질적으로 애매하다”라고 한다.

**예제 3.16**  $L = \{a^n b^n c^m\} \cup \{a^m b^n c^n\}$ 은 본질적으로 애매하다.

(증명이 아니고 설명)  $L$ 을 생성하는 어떤 문법이든지  $\{a^n b^n c^m\}$ 과  $\{a^m b^n c^n\}$ 을 서로 다른 방법으로 만들어야 하므로  $a^n b^n c^n$ 을 만드는 파스 트리가 2개 이상 존재하게 된다.

## 3.4 표준형

문맥무관 언어  $L$ 이  $\epsilon$ 을 포함하면,  $L$ 을 생성하는 문법은 반드시  $A \rightarrow \epsilon$  형태의 생성규칙을 가져야 한다. 이 경우 시작변수  $S_0$ 에서

$$S_0 \rightarrow \epsilon \mid S$$

의 생성규칙을 만든 후,  $S$ 에서  $L - \{\epsilon\}$ 을 생성하면 된다. 따라서 이후의 문맥무관 언어는  $\epsilon$ 을 포함하지 않는다고 가정한다.

**정의 3.7 (Chomsky 표준형)** 문맥무관 문법  $G = (V, \Sigma, S, P)$ 의 모든 생성규칙이  $A, B, C \in V$ 와  $a \in \Sigma$ 에 대하여

$$A \rightarrow BC$$

$$A \rightarrow a$$

형태이면,  $G$ 를 Chomsky 표준형이라고 부른다.

**정의 3.8 (Greibach 표준형)** 문맥무관 문법  $G = (V, \Sigma, S, P)$ 의 모든 생성규칙이  $A \in V$ ,  $a \in \Sigma$ ,  $x \in V^*$ 에 대하여

$$A \rightarrow ax$$

형태이면,  $G$ 를 Greibach 표준형이라고 부른다.

예제 3.8의 두 번째 답은 Greibach 표준형이다.

**정리 3.3** 모든 문맥무관 언어는 Chomsky 표준형의 문법에 의해 생성된다.

**증명.** 임의의 문맥무관 문법  $G = (V, \Sigma, S, P)$ 를 다음 과정에 의해 Chomsky 표준형으로 바꾼다.

1.  $P$ 에서  $A \rightarrow \epsilon$  형태의  $\epsilon$  생성규칙을 없앤다.

1.1.  $A \xrightarrow{*} \epsilon$ 을 만족하는 모든 변수의 집합  $V_\epsilon$ 을 구한다.

(a)  $A \rightarrow \epsilon$ 인  $A$ 를  $V_\epsilon$ 에 넣는다.

(b)  $A \rightarrow B_1 \cdots B_r$ 에서 모든  $B_i$  ( $1 \leq i \leq r$ )이  $V_\epsilon$ 의 원소이면  $A$ 를  $V_\epsilon$ 에 넣는다.

1.2. 새로운 생성규칙의 집합  $P_1$ 을 만든다.  $P$ 의 각 생성규칙  $A \rightarrow x_1 \cdots x_t$  ( $x_i \in V \cup \Sigma$ )에 대하여  $x_i$ 가  $V_\epsilon$ 의 원소이면 그 자리에  $x_i$ 와  $\epsilon$ 을 번갈아 넣어 만들어지는 모든 생성규칙을  $P_1$ 에 넣는다. 단  $A \rightarrow \epsilon$ 이 만들어지는 경우 이것은  $P_1$ 에 넣지 않는다.

1.3.  $G_1 = (V, \Sigma, S, P_1)$ 는  $G$ 와 동등하다.

2.  $P_1$ 에서  $A \rightarrow B$  형태의 단위 생성규칙을 없앤다.

2.1.  $A \xrightarrow{*} B$ 를 만족하는 변수들을 구한다. 단위 생성규칙  $A \rightarrow B$ 에 대하여  $A$  정점에서  $B$  정점으로 가는 간선을 가지는 그래프를 만들면  $A \xrightarrow{*} B$  관계를 파악할 수 있다.

2.2.  $P_1$ 에서 단위 생성규칙이 아닌 것을  $P_2$ 에 넣는다.

2.3.  $A \xrightarrow{*} B$ 이고  $P_1$ 의 원소  $B \rightarrow x$ 이 단위 생성규칙이 아니면  $A \rightarrow x$ 를  $P_2$ 에 넣는다.

2.4.  $G_2 = (V, \Sigma, S, P_2)$ 은  $G_1$ 과 동등하다.

3.  $P_2$ 의 모든 생성규칙을 Chomsky 표준형으로 바꾼다.

- 3.1.  $A \rightarrow a$ 인 생성규칙은 이미 Chomsky 표준형이므로  $P_3$ 에 넣는다.
- 3.2.  $r \geq 2$ 인 각 생성규칙  $A \rightarrow x_1 \cdots x_r$  ( $x_i \in V \cup \Sigma$ )에서  $x_i$ 가 알파벳 글자  $a$ 이면 새로운 변수  $C_a$ 를 만들고  $C_a \rightarrow a$ 를  $P_3$ 에 넣고  $x_i$ 를  $C_a$ 로 대치한다. 그러면 생성규칙의 우변에 변수만이 나타난다.
- 3.3. 생성규칙  $A \rightarrow B_1 \cdots B_r$ 에 대하여 새로운 변수  $D_1, \dots, D_{r-2}$  을 도입하고

$$\begin{array}{lcl} A & \rightarrow & B_1 D_1 \\ D_1 & \rightarrow & B_2 D_2 \\ & \vdots & \\ D_{r-3} & \rightarrow & B_{r-2} D_{r-2} \\ D_{r-2} & \rightarrow & B_{r-1} B_r \end{array}$$

을  $P_3$ 에 넣는다.

- 3.4.  $V$ 와 새 변수들을 합하여  $V'$ 라 하면,  $G_3 = (V', \Sigma, S, P_3)$  은  $G_2$ 와 동등하다.

□

**예제 3.17** 다음 문맥무관 문법을 Chomsky 표준형으로 바꾸라.

$$\begin{array}{lcl} S & \rightarrow & aB \mid AAa \\ A & \rightarrow & B \mid S \\ B & \rightarrow & b \mid \epsilon \end{array}$$

$V_\epsilon = \{B, A\}$ 이므로 1 단계 후  $G_1$ 은 다음과 같다

$$S \rightarrow a \mid aB \mid AAa \mid Aa$$

$$\begin{array}{l} A \rightarrow B \mid S \\ B \rightarrow b \end{array}$$

$A \xrightarrow{*} B, A \xrightarrow{*} S$  이므로 2 단계 후  $G_2$ 는 다음과 같다

$$\begin{array}{l} S \rightarrow a \mid aB \mid AAa \mid Aa \\ A \rightarrow b \mid a \mid aB \mid AAa \mid Aa \\ B \rightarrow b \end{array}$$

3 단계 후 Chomsky 표준형은 다음과 같다.

$$\begin{array}{l} S \rightarrow a \mid CB \mid AD \mid AC \\ A \rightarrow b \mid a \mid CB \mid AD \mid AC \\ B \rightarrow b \\ C \rightarrow a \\ D \rightarrow AC \end{array}$$

**정리 3.4** 모든 문맥무관 언어는 Greibach 표준형의 문법에 의해 생성된다.

### 3.5 내리누름 오토마타

내리누름 오토마타(pushdown automata)를 줄여서 PA로 부르기도 하자. PA는 입력 테입과 제어장치외에 내리누름 저장장치인 스택을 가지고 있다. 내리누름 오토마타를 스택 오토마타라고 부르기도 한다. 그럼 3.8를 보라.

**정의 3.9** 내리누름 오토마타  $M$ 은 여섯 가지 요소  $(Q, \Sigma, \Gamma, \Delta, q_0, F)$ 로 구성된다. 여기에서

1.  $Q$ 는 상태들의 유한 집합이고

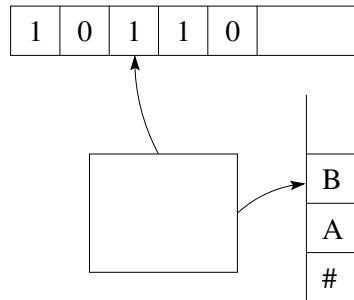


Figure 3.8:

2.  $\Sigma$ 는 입력 알파벳이고
3.  $\Gamma$ 는 스택 알파벳이고 시작 글자  $\#$ 를 포함한다.
4.  $\Delta$ 는  $(Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\})) \times (Q \times \Gamma^*)$ 의 부분집합인 전이 관계이고
5.  $q_0 \in Q$ 는 초기 상태이고
6.  $F \subseteq Q$ 는 최종상태들의 집합이다.

PA는  $\Delta$ 가 전이 관계이므로 비결정 오토마타이다. 따라서 유한 오토마타 때와 마찬가지로 최종상태는 한 개만 있다고 가정할 수 있다.  $((p, a, A), (q, u)) \in \Delta$ 는 PA의 상태가  $p$ 이고 입력 글자가  $a$ 이고 스택의 맨위 원소가  $A$ 일 때,  $a$ 를 읽고 스택의  $A$ 를  $u$ 로 바꾸고  $q$  상태가 되는 전이를 나타낸다. 따라서  $((p, 0, 0), (q, 10))$ 은 스택에 1을 넣는(push) 연산이고,  $((p, 0, 0), (q, \epsilon))$ 는 스택의 맨위 원소 0을 빼는(pop) 연산이다. 전이 관계에서  $\Sigma \cup \{\epsilon\}$ 의  $\epsilon$ 는 입력을 읽지 않고 전이가 일어날 수 있음을 나타내고,  $\Gamma \cup \{\epsilon\}$ 의  $\epsilon$ 는 스택을 보지 않고 전이가 일어날 수 있음을 나타낸다.

PA는 시작할 때, 스택에  $\#$ 만을 가지고 있다.  $\#$ 는 스택의 밑바닥을 나타내는 기능을 한다. PA의 전이 도중  $\#$ 를 스택에서 빼는 연산은 하지 않는다고 가정한다.

PA의 상황(configuration)은 현재 상태, 입력 스트링의 읽지 않은 부분과 현재 스택의 내용에 의해 결정된다. 입력 스트링이  $w$ 일 때, 시작 상황은  $(q_0, w, \#)$ 이다. 한 번의 전이에 의해서 PA  $M$ 의 상황이

변화하는 과정을  $\vdash_M$ 으로 표시한다. 현재 상황이  $(q, 01101, 100\#)$ 이고  $((q, 0, \epsilon), (p, 1)) \in \Delta$ 이면,

$$(q, 01101, 100\#) \vdash_M (p, 1101, 1100\#).$$

PA는 입력 스트링  $w$ 를 읽은 후의 상태가 최종상태이면 (스택의 내용에 관계없이)  $w$ 를 받아들인다. 즉,

$$L(M) = \{w \in \Sigma^* : (q_0, w, \#) \vdash_M^* (p, \epsilon, u), p \in F, u \in \Gamma^*\}.$$

**예제 3.18**  $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \Gamma = \{0, \#\}, \Delta, q_1, \{q_3\})$ 의  $\Delta$ 는 아래의 전이를 가지고 있다.

$$\begin{aligned} & ((q_1, 0, \epsilon), (q_1, 0)) \\ & ((q_1, \epsilon, \#), (q_3, \#)) \\ & ((q_1, 1, 0), (q_2, \epsilon)) \\ & ((q_2, 1, 0), (q_2, \epsilon)) \\ & ((q_2, \epsilon, \#), (q_3, \#)) \end{aligned}$$

$M$ 을 그림으로 나타내면 그림 3.9와 같다.  $L(M) = \{0^n 1^n : n \geq 0\}$ . 입력 스트링이 01이면  $M$ 의 전이과정은 다음과 같다.

$$\begin{aligned} & (q_1, 01, \#) \vdash (q_1, 1, 0\#) \vdash (q_2, \epsilon, \#) \vdash (q_3, \epsilon, \#) \\ & (q_1, 01, \#) \vdash (q_3, 01, \#) \text{ 전이는 더 이상 진행하지 못한다.} \end{aligned}$$

**예제 3.19**  $\{wcw^R : w \in \{a, b\}^*\}$ 을 받아들이는 PA를 구하라. 그림 3.10을 보라.

**예제 3.20**  $\{ww^R : w \in \{a, b\}^*\}$ 을 받아들이는 PA를 구하라. 그림 3.10에서  $((q_1, c, \epsilon), (q_2, \epsilon))$ 을  $((q_1, \epsilon, \epsilon), (q_2, \epsilon))$ 으로 바꾼다.

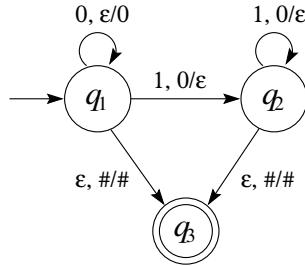


Figure 3.9:

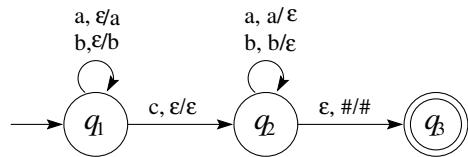


Figure 3.10:

**예제 3.21**  $\{a^n b^m c^k : n + m = k\}$ 를 받아들이는 PA를 구하라.

**예제 3.22**  $\{0^n 1^m : n \leq m \leq 2n\}$ 를 받아들이는 PA를 구하라.

**예제 3.23**  $\{vw : v, w \in \{0, 1\}^*, w \neq v^R, |v| = |w|\}$ 를 받아들이는 PA를 구하라.

## 3.6 내리누름 오토마타와 문맥무관 언어

**정리 3.5** 문맥무관 언어  $L$ 에 대하여,  $L = L(M)$ 인 내리누름 오토마타  $M$ 이 존재한다.

**증명.**  $L$ 을 생성하는 Greibach 표준형의 문맥무관 문법을  $G = (V, \Sigma, S, P)$ 라 하자.  $G$ 의 좌측 유도를 흉내내는 PA  $M$ 을 만들고자 한다.  $G$ 가 Greibach 표준형이므로 좌측 유도에서 문장 형태는  $xu$  ( $x \in \Sigma^*$ ,  $u \in V^*$ )이다. 이 순간  $M$ 은 입력 스트링에서  $x$ 를 읽고, 스택에  $u\#$ 를 가지고자 한다.

$G$ 와 동등한 PA  $M$ 은 다음과 같다 (그림 3.11).

$$M = (\{p, q, r\}, \Sigma, V \cup \{\#\}, \Delta, p, \{r\})$$

1.  $((p, \epsilon, \#), (q, S\#)) \in \Delta$
2. 모든  $A \rightarrow ay$ 에 대하여  $((q, a, A), (q, y)) \in \Delta$
3.  $((q, \epsilon, \#), (r, \#)) \in \Delta$ .

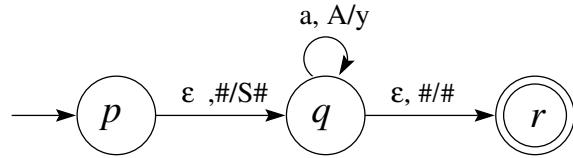


Figure 3.11:

$$S \xrightarrow{*} xu \text{인 경우에만 } (q, x, S\#) \vdash_M^* (q, \epsilon, u\#) \quad (3.1)$$

임을 귀납법으로 보이고자 한다. PA의 상황  $(q, x, S\#)$ 에서  $x$  이후에도 입력 글자가 있을 수 있으나 그 부분은 이 증명에 관여되지 않으므로 생략되었다.

(3.1)의  $\Rightarrow$  경우를 증명하자. 유도 횟수가 0일 때는  $x = \epsilon, u = S$  이므로 당연히 성립한다. 유도 횟수가  $i$  미만일 때는 (3.1)가 성립한다고 가정한다. 유도 횟수가  $i$ 일 때 마지막 생성규칙이  $A \rightarrow au_2$ 라고 하면

$$S \xrightarrow{*} x_1 Au_1 \Rightarrow x_1 au_2 u_1$$

이고  $x = x_1 a, u = u_2 u_1$ 이다. 귀납법 가정에 의해  $(q, x_1, S\#) \vdash_M^* (q, \epsilon, Au_1\#)$ 이다.  $A \rightarrow au_2$ 가 생성 규칙이므로  $((q, a, A), (q, u_2)) \in \Delta$ 이다. 따라서,

$$(q, x_1 a, S\#) \vdash_M^* (q, a, Au_1\#) \vdash_M (q, \epsilon, u_2 u_1\#).$$

마찬가지로 (3.1)의  $\Leftarrow$  경우를 증명할 수 있다. (3.1)에 의해서  $S \xrightarrow{*} w$ 인 경우에만  $(q, w, S\#) \vdash_M^* (q, \epsilon, \#)$ 이고 그래서

$$(p, w, \#) \vdash_M (q, w, S\#) \vdash_M^* (q, \epsilon, \#) \vdash_M (r, \epsilon, \#)$$

이다. 즉  $w \in L(G)$ 인 경우에만  $w \in L(M)$ 이다.

또 다른 증명:  $L$ 을 생성하는 문맥무관 문법을  $G = (V, \Sigma, S, P)$ 라 하자.  $G$ 를 흉내내는 PA는 다음과 같다.

$$(\{p, q, r\}, \Sigma, V \cup \Sigma \cup \{\#\}, \Delta, p, \{r\})$$

1.  $((p, \epsilon, \#), (q, S\#)) \in \Delta$
2. 모든  $A \rightarrow x$ 에 대하여  $((q, \epsilon, A), (q, x)) \in \Delta$
3. 모든  $a \in \Sigma$ 에 대하여  $((q, a, a), (q, \epsilon)) \in \Delta$
4.  $((q, \epsilon, \#), (r, \#)) \in \Delta$ .

이 경우 문법  $G$ 가 Greibach 표준형일 필요가 없다.  $\square$

#### 예제 3.24 Greibach 표준형인 문법

$$\begin{aligned} S &\rightarrow 0SA \mid 0A \\ A &\rightarrow 1 \end{aligned}$$

과 동등한 내리누름 오토마타를 구하라. 그림 3.12를 보라.

$$S \Rightarrow 0SA \Rightarrow 00AA \Rightarrow 001A \Rightarrow 0011$$

유도과정에 해당되는 PA의 전이를 설명한다.

#### 예제 3.25 문법

$$S \rightarrow aSb \mid bSa \mid SS \mid \epsilon$$

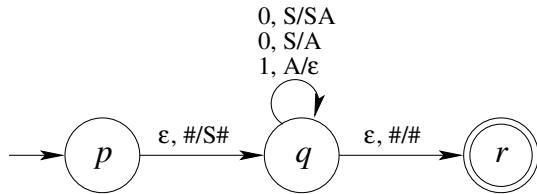


Figure 3.12:

과 동등한 내리누름 오토마타를 구하라. 그림 3.13을 보라.

$$S \Rightarrow aSb \Rightarrow abSab \Rightarrow abab$$

유도과정에 해당되는 PA의 전이를 설명한다.

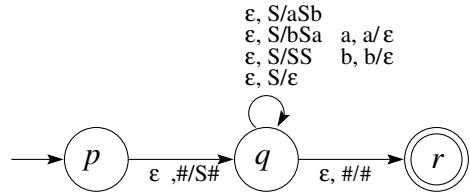


Figure 3.13:

**정리 3.6** 내리누름 오토마타  $M$ 에 대하여,  $L(M)$ 은 문맥무관 언어이다.

**증명.** 일반성을 잃지 않고  $M = (Q, \Sigma, \Gamma, \Delta, q_0, F)$ 에 대하여 다음과 같이 가정한다.  $a, b \in \Sigma \cup \{\epsilon\}$ ,  $A \in \Gamma$ 라고 하자.

- $M$ 은 최종상태가  $q_f$  한 개만 있고, 끝날 때에 스택에  $\#$ 만 남기고  $q_f$ 로 간다. (그렇지 않은 경우에는 스택에서 pop하는 전이를 더해주면 된다.)
- $M$ 의 전이는  $((p, a, \epsilon), (q, A))$  (push) 또는  $((p, a, A), (q, \epsilon))$  (pop) 형태만을 갖는다. (일반적인 전이는 push와 pop의 연산들로 바꿀 수 있다.)

주요 개념은  $M$ 이  $p$  상태에서 시작하여 동일한 스택으로 (스택에 push한 후 pop할 수는 있지만 아래쪽으로는 내려가지 않고)  $q$  상태에 도달하는 동안 입력에서  $x$ 를 읽으면, 문법에서는  $\langle pq \rangle \xrightarrow{*} x$ 이 되도록 하는 것이다 (그림 3.14).

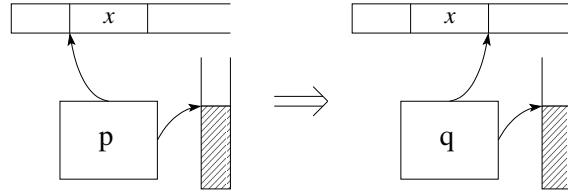


Figure 3.14:

$M$ 이  $x$ 를 읽는 동안 스택의 변화는 다음 두 가지중 하나이다.

- 처음에 push된 글자가 마지막에 pop되는 경우: 이 경우를  $\langle pq \rangle \rightarrow a\langle rs \rangle b$ 로 흉내내려고 한다. 여기에서  $a$ 와  $r$ 은 처음 전이 때 읽은 글자와 다음 상태이고,  $b$ 와  $s$ 는 마지막 전이 때 읽은 글자와 이전 상태이다.
- 처음에 push된 글자가 중간에 pop되는 경우: 이 경우를  $\langle pq \rangle \rightarrow \langle pr \rangle \langle rq \rangle$ 로 흉내내려고 한다. 여기에서  $r$ 은 처음에 push된 글자가 pop되었을 때의 상태이다.

$M$ 과 동등한 문맥무관 문법  $G = (V, \Sigma, S, P)$ 는 다음과 같다.

- $V$ 는  $p, q \in Q$ 에 대하여  $\langle pq \rangle$  형태의 변수들로 구성된다.
- $S$ 는  $\langle q_0 q_f \rangle$ 이다.
- $((p, a, \epsilon), (r, A)), ((s, b, A), (q, \epsilon)) \in \Delta$ 에 대하여  $\langle pq \rangle \rightarrow a\langle rs \rangle b$ 를 만든다.
- 모든  $p, q, r \in Q$ 에 대하여  $\langle pq \rangle \rightarrow \langle pr \rangle \langle rq \rangle$ 를 만든다.
- 모든  $q \in Q$ 에 대하여  $\langle qq \rangle \rightarrow \epsilon$ 을 만든다.

$$(p, x, \epsilon) \vdash_M^* (q, \epsilon, \epsilon) \text{인 경우에만 } \langle pq \rangle \xrightarrow{*} x \quad (3.2)$$

임을 귀납법으로 보이고자 한다. PA의 상황  $(p, x, \epsilon)$ 에서 스택에 글자가 있을 수 있으나 그 부분은 이 증명에 관여되지 않으므로 생략되었다 (그림 3.14에서 빗금친 부분).

(3.2)의  $\Rightarrow$  경우를 증명하자. 전이 횟수가 0인 경우에는  $p = q$ ,  $x = \epsilon$ 이고, 규칙 5에 의해  $\langle pp \rangle \rightarrow \epsilon$ 이므로 (3.2)이 성립한다. 전이 횟수가  $i$  미만일 때 (3.2)이 성립한다고 가정하자.

전이 횟수가  $i \geq 1$ 일 때 스택에서 (a)와 (b)중 하나가 발생한다. (a)가 발생하는 경우,  $A$ 를 처음에 스택에 push된 글자,  $a$ 와  $r$ 은 처음 전이 때 읽은 글자와 다음 상태,  $b$ 와  $s$ 는 마지막 전이 때 읽은 글자와 이전 상태라고 하자. 즉 처음 전이가  $((p, a, \epsilon), (r, A))$ , 마지막 전이가  $((s, b, A), (q, \epsilon))$ 이다. 규칙 3에 의해  $G$ 는  $\langle pq \rangle \rightarrow a\langle rs \rangle b$ 를 갖는다.  $x = ayb$ 라고 하자. 귀납법 가정에 의해  $(r, y, \epsilon) \vdash^* (s, \epsilon, \epsilon)$  이므로  $\langle rs \rangle \xrightarrow{*} y$ 이다. 따라서  $\langle pq \rangle \Rightarrow a\langle rs \rangle b \xrightarrow{*} x$ .

(b)가 발생하는 경우,  $r$ 을 처음에 push된 글자가 pop되었을 때의 상태라고 하고,  $y$ 와  $z$ 를 각각  $r$  이전과 이후에 읽은 입력 스트링이라고 하자. 규칙 4에 의해  $G$ 는  $\langle pq \rangle \rightarrow \langle pr \rangle \langle rq \rangle$ 를 갖는다. 귀납법 가정에 의해  $\langle pr \rangle \xrightarrow{*} y$ 이고  $\langle rq \rangle \xrightarrow{*} z$ 이다. 따라서  $\langle pq \rangle \Rightarrow \langle pr \rangle \langle rq \rangle \xrightarrow{*} x$ .

마찬가지로 (3.2)의  $\Leftarrow$  경우를 증명할 수 있다. (3.2)에 의해서

$$\langle q_0 q_f \rangle \xrightarrow{*} w \text{인 경우에만 } (q_0, w, \#) \vdash^* (q_f, \epsilon, \#).$$

즉  $w \in L(G)$ 인 경우에만  $w \in L(M)$ 이다.  $\square$

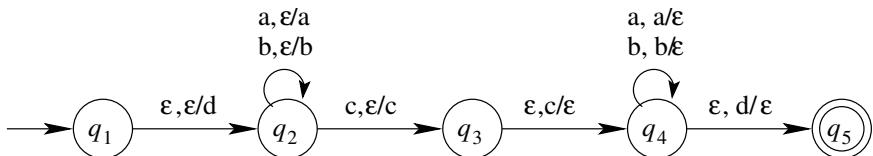


Figure 3.15:

**예제 3.26** 그림 3.15의 PA와 동등한 문맥무관 문법을 구하라.

$$\begin{aligned}
 \langle q_1 q_1 \rangle &\rightarrow \epsilon \\
 &\dots \\
 \langle q_5 q_5 \rangle &\rightarrow \epsilon \\
 \\ 
 \langle q_1 q_3 \rangle &\rightarrow \langle q_1 q_2 \rangle \langle q_2 q_3 \rangle \\
 &\dots \\
 \langle q_1 q_5 \rangle &\rightarrow \langle q_1 q_3 \rangle \langle q_3 q_5 \rangle \\
 \\ 
 \langle q_1 q_5 \rangle &\rightarrow \langle q_2 q_4 \rangle \\
 \langle q_2 q_4 \rangle &\rightarrow c \langle q_3 q_3 \rangle \\
 \langle q_2 q_4 \rangle &\rightarrow a \langle q_2 q_4 \rangle a \\
 \langle q_2 q_4 \rangle &\rightarrow b \langle q_2 q_4 \rangle b
 \end{aligned}$$

o) 문법으로 스트링  $abcbba$ 를 생성하는 과정:

$$\begin{aligned}
 \langle q_1 q_5 \rangle &\Rightarrow \langle q_2 q_4 \rangle \\
 &\stackrel{*}{\Rightarrow} ab \langle q_2 q_4 \rangle ba \\
 &\Rightarrow abc \langle q_3 q_3 \rangle ba \\
 &\Rightarrow abcbba.
 \end{aligned}$$

### 3.7 문맥무관 언어의 성질

어떤 언어가 문맥무관 언어가 아님을 보이기 위해서는 다음 정리를 사용한다.

**정리 3.7 (펌프 정리)**  $L$ 을 문맥무관 언어라고 하면, 다음을 만족하는 양의 정수  $t$ 가 존재한다. 길이가  $t$  이상인 임의의 스트링  $w \in L$  는  $w = uvxyz$ 로 표현되며 여기서

1.  $|vxy| \leq t$ ,
2.  $|vy| \geq 1$ ,
3. 모든  $i \geq 0$ 에 대하여  $uv^i xy^i z \in L$ 이다.

**증명.**  $L - \{\epsilon\}$ 를 생성하는 Chomsky 표준형의 문법을  $G = (V, \Sigma, S, P)$ 라고 하자.  $G$ 가 Chomsky 표준형이므로  $w \in L(G)$ 의 파스 트리의 높이가  $i$ 이면  $|w| \leq 2^{i-1}$ 임을 귀납법으로 보일 수 있다.

$|V| = k$ 라고 하면,  $t$ 는  $2^k$ 이다.  $w \in L(G)$ 이고  $|w| \geq t$ 이면,  $w$ 의 파스 트리의 높이가  $k+1$  이상이다. 루트에서 가장 긴 경로를  $P$ 라고 하면,  $P$ 의 길이는  $k+1$  이상이고 그 경로 상에  $k+1$  이상의 변수가 나타난다.  $G$ 는  $k$  개의 변수만을 가지므로  $P$ 를 리프에서부터 올라가면, 반복되는 변수가 존재한다. 처음 반복되는 변수를  $A$ 라 하고  $A$  이름의 노드를  $p, q$  ( $p$ 가  $q$ 의 아래쪽)라고 하면, 리프부터  $q$ 까지의 길이는  $k+1$  이하이다. 그림 3.16를 보라.  $P$ 가 가장 긴 경로이므로  $q$ 의 높이는  $k+1$  이하이다. 따라서  $q$ 를 루트로 갖는 부분 트리의 열매 스트링  $vxy$ 의 길이는  $2^k$  이하이다 (즉  $|vxy| \leq t$ ). 여기에서  $x$ 는  $p$ 를 루트로 갖는 부분 트리의 열매 스트링이다. 즉  $w$ 는  $uvxyz$ 로 표시된다. 또한  $q$ 에서 사용된 생성규칙은  $A \rightarrow BC$  형태이므로  $v, y$  둘 중의 하나는  $\epsilon$ 가 될 수 없다 (즉  $|vy| \geq 1$ ).

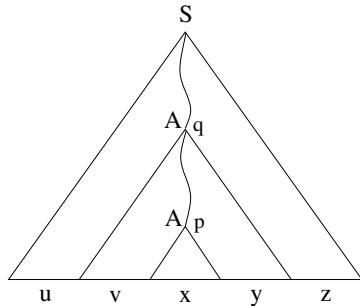


Figure 3.16:

$q$ 와  $p$ 에서 시작하는 유도 과정은

$$A \xrightarrow{*} vAy \text{와 } A \xrightarrow{*} x$$

이므로 모든  $i \geq 0$ 에 대하여  $A \xrightarrow{*} v^i xy^i$ 이다.  $\square$

**예제 3.27**  $L = \{a^n b^n c^n : n \geq 0\}$ 는 문맥무관 언어가 아님을 증명하라.

$L$ 이 문맥무관 언어라고 가정하자. 그러면 정리 3.7를 만족하는  $t$ 가 존재한다. 스트링  $w = a^t b^t c^t$ 를 고려하자.  $|vxy| \leq t$ 이므로  $vy$ 는  $a, b, c$  세 글자를 모두 포함할 수 없다. 따라서  $uv^2xy^2z$ 는 같은 수의  $a, b, c$ 를 가질 수 없다.

**예제 3.28**  $L = \{uu : u \in \{a, b\}^*\}$ 는 문맥무관 언어가 아니다.

$L$ 이 문맥무관 언어라고 가정하고  $w = a^t b^t a^t b^t$ 를 선택한다.  $vy$ 가 어디에 위치하든지  $uxz \notin L$ 이다.

**예제 3.29**  $L = \{a^n : n\text{은 소수}\}$ 는 문맥무관 언어가 아니다.

$L$ 이 문맥무관 언어라고 가정하면 정리 3.7를 만족하는  $t$ 가 존재한다.  $p$ 를  $t$ 보다 큰 소수라고 하자. 즉  $a^p \in L$ .  $q = |vy|, r = |uxz|$ 라고 하면,  $q \geq 1, r \geq 1$ 이다.  $i = q + r + 1$ 을 선택하면  $iq + r = (q + r)(q + 1)$ 이고  $q + r \geq 2, q + 1 \geq 2$ 이므로  $uv^i xy^i z \notin L$ 이다.

**예제 3.30**  $L = \{a^n b^{n^2} : n \geq 1\}$ 은 문맥무관 언어가 아니다.

**정리 3.8** 문맥무관 언어 종류는 (1) 합집합, (2) 접합, (3) Kleene 스타 연산에 대하여 닫혀 있다.

**증명.**  $G_1 = (V_1, \Sigma_1, S_1, P_1)$ 과  $G_2 = (V_2, \Sigma_2, S_2, P_2)$ 를 문맥무관 문법이라고 하자. 일반성을 잃지 않고  $V_1$ 와  $V_2$ 는 서로 교차하지 않는 집합이라고 하자.

(1) 새로운 변수  $S$ 를 사용하여

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, S, P_1 \cup P_2 \cup \{S \rightarrow S_1 \mid S_2\})$$

가  $L(G_1) \cup L(G_2)$ 를 생성한다.

(2) 합집합의 경우와 마찬가지이나,  $S \rightarrow S_1 \mid S_2$  대신  $S \rightarrow S_1 S_2$ 를

첨가한다.

(3)

$$G = (V_1 \cup \{S\}, \Sigma_1, S, P_1 \cup \{S \rightarrow SS_1 \mid \epsilon\})$$

이  $L(G_1)^*$ 를 생성한다.  $\square$

**정리 3.9** 문맥무관 언어 종류는 (1) 교집합, (2) 여집합 연산에 대하여 닫혀 있지 않다.

**증명.** (1)  $L_1 = \{a^n b^n c^m : n, m \geq 0\}$ ,  $L_2 = \{a^n b^m c^m : n, m \geq 0\}$ 라고 하자.  $L_1$ 과  $L_2$ 는 문맥무관 언어이다. 즉  $L_1$ 은 다음 문법에 의해 생성된다.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aAb \mid \epsilon \\ B &\rightarrow cB \mid \epsilon \end{aligned}$$

그러나  $L_1 \cap L_2$ 는 문맥무관 언어가 아니다.

(2)

$$L_1 \cap L_2 = \overline{L_1} \cup \overline{L_2}$$

이므로 여집합에 대해 닫혀 있으면 교집합에 대해서도 닫혀 있어야 된다.  $\square$

**정리 3.10**  $L$ 이 문맥무관 언어이고  $R$ 이 정규언어이면,  $L \cap R$ 은 문맥무관 언어이다.

**증명.**  $M_1 = (Q_1, \Sigma, \Gamma, \Delta_1, q_1, F_1)$ 을  $L$ 을 받아들이는 PA,  $M_2 = (Q_2, \Sigma, \delta, q_2, F_2)$ 를  $R$ 을 받아들이는 DFA라고 하자. 주요 개념은  $M_1$ 과  $M_2$ 를 동시에 돌리는 PA  $M$ 을 만드는 것이다. 구체적으로

$$M = (Q_1 \times Q_2, \Sigma, \Gamma, \Delta, (q_1, q_2), F_1 \times F_2)$$

인데, 전이  $\Delta$ 는 다음과 같다.

- $((p, a, A), (p', u)) \in \Delta_1$ 이고  $\delta(q, a) = q'$ 이면,  
 $((p, q), a, A), ((p', q'), u)) \in \Delta$ 이다.
- $((p, \epsilon, A), (p', u)) \in \Delta_1$ 이면, 모든  $q \in Q_2$ 에 대하여  
 $((p, q), \epsilon, A), ((p', q), u)) \in \Delta$ 이다.

$w \in L \cap R$ 인 경우에만  $w \in L(M)$ 임을 보일 수 있다.  $\square$

**예제 3.31 (소속문제)** CFG  $G = (V, \Sigma, S, P)$ 과  $w \in \Sigma^*$ 가 주어졌을 때,  $w$ 가  $L(G)$ 의 원소인지 아닌지를 결정하라. 이 과정에서  $w \in L(G)$ 인 경우 파스 트리를 만들게 되므로 소속문제를 푸는 과정을 파싱(parsing)이라고 부른다. ( $PA$ 를 만들지 않는 이유는?)

CYK (Cocke, Younger, Kasami) 알고리즘은 동적 계획법(dynamic programming)의 일종으로 다음과 같다.

$G$ 가 Chomsky 표준형이고  $w = a_1a_2 \cdots a_n$ 이라고 하자.  $V_{ij} = \{A \in V : A \xrightarrow{*} a_i \cdots a_j\}$ 라고 하자. 그러면,  $S \in V_{1n}$ 인 경우에만  $w \in L(G)$ 이다. 따라서  $V_{ij}$ 를  $d = j - i$ 가 작은 값에서 큰 값으로 가면서 계산하면 된다.  $G$ 에 생성 규칙이 상수 개 있으면, CYK 알고리즘은  $O(n^3)$  시간이 걸린다.

**예제 3.32** 다음 문맥무관 문법이  $baaba$ 를 생성하는지 CYK 알고리즘을 사용하여 확인하라.

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

```

for  $i = 1$  to  $n$  do
     $V_{ii} = \{A : A \rightarrow a_i\}$ 
od
for  $d = 1$  to  $n - 1$  do
    for  $i = 1$  to  $n - d$  do
         $j = i + d$ 
         $V_{ij} = \emptyset$ 
        for  $k = i$  to  $j - 1$  do
             $V_{ij} = V_{ij} \cup \{A : A \rightarrow BC, B \in V_{ik}, C \in V_{k+1,j}\}$ 
        od
    od
od

```

Figure 3.17:

CYK 알고리즘이 만드는 표는 다음과 같다.

$i \setminus j$	1	2	3	4	5
1	$B$	$SA$	$\emptyset$	$\emptyset$	$SAC$
2		$AC$	$B$	$B$	$SAC$
3			$AC$	$SC$	$B$
4				$B$	$SA$
5					$AC$

$S \in V_{15}$  이므로  $baaba$ 는 이 문법에 의해 생성된다.

### 3.8 결정 내리누름 오토마타

내리누름 오토마타가 한 시점에 두 개 이상의 전이를 가질 수 없으면 이를 결정 내리누름 오토마타라고 부른다. 결정 내리누름 오토마타 (deterministic pushdown automata)를 줄여서 DPA로 부르기로 하자. DPA가 입력이나 스택을 읽지 않고 전이할 수 있도록 하는 것이 편리하므로 전이함수  $\delta$ 를 다음 조건을 만족하는  $Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup$

$\{\epsilon\}$ )에서  $Q \times \Gamma^*$ 로의 부분함수(partial function)로 정의한다.

- $\delta(q, \epsilon, A)$ 가 정의되면, 모든  $a \in \Sigma$ 에 대하여  $\delta(q, a, A)$ 가 정의 되지 않는다.
- $\delta(q, a, \epsilon)$ 가 정의되면, 모든  $A \in \Gamma$ 에 대하여  $\delta(q, a, A)$ 가 정의 되지 않는다.

또한 DPA가 입력의 끝을 알 수 있도록 입력 다음 칸에 공백 글자 #가 들어 있다고 가정한다. DPA가 받아들이는 언어를 결정 문맥무관 언어라고 부른다.

**예제 3.33**  $\{0^n 1^n : n \geq 0\}$ 을 받아들이는 DPA는 그림 3.18과 같다.

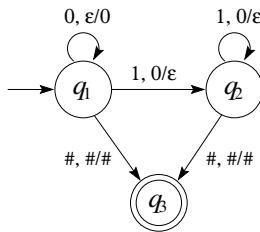


Figure 3.18:

**정리 3.11** 결정 문맥무관 언어는 여집합에 대하여 닫혀 있다.

**증명.** (설명) DPA에서는 최종상태와 그렇지 않은 상태를 맞바꿈으로 여집합을 받아들이는 DPA를 만들 수 없다. 왜냐하면 계속 입력을 읽지 않고 무한루프에 들어가는 경우가 있기 때문이다. 따라서 이 증명을 위해서는 주어진 DPA를 매번 한 글자씩 입력을 읽는 DPA로 바꾼 다음에 최종상태와 그렇지 않은 상태를 바꿔야 한다.  $\square$

**정리 3.12** 결정 문맥무관 언어 종류는 문맥무관 언어 종류의 진부분집합이다.

**증명.**  $L = \{a^i b^j c^k : i \neq j \text{ 또는 } j \neq k\}$ 를 고려하자.  $L$ 은 문맥무관 언어이다.

$L$ 이 결정 문맥무관 언어라고 가정하자. 그러면  $\bar{L}$ 도 결정 문맥무관이고 따라서 문맥무관이다.  $\bar{L} \cap L(a^* b^* c^*)$ 는 문맥무관이 되는데, 이것이  $\{a^n b^n c^n : n \geq 0\}$ 이므로 모순이다.  $\square$

### 3.9 하향 파싱

입력을 받아들여서 파스 트리를 만드는 과정을 파싱(parsing)이라고 한다. CYK 알고리즘을 사용하면 파싱을 할 수 있으나, 큰 프로그램을 파싱할 경우 너무 많은 시간이 걸린다. 일반적으로 컴파일러는  $O(n)$  시간에 파싱하는 알고리즘을 필요로 한다.

**예제 3.34** 예제 3.24의 문법

$$\begin{aligned} S &\rightarrow 0SA \mid 0A \\ A &\rightarrow 1 \end{aligned}$$

과 스트링 0011이 주어졌을 때, 파싱을 해보자.

스트링의 첫 번째 글자가 0이므로 시작변수에서  $S \rightarrow 0SA$  또는  $S \rightarrow 0A$ 를 적용해야 한다. 어느 생성규칙을 적용해야 할지 알 수 없지만  $S \rightarrow 0SA$ 를 적용해보면, 유도과정

$$S \Rightarrow 0SA$$

을 얻는다. 변수가  $S$ 이고 두 번째 글자도 0이므로 앞의 경우와 마찬 가지인데,  $S \rightarrow 0A$ 를 적용해보면

$$S \xrightarrow{*} 00AA$$

을 얻는다. 변수가  $A$ 이고 세 번째 글자가 1이므로  $A \rightarrow 1$ 을 적용해야 한다. 네 번째 글자의 경우도 마찬가지이다. (참고로, 입력 글자를

두 개 읽으면 어느 생성규칙을 적용해야 할지 바로 알 수 있다. 현재 변수가  $S$ 일 때, 00을 읽으면  $S \rightarrow 0SA$ 를, 01을 읽으면  $S \rightarrow 0A$ 를 적용하면 된다.)

현재 변수  $A$ 와 글자  $a$ 에 대해서  $A \rightarrow a \dots$  형태의 생성규칙이 하나만 있으면 유도과정을  $O(n)$ 에 완성할 수 있다. 즉,  $O(n)$  시간에 파싱할 수 있다.

하향 파싱(topdown parsing)은 위의 개념을 일반화시킨 것이다. 하향 파싱은 입력을 왼쪽에서 오른쪽으로 읽으면서 파스 트리를 위에서 아래로 만들어 가는 파싱이다. 파스 트리를 위에서 아래로 만들어 가는 과정은 좌측 유도를 만드는 것과 동일하다.

다음 문법  $G$ 에 대하여  $w \in L(G)$ 를 파싱하는 DPA를 구하라.

$$\begin{array}{lcl} E & \rightarrow & E + T \mid T \\ T & \rightarrow & T * F \mid F \\ F & \rightarrow & \text{id} \end{array}$$

여기에서  $\text{id}$ 는 하나의 글자로 간주한다.

하향 파싱이 용이하도록 다음 두 규칙을 사용하여 문법을 바꾼다.

1.  $A \rightarrow Ax$  형태의 생성규칙을 없앤다.  $A$ 가 좌변에 나타나는 모든 생성규칙이

$$\begin{array}{lcl} A & \rightarrow & Ax_1 \mid \dots \mid Ax_r \\ A & \rightarrow & y_1 \mid \dots \mid y_t \end{array}$$

이면, 이를

$$\begin{array}{lcl} A & \rightarrow & y_1 A' \mid \dots \mid y_t A' \\ A' & \rightarrow & x_1 A' \mid \dots \mid x_r A' \mid \epsilon \end{array}$$

로 바꾼다.

2. 생성규칙의 우변이 같은 스트링으로 시작되지 않도록 고친다.

$$A \rightarrow xy_1 \mid \cdots \mid xy_r$$

$(x \neq \epsilon, r \geq 2)$  이면, 이를

$$\begin{aligned} A &\rightarrow xA' \\ A' &\rightarrow y_1 \mid \cdots \mid y_r \end{aligned}$$

로 바꾼다.

문법  $G$ 에 규칙 1을 적용하면

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \mid \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \mid \epsilon \\ F &\rightarrow \text{id} \end{aligned}$$

**예제 3.35** 위의 문법으로  $\text{id} + \text{id}$ 를 생성하는 과정은 다음과 같다.

$$\begin{aligned} E &\Rightarrow TE' \\ &\Rightarrow T + TE' \\ &\Rightarrow FT' + TE' \\ &\stackrel{*}{\Rightarrow} F + T \\ &\stackrel{*}{\Rightarrow} F + F \\ &\stackrel{*}{\Rightarrow} \text{id} + \text{id} \end{aligned}$$

DPA가 입력 글자를 하나 읽어서 현재 변수에서 어떤 생성규칙을 적용할지 결정하도록 만들려고 한다. 이를 위해서 변수  $A$ 와 입력 글자  $a$ 가 주어지면 적용할 생성규칙을 파싱 표  $M(A, a)$ 에 저장한다.

각 생성규칙  $A \rightarrow x$ 에 대하여

1.  $x$ 에서 유도되는 첫 번째 글자가  $a$ 이면 (즉  $x \xrightarrow{*} ay$ )  $M(A, a)$ 에  $A \rightarrow x$ 를 넣는다.
2.  $x \xrightarrow{*} \epsilon$ 인 경우,  $A$  다음에 나올 수 있는 글자가  $a$ 이면  $M(A, a)$ 에  $A \rightarrow x$ 를 넣는다.

$G$ 의 생성규칙을 고려하면

- $E \rightarrow TE'$ :  $T$ 는  $F$ 를 거쳐  $\text{id}$ 를 생성하므로  $M(E, \text{id})$ 에 이 생성규칙을 넣는다.
- $E' \rightarrow \epsilon$ :  $E \rightarrow TE'$ 에 의해  $E'$ 은 입력의 끝에 위치하므로 그 다음에  $\#$ 이 올 수 있어서  $M(E', \#)$ 에 넣는다.
- $T' \rightarrow \epsilon$ :  $T'$ 도 입력의 끝에 위치할 수 있으므로  $M(T', \#)$ 에 넣고 또한  $T'$  다음에  $+$ 이 올 수 있으므로  $M(T', +)$ 에 넣는다.

이와 같이 파싱 표를 만들면 그림 3.19과 같다. 파싱 표를 만드는 자세한 방법은 [ASU]를 참고하라.

	$\text{id}$	$+$	$*$	$\#$
$E$	$E \rightarrow TE'$			
$E'$		$E' \rightarrow +TE'$		$E' \rightarrow \epsilon$
$T$	$T \rightarrow FT'$			
$T'$		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$	$T' \rightarrow \epsilon$
$F$	$F \rightarrow \text{id}$			

Figure 3.19:

이제 DPA는 입력 글자를 읽은 후 위의 파싱 표에 의해 생성규칙을 적용한다 (그림 3.20).

입력  $\text{id} + \text{id} * \text{id} \#$ 에 대한 DPA의 동작과정이 그림 3.21에 있다. 상태나 입력, 스택이 변하지 않은 경우에는 -로 표시하였다. 이와 같이 만들어진 파스 트리는 그림 3.22에 있다.

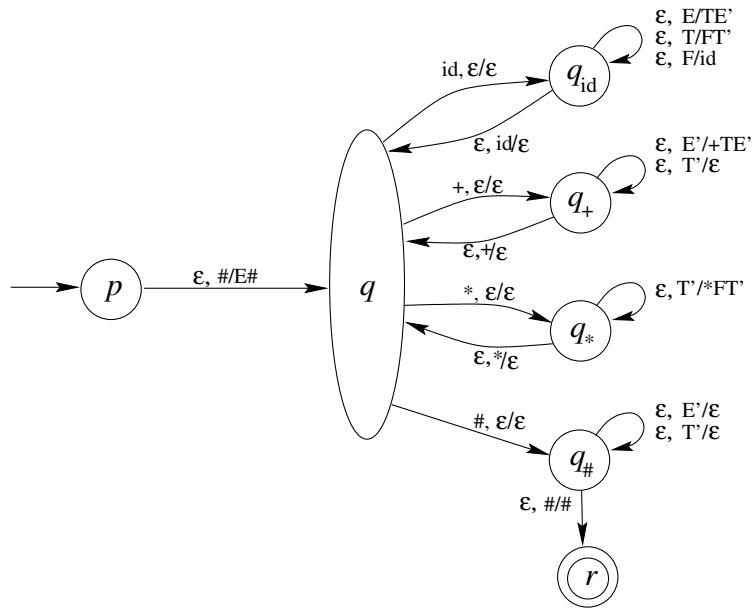


Figure 3.20:

`if`문을 나타내는 문법을 간략히 하면 다음과 같다.

$$\begin{aligned} S &\rightarrow iCtSeS \mid iCtS \mid x \\ C &\rightarrow a \end{aligned}$$

여기에서  $i, t, e$ 는 각각 `if`, `then`, `else`를 나타낸다.

규칙 2를 적용하면

$$\begin{aligned} S &\rightarrow iCtSS' \mid x \\ S' &\rightarrow eS \mid \epsilon \\ C &\rightarrow a \end{aligned}$$

를 얻는다. 이제 파싱 표를 구하자.

- $S' \rightarrow eS$ 는  $M(S', e)$ 에 들어가야 된다.
- $S' \rightarrow \epsilon$ 을 고려하면,  $iCtSS'$ 에서  $S \rightarrow iCtSS'$ 와  $S' \rightarrow eS$ 를 사

상태	입력	스택	생성규칙
$p$	$\text{id} + \text{id} * \text{id} \#$	#	
$q$	—	$E \#$	
$q_{id}$	$+ \text{id} * \text{id} \#$	—	
—	—	$TE' \#$	$E \rightarrow TE'$
—	—	$FT'E' \#$	$T \rightarrow FT'$
—	—	$\text{id}T'E' \#$	$F \rightarrow \text{id}$
$q$	—	$T'E' \#$	
$q_+$	$\text{id} * \text{id} \#$	—	
—	—	$E' \#$	$T' \rightarrow \epsilon$
—	—	$+TE' \#$	$E' \rightarrow +TE'$
$q$	—	$TE' \#$	
$q_{id}$	$*\text{id} \#$	—	
—	—	$FT'E' \#$	$T \rightarrow FT'$
—	—	$\text{id}T'E' \#$	$F \rightarrow \text{id}$
$q$	—	$T'E' \#$	
$q_*$	$\text{id} \#$	—	
—	—	$*FT'E' \#$	$T' \rightarrow *FT'$
$q$	—	$FT'E' \#$	
$q_{id}$	#	—	
—	—	$\text{id}T'E' \#$	$F \rightarrow \text{id}$
$q$	—	$T'E' \#$	
$q_{\#}$	$\epsilon$	—	
—	—	$E' \#$	$T' \rightarrow \epsilon$
—	—	#	$E' \rightarrow \epsilon$
$r$	—	—	

Figure 3.21:

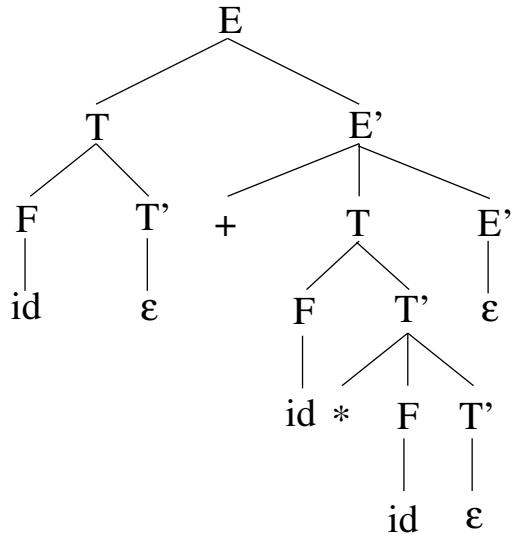


Figure 3.22:

용하면  $S'$  다음에  $e$ 가 나오므로  $M(S', e)$ 에  $S' \rightarrow \epsilon$  들어가야 된다.

원래 문법이 애매하기 때문에  $M(S', e)$ 에 두 개의 생성규칙이 들어가게 된다. 이 경우 두 생성규칙 중  $S' \rightarrow eS$ 를 선택하면 `else`를 가장 가까운 `if`에 붙이는 결과가 된다. 파싱 표를 완성하면 그림 3.23과 같다.

	$i$	$e$	$x$	$a$	#
$S$	$S \rightarrow iCtSS'$				
$S'$		$S' \rightarrow eS$			
$C$			$S \rightarrow x$	$C \rightarrow a$	$S' \rightarrow \epsilon$

Figure 3.23:

## 3.10 연습 문제

### 1. 문맥무관 문법

1.1. Palindrome이란 앞으로 읽으나 뒤로 읽으나 같은 스트링이다.

- (a) 알파벳  $\{0, 1\}$ 에서 palindrome을 생성하는 문맥무관 문법을 구하라.
- (b) 알파벳  $\{a, b, c\}$ 에서 palindrome을 생성하는 문맥무관 문법을 구하라.

1.2. 다음 언어에 대한 문맥무관 문법을 구하라.

- (a)  $\{a^m b^n c^u d^v : m + n = u + v, m, n, u, v \geq 0\}$ .
- (b)  $\{0^n 1^m : n \leq m \leq 2n\}$ .
- (c)  $\{0^n 1^m : \frac{n}{2} \leq m \leq n\}$ .
- (d)  $\{a^m b^n c^u d^v : m + n = u + v\}$ .
- (e)  $\{a^n b^n : n \geq 0\}$ .
- (f)  $\{a^n b^m : n \neq m\}$ .
- (g)  $\{a^n b^m \in \{a, b\}^* : n + 1 \neq m\}$ .
- (h)  $\{a^n b^m : 2n \leq m \leq 3n, n \geq 0, m \geq 0\}$ .
- (i)  $\{a^n b^m c^k : n + m = k\}, n, m, k \geq 0$ .
- (j)  $\{uvwv^R : u, v, w \in \{a, b\}^*, |u| = |w| = 2\}$ .
- (k)  $\{w \in \{a, b\}^* : w\text{는 같은 수의 } a\text{와 } b\text{를 갖는다}\}$ .

1.3. 알파벳  $\{a, b\}$ 에서 정규식을 생성하는 문맥무관 문법을 구하라, 예를 들면  $(a \cdot b)^* + a \cdot (b \cdot a + \epsilon)$ . 문법은 연산의 일반적인 우선 순위를 따른다.

1.4. 2 종류의 괄호 ( ), [ ]를 포함하는, 올바로 중첩된 괄호 (nested parenthesis) 구조가 의미하는 바를 정의 하라. 예를 들어, ( [ ] ), ( [ [ ] ] ) [ () ]는 올바로 중첩된 괄호이고, ( [ ] ), ( ( ) ]는 올바로 중첩된 괄호가 아니다. 정의를 이용하여, 올바로 중첩된 괄호를 생성하는 문맥무관 문법을 구하라.

## 2. 정규문법

- 2.1.  $L = \{a^n b^m : n \text{ 과 } m \text{ 은 홀수}\}$ 에 대한 정규문법을 구하라. (힌트:  $L$ 에 대한 DFA를 구하고  $L$ 에 대한 정규문법을 구하라.)
- 2.2.  $L(0(01 + 10)^*)$ 을 생성하는 정규문법을 구하라.
- 2.3.  $\{w \in \{a, b\}^* \mid w\text{는 }ab\text{를 부분스트링으로 갖지 않는다 }\}$ 을 생성하는 정규문법을 구하라.
- 2.4.  $a(ba + ab)^*bb$ 를 생성하는 정규문법을 구하라.
- 2.5.  $\{w \in \{a, b\}^* : N_a(w)\text{와 }N_b(w)\text{는 모두 짝수}\}$ 를 생성하는 정규문법을 구하라.

### 3. 파스 트리

- 3.1.  $L = \{ww^R : w \in \{0, 1\}^*\}$ 이 본질적으로 애매하지 않은지 증명하라.
- 3.2. 다음 문법에 대하여  $aabbbaa$ 의 좌측 유도와 파스 트리를 구하라.

$$\begin{aligned} S &\rightarrow aAS \mid a \\ A &\rightarrow SbA \mid SS \mid ba \end{aligned}$$

- 3.3. 다음 문법이 애매한지 증명하라.

$$S \rightarrow aSbS \mid bSaS \mid \epsilon.$$

- 3.4. 문맥무관 문법  $G$ 를 고려하자.

$$\begin{aligned} S &\rightarrow aB \mid bA \\ A &\rightarrow a \mid aS \mid bAA \\ B &\rightarrow b \mid bS \mid aBB \end{aligned}$$

$aaabbbba \in L(G)$ 의 좌측 유도와 파스 트리를 구하라.

3.5. 다음 문법이 애매한지 증명하라.

$$\begin{aligned} S &\rightarrow \text{if } C \text{ then } S \\ S &\rightarrow \text{if } C \text{ then } S \text{ else } S \\ S &\rightarrow x \mid y \\ C &\rightarrow p \mid q \end{aligned}$$

3.6. (a) “문맥무관 문법의 애매성”과 “문맥무관 문법의 본질적인 애매성”의 정의를 설명하라.

(b) 다음 문법이 애매한지 증명하라.

$$\begin{aligned} E &\rightarrow E + E \\ E &\rightarrow E - E \\ E &\rightarrow (E) \\ E &\rightarrow a \mid b \mid c \end{aligned}$$

3.7. 예제 3.15(if 문)의 문법에서 `else`는 가장 가까운 (대응되지 않은) `then`에 대응된다고 가정했을 때, if 문을 생성하는 애매하지 않은 문법을 구하라. (힌트: `then`과 `else`를 대응시키는 `if`를 생성하는 변수  $M$ 과 `then`과 `else`를 대응시키지 않는 `if`를 생성하는 변수  $U$ 를 사용하라.)

#### 4. 표준형

4.1. 언어  $\{0^n 1^n : n \geq 1\}$ 에 대한 Chomsky 표준형을 구하라.

4.2. 다음 문법에 대한 Chomsky 표준형을 구하라.

(a)

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

(b)

$$\begin{aligned} S &\rightarrow bA \mid aB \\ A &\rightarrow aS \mid bAA \mid a \\ B &\rightarrow bS \mid aBB \mid b \end{aligned}$$

(c)

$$\begin{aligned} S &\rightarrow abAB \\ A &\rightarrow bAB \mid \epsilon \\ B &\rightarrow BAa \mid A \mid \epsilon \end{aligned}$$

4.3. 다음 문법과 동등하면서 단위 생성규칙이 없는 문맥무관 문법을 구하라.

$$\begin{aligned} S &\rightarrow Ab \mid aB \mid B \\ A &\rightarrow a \mid B \\ B &\rightarrow A \mid b \end{aligned}$$

4.4. 다음 문법에 대하여 모든  $\epsilon$  생성규칙을 제거하라.

$$S \rightarrow abB \mid AB$$

$$\begin{aligned} A &\rightarrow a \mid \epsilon \\ B &\rightarrow bA \mid A \end{aligned}$$

## 5. 내리누름 오토마타

5.1. 다음 언어를 받아들이는 PA를 구하라.

- (a)  $\{0^n 1^m : n \leq m \leq 2n\}$ .
- (b)  $\{a^n b^{n+m} c^m : n \geq 1, m \geq 1\}$ .
- (c)  $\{0^n 1^m : n \neq m\}$ .
- (d)  $\{a^n b^m : n \leq m \leq 2n\}$ .

## 6. 내리누름 오토마타와 문맥무관 언어

6.1. 다음 문법과 동등한 PA를 구하라.

(a)

$$S \rightarrow aSb \mid SS \mid \epsilon$$

(b)

$$\begin{aligned} S &\rightarrow aAS \mid a \\ A &\rightarrow SbA \mid SS \mid ba \end{aligned}$$

(c)

$$\begin{aligned} S &\rightarrow aAA \\ A &\rightarrow aS \mid bS \mid a \end{aligned}$$

6.2.  $L = \{a^i b^j c^k : i \neq j \text{ 또는 } j \neq k\}$ 에 대한 문맥무관 문법을 구하고,  $L$ 에 대한 PA를 구하라.6.3.  $M = (K, \Sigma, \Gamma, \Delta, q, F)$ ,  $K = \{q\}$ ,  $\Sigma = \{a, b\}$ ,  $\Gamma = \{a\}$ ,  $\Delta = \{((q, a, e)(q, a)), ((q, b, a)(q, e))\}$ ,  $F = \{q\}$ 인 PA와

동등한 문맥무관 문법을 구하라. 단, PA에서 CFG로의 일반적인 생성 방법을 이용해야 한다.

6.4. (a) 다음 문법을 Greibach 표준형으로 변환하라.

$$S \rightarrow aSb \mid ab$$

(b) (a)에서 얻은 문법으로부터, CFG에서 PA로의 일반적인 생성 방법을 이용하여 동등한 PA를 구하라.

6.5. (a) 다음 문법을 Greibach 표준형으로 변환하라.

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aA \mid bB \mid b \\ B &\rightarrow b \end{aligned}$$

(b) (a)에서 얻은 문법으로부터, CFG에서 PA로의 일반적인 생성 방법을 이용하여 동등한 PA를 구하라.

## 7. 문맥무관 언어의 성질

7.1. 다음 언어가 문맥무관 언어인지 아닌지 증명하라.

- (a)  $\{0^n 1^m : m \leq n^2\}$
- (b)  $\{0^n 1^m 0^m 1^n : n, m \geq 0\}$
- (c)  $\{a^{n^2} : n \geq 0\}$
- (d)  $\{0^n 1^m : m = n^2, n \geq 1\}$
- (e)  $\{w \in \{a, b, c\}^* : N_a(w) = N_b(w) = N_c(w)\}$
- (f)  $L = \{0^n 1^n : n \geq 1\}$  일 때,  $L^*$
- (g)  $\{a^n b^m a^n b^m : n \geq 0, m \geq 0\}$
- (h)  $\{a^{n!} : n \geq 0\}$ .
- (i)  $\{vw : v, w \in \{a, b\}^*, v^R \neq w\}$ .

- 7.2. 문맥무관 언어 종류가 교집합 연산에 닫혀 있지 않음을 증명하라.
- 7.3. 결정 문맥무관 언어 종류가 여집합 연산에 닫혀 있고 교집합 연산에는 닫혀 있지 않음을 증명하라.
- 7.4.  $L = \{w : N_a(w) = N_b(w)\}$ 에 대하여 다음 문법이 LL 문법이 아님을 증명하라.

$$S \rightarrow SS \mid aSb \mid bSa \mid \epsilon$$

- 7.5. 다음 문법이 생성하는 언어에 스트링  $aabbb$ 가 포함되는지 CYK 알고리즘을 사용하여 확인하라.

$$\begin{array}{l} S \rightarrow AB \\ A \rightarrow BB \mid a \\ B \rightarrow AB \mid b \end{array}$$

#### 8. 결정 내리누름 오토마타

- 8.1.  $L = a^* \cup \{a^n b^n : n \geq 0\}$ 을 받아들이는 DFA를 구하라.
- 8.2.  $L = \{a^n b^{2n} : n \geq 0\}$ 에 대한 DFA를 구하라.

#### 9. 하향 파싱

- 9.1. 다음 파싱 표를 기초로 하여 하향 파싱을 위한 DFA를 구하라.

	$i$	$e$	$x$	$a$	#
$S$	$S \rightarrow iCtSS'$		$S \rightarrow x$		
$S'$		$S' \rightarrow eS$			$S' \rightarrow \epsilon$
$C$				$C \rightarrow a$	



# Chapter 4

## 튜링기계

### 4.1 튜링기계

튜링 기계(Turing machine)를 줄여서 TM으로 부르기로 하자. TM은 제어기와 테입으로 구성되는데, 테입 헤드는 좌우로 움직일 수 있고 읽고 쓰는 것이 가능하다. 그림 4.1을 보라.

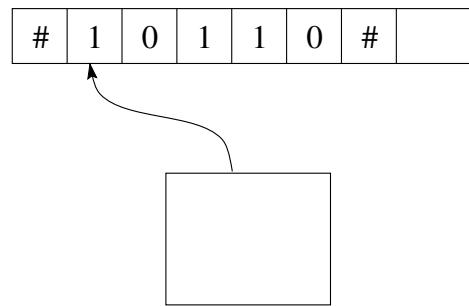


Figure 4.1:

**정의 4.1** 튜링 기계  $M$ 은 여섯 가지 요소  $(Q, \Sigma, \Gamma, \delta, q_0, H)$ 로 구성된다. 여기에서

1.  $Q$ 는 상태들의 유한 집합이고
2.  $\Sigma$ 는 입력 알파벳이고

3.  $\Gamma$ 는 테입 알파벳이고 공백 글자  $\#$ 를 포함한다.
4.  $\delta$ 는  $Q' \times \Gamma$ 에서  $Q \times \Gamma \times \{L, R, S\}$ 으로의 함수이다. 여기에서  $Q' = Q - H$ 이다.
5.  $q_0 \in Q$ 는 초기 상태이고
6.  $H \subseteq Q$ 는 정지상태들의 집합이다.

TM은  $\delta$ 가 전이 함수이므로 결정 오토마타이다.  $\delta$ 의 정의에서  $L$ 은 헤드가 왼쪽으로  $R$ 은 헤드가 오른쪽으로 한 칸 움직이는 것을,  $S$ 는 그 자리에 머무는 것을 의미한다. 유한 오토마타나 내리누름 오토마타는 입력을 다 읽고 나면 정지하지만, 튜링기계는 입력을 다 읽은 후에도 계속 진행할 수 있으므로 정지를 명시하는 정지상태가 필요하다. TM은 맨 왼쪽 칸에서 왼쪽으로 움직이지 않는다고 가정한다.

TM의 상황(configuration)은 현재 상태  $q$ , 테입의 내용과 헤드의 위치에 의해 결정된다. 헤드 왼쪽의 스트링이  $u$ 이고 헤드 상의 글자가  $a$ 이고 헤드 오른쪽의 스트링이  $v$ 이면 ( $v$ 는 오른쪽 끝의 공백 아닌 글자까지 포함한다), 현재 상황은  $(q, uav)$ 로 표시된다 ( $uaqv$ 로 표시하는 것도 가능함). 입력 스트링이  $w$ 일 때, 시작 상황은  $(q_0, \underline{\#}w)$ 이다.

**예제 4.1** 테입 구성이  $\#\underline{aaa}aa$ 에서 헤드가 오른쪽으로 가면  $\#aaaaa$ 가 되고, 이어서 왼쪽으로 가면 원래의 모습이 된다.  $\#\underline{aaa}a$ 에서 오른쪽으로 가면  $\#\underline{aaa}\#$ 가 되고, 다시 오른쪽으로 가면  $\#\underline{aaa}\#\#\underline{}$  된다. 이어서 왼쪽으로 두 번 가면 원래 모습이 된다.

**정의 4.2**  $TM M = (Q, \Sigma, \Gamma, \delta, q_0, H)$ 이 정의하는 언어  $L(M)$ 은 다음과 같다.

$$L(M) = \{w \in \Sigma^* : (q_0, \underline{\#}w) \vdash_M^* (h, *)\}$$

여기에서  $h \in H$ 이고  $*$ 는 임의의 테입 구성을 표시한다. 즉  $L(M)$ 은  $M$ 을 정지하게 만드는 스트링의 집합이다.  $L$ 을 정의하는 TM  $M$  (즉  $L = L(M)$ )이 존재하는 언어  $L$ 을 재귀열거 언어라고 부른다.

**예제 4.2**  $L = \{0^n 1^n : n \geq 1\}$ 을 정의하는 TM  $M$ 을 구하라.

$M$ 은 오른쪽으로 한 칸 가서, 맨 왼쪽의 0을  $x$ 로 바꾸고 ( $q_1$ ), 오른쪽으로 진행하여 맨 왼쪽의 1을 찾아  $y$ 로 바꾸고 ( $q_2$ ), 왼쪽으로 진행하여 맨 오른쪽의  $x$ 를 찾는 ( $q_3$ ) 작업을 반복적으로 수행한다. 또한  $q_0$ 은  $0\mid 1$  없을 때 오른쪽으로 진행하여 1도 없으면 ( $q_4$ ) 정지한다. 예: 000111... $x$ 00 $y$ 11.

구체적으로  $M = (\{q_0, \dots, q_6\}, \{0, 1\}, \{0, 1, x, y, \#\}, \delta, q_0, \{q_6\})$  이고  $\delta$ 는 다음과 같다.

	0	1	$x$	$y$	#
$q_0$					$(q_1, \#, R)$
$q_1$	$(q_2, x, R)$			$(q_4, y, R)$	
$q_2$	$(q_2, 0, R)$	$(q_3, y, L)$		$(q_2, y, R)$	
$q_3$	$(q_3, 0, L)$		$(q_1, x, R)$	$(q_3, y, L)$	
$q_4$				$(q_4, y, R)$	$(q_6, \#, S)$
$q_5$					

여기에서 모든  $q \in Q, a \in \Gamma$ 에 대하여 빈칸은  $\delta(q, a) = (q_5, a, S)$ 를 의미한다. 즉  $q_5$ 는 무한 순환하는 상태이다. 앞으로 무한 순환 상태는 표시하지 않기로 한다.  $M$ 을 그림으로 나타내면 그림 4.2과 같다.

이와 같이 TM은 글자를 다른 글자로 바꾸면서 계산할 수 있다.  $\{a^n b^n c^n : n \geq 1\}$ 을 정의하는 TM도 마찬가지로 구할 수 있다.

**예제 4.3**  $L = \{ww : w \in \{0, 1\}^+\}$ 을 정의하는 TM을 구하라.

먼저 입력 스트링을 앞부분과 뒷부분으로 나누기 위하여 앞부분의 0, 1은 각각  $x, y$ 로 뒷부분의 0, 1은 각각  $x', y'$ 로 바꾼다. 예: 110011... $yyxx'y'y'$ . 그림 4.3을 보라.

이후 앞부분과 뒷부분을 맞추어 보는 것은 앞의 예제와 비슷하다.

또한 TM은 모든 입력에 대해 정지하면서 정지상태에 따라 언어를 정의할 수 있다.

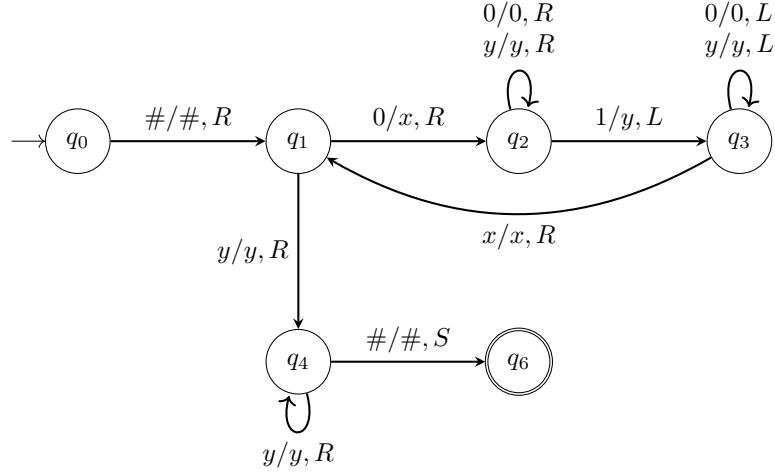


Figure 4.2:

	0	1	x	y	$x'$	$y'$	#
$q_0$							$(q_1, \#, R)$
$q_1$	$(q_2, x, R)$	$(q_2, y, R)$			$(q_5, x', L)$	$(q_5, y', L)$	
$q_2$	$(q_2, 0, R)$	$(q_2, 1, R)$			$(q_3, x', L)$	$(q_3, y', L)$	$(q_3, \#, L)$
$q_3$	$(q_4, x', L)$	$(q_4, y', L)$					
$q_4$	$(q_4, 0, L)$	$(q_4, 1, L)$	$(q_1, x, R)$	$(q_1, y, R)$			
$q_5$			$(q_5, x, L)$	$(q_5, y, L)$			$(q_6, \#, S)$

정의 4.3 언어  $L$ 에 대하여  $TM M = (Q, \Sigma, \Gamma, \delta, q_0, \{y, n\})$  이

- $w \in L$  이면  $(q_0, \underline{\#}w) \vdash_M^* (y, *)$
- $w \notin L$  이면  $(q_0, \underline{\#}w) \vdash_M^* (n, *)$

를 만족하면  $M$ 이  $L$ 을 결정한다고 말한다.  $L$ 을 결정하는  $TM$ 이 존재하면  $L$ 을 재귀 언어라고 부른다.

예제 4.4  $L = \{0^n 1^n : n \geq 1\}$ 에 대하여 예 4.2의  $TM$ 에서  $H = \{q_5, q_6\}$  이고  $q_5 = n, q_6 = y$  이면 이는  $L$ 을 결정하는  $TM$ 이다.

정리 4.1 재귀언어 종류는 재귀열거언어 종류의 부분집합이다.

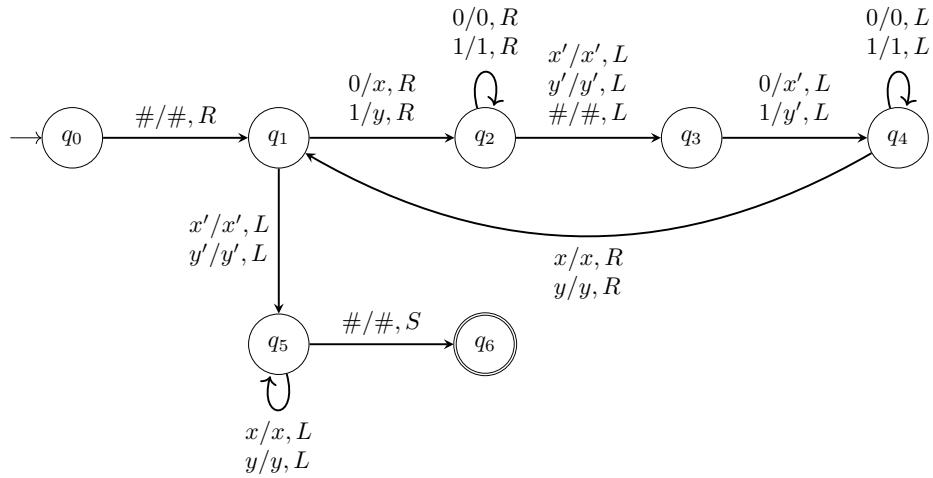


Figure 4.3:

**증명.**  $L$ 을 재귀 언어라고 하면,  $L$ 을 결정하는 TM  $M$ 이 존재한다.  $M$ 에서 상태  $n$ 을 무한 순환 상태로 만들고  $y$ 만을 정지 상태로 하면,  $L$ 을 정의하는 TM을 얻는다. 따라서  $L$ 은 재귀열거 언어이다.  $\square$

TM은 입력에서 출력을 계산하는 도구로 사용될 수도 있다. 이 경우 TM은 어떤 함수를 계산하게 된다. 자연수에서 자연수로의 함수를 다룰 때, 자연수를 단일진법으로 표시한다. 즉 자연수  $n$ 은  $1^n$ 으로 표시된다.

**정의 4.4**  $f$ 를  $\Sigma^*$ 에서  $\Sigma^*$ 로의 함수라고 하자. TM  $M = (Q, \Sigma, \Gamma, \delta, q_0, \{h\})$   $\circ|$

$$(q_0, \underline{\#}w) \vdash_M^* (h, \underline{\#}f(w))$$

을 만족하면  $M$ 이  $f$ 를 계산한다고 말한다. 이 경우  $f$ 를 계산가능 함수라고 부른다.

함수  $f$ 가 입력으로  $k$  개의 자연수를 취할 때 즉  $f(n_1, \dots, n_k)$ 일 때, 입력은  $1^{n_1}0\dots01^{n_k}$ 로 표시된다.

**예제 4.5** 더하기 함수  $f(n, m) = n + m$ 을 계산하는 TM을 구하라.

입력이  $1^n01^m$ 으로 0을 1로 바꾸고 맨 끝의 1을 공백으로 바꾸면 된다. TM의 전이함수는 다음과 같다. 그림 4.4를 보라.

	0	1	#
$q_0$			$(q_1, \#, R)$
$q_1$	$(q_1, 1, R)$	$(q_1, 1, R)$	$(q_2, \#, L)$
$q_2$		$(q_3, \#, L)$	
$q_3$		$(q_3, 1, L)$	$(h, \#, S)$

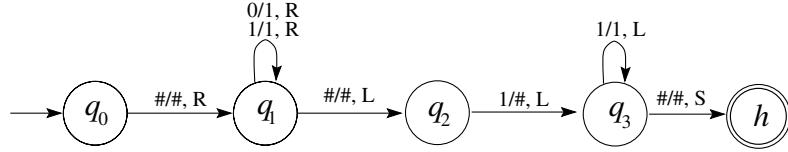


Figure 4.4:

**예제 4.6** 곱하기 함수  $f(n, m) = nm$ 을 계산하는 TM을 구하라.

입력 중  $n$ 이 0이면 테입의 내용을 다 지우고 끝난다 (즉 0을 출력).  $n$ 이 2 이상이면  $m$ 을  $n - 1$  번 복사한 후 테입에 있는 0을 지우면서 1을 왼쪽으로 이동시킨다. 그림 4.5를 보라.

$n = 2, m = 3$  일 때 테입의 변화

```

#1\underline{1}0111#
#\underline{1}00111#
#\underline{a}0011\underline{1}0#
#\underline{a}0011\underline{b}01#
#\underline{a}0\underline{0}1110111#
#\underline{0}0011111\underline{1}###
#\underline{1}11111#

```

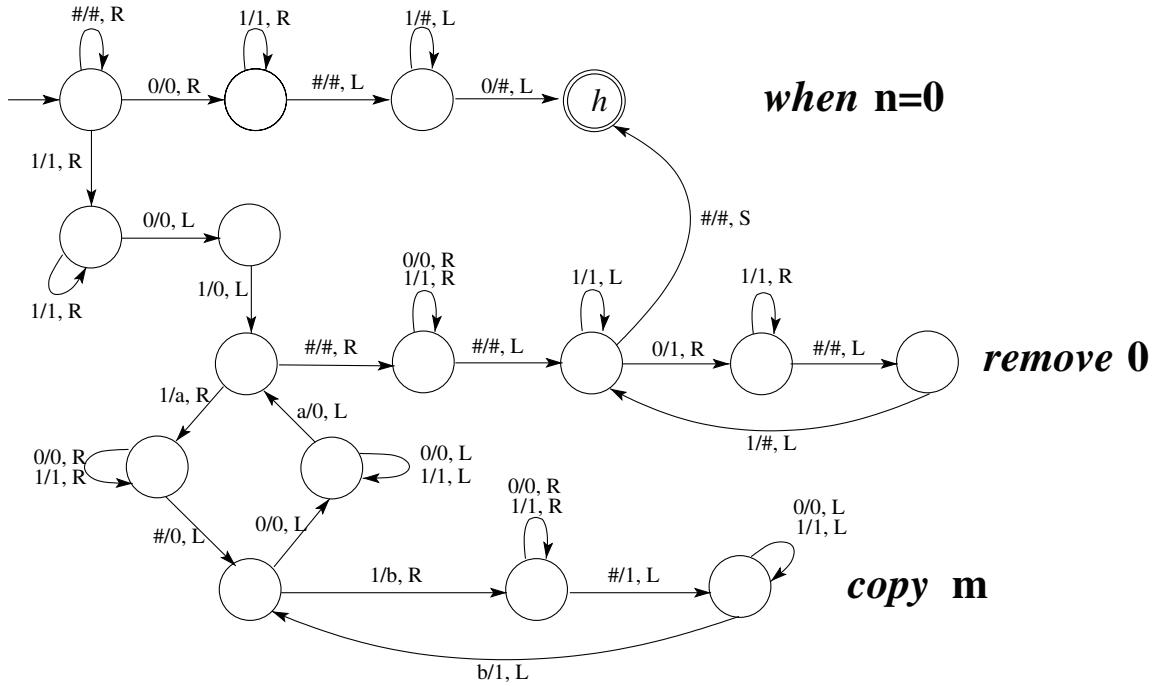


Figure 4.5:

현재 사용되는 컴퓨터가 무엇을 계산할 수 있는가? 컴퓨터의 기계언어(machine instruction)를 보면 load, store 외에 사칙연산과 조건문을 수행하기 위한 jump 등으로 구성되어 있다. 튜링기계도 읽고 쓸 수 있고, 사칙연산을 할 수 있고 제어기를 통해 조건문을 수행할 수 있으므로 현재 컴퓨터가 하는 모든 계산을 다 할 수 있다는 것을 추론할 수 있다.

## 4.2 확장된 튜링기계

앞장에서는 튜링기계가 여러 가지 계산을 할 수 있음을 보았다. 유한 오토마타에서 시작하여 점차로 기능을 더할 때마다 계산능력이 커지는 것을 보았는데, 본 장에서는 튜링기계에 여러 가지 기능을 첨가해도 이 확장된 튜링기계의 계산을 원래 튜링기계가 흉내낼 수

있음을 보이고자 한다. 즉 기능을 더해서 계산능력을 증가시키는 면에서 한계에 도달했다는 것이다.

$k$ 테입 튜링기계는  $k$  개의 테입을 가진다. 따라서 한 번에  $k$  개의 글자를 읽고 움직임을 결정한다. 즉 전이함수  $\delta$ 는  $Q' \times \Gamma^k$ 에서  $Q \times (\Gamma \times \{L, R, S\})^k$ 으로의 함수이다.  $k$ 테입 TM의 상황은 현재 상태와  $k$  개 테입의 모양으로 구성되고,  $(q, u_1 a_1 v_1, \dots, u_k a_k v_k)$ 로 표시된다. 시작할 때, 입력은 첫째 테입에 주어지고 나머지 테입은 비어 있다. 즉 시작 상황은  $(q_0, \#w, \#, \dots, \#)$ 이다. 함수를 계산할 경우,  $k$ 테입 TM의 출력은 첫째 테입에 주어지고 나머지 테입의 내용은 무시된다.

**예제 4.7** 2테입 TM에서 전이함수  $\delta(q, a, b) = (q, (a, R), (a, R))$ 는 첫째 테입에서 읽은 글자를 둘째 테입에 쓰고 헤드를 오른쪽으로 움직이는 것이다.

**예제 4.8**  $k$ 테입 TM은 원래 TM보다 간단하게 계산을 할 수 있다. 예를 들어  $\#w$ 를  $\#w\#w$ 로 바꾸는 계산을 고려하자. 원래 TM은 좌우로 움직이면서  $w$ 를 한 글자씩 옮겨 적는다.  $n = |w|$ 라고 하면 원래 TM은  $O(n^2)$ 의 전이가 필요하다.

2테입 TM은 이 작업을 다음과 같이  $O(n)$ 의 전이에 할 수 있다.

1. 두 헤드를 오른쪽으로 움직이면서 첫째 테입의 내용을 둘째 테입에 적는다. 즉 테입의 상황을  $(\#w, \#)$ 에서  $(\#w\#, \#w\#)$ 로 바꾼다.
2. 둘째 테입의 헤드를 왼쪽으로 옮긴다. 즉  $(\#w\#, \#w\#)$ 로 바꾼다.
3. 두 헤드를 오른쪽으로 움직이면서 둘째 테입의 내용을 첫째 테입에 적는다. 즉  $(\#w\#w\#, \#w\#)$ 로 바꾼다.
4. 첫째 테입의 헤드를 왼쪽으로 옮긴다.

이와 같이 특정한 계산을 하는 편이성(또는 시간)에 있어서는  $k$  테입 TM이 원래 TM보다 나을 수 있지만, 어떤 계산을 할 수 있느냐

없느냐의 관점에서는 두 개가 동일하다. 즉  $k$ 테입 TM이 할 수 있는 계산은 원래 TM도 할 수 있다.

**정리 4.2**  $k$ 테입 TM  $M = (Q, \Sigma, \Gamma, \delta, q_0, H)$ 와 동등한 원래 TM  $M'$ 이 존재한다 ( $L(M) = L(M')$ ).

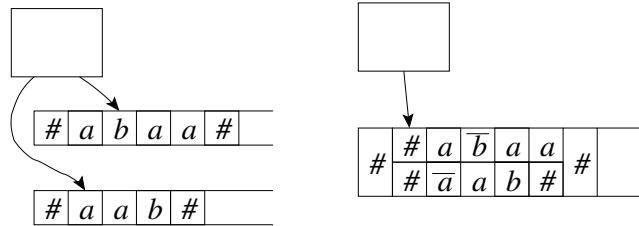


Figure 4.6:

**증명.** 원래 TM  $M'$ 이  $M$ 을 흉내내기 위해서는  $k$  테입의 내용을 보관하고 있어야 한다. 이를 위해  $M'$ 의 테입을  $k$  개의 트랙(track)으로 나눈다. 그럼 4.6을 보라.  $\bar{\Gamma} = \{\bar{a} : a \in \Gamma\}$ 라고 하자.  $\bar{a}$ 는  $M$ 에서 헤드가  $a$ 글자에 있음을 나타낸다. 그럼과 같이  $i$  번째 트랙에는  $i$  번째 테입의 내용과 헤드의 위치를 기록한다.

$k$  개의 트랙을 사용하는 것이 원래 TM의 정의를 벗어나는 것이 아니고, 단지  $M'$ 의 테입 알파벳  $\Gamma'$ 으로  $\Gamma \cup (\Gamma \cup \bar{\Gamma})^k$ 을 사용하는 것 뿐이다. 즉  $M'$ 이  $(\bar{a}, b)$ 를 읽으면 첫째 테입의 이 위치에  $a$ 가 있고 여기에 헤드가 있으며 둘째 테입에는  $b$ 가 있다는 것을 알 수 있다.

입력  $w \in \Sigma^*$ 가 주어지면  $M'$ 은 다음과 같이 작동한다.

1. 입력을 하나씩 오른쪽으로 옮긴 후 테입을  $M$ 의  $k$  테입 형태로 바꾼다.  $w = abaa$ 이고  $M$ 이 2 테입 TM이면,  $M'$ 의 테입을  $\#\#abaa$ 에서  $\#\#abaa$ 로, 그리고  $\#(\#, \#)(a, \#)(b, \#)(a, \#)(a, \#)$ 으로 바꾼다.
2.  $M$ 의 계산을 흉내낸다.  $M$ 의 한 전이를 흉내내기 위해  $M'$ 은 다음의 움직임을 한다.

1. 테입을 지나가면서  $M$ 의  $k$  헤드에 있는 글자를 읽는다.  
 $M$ 의 상태를 기억하고  $k$  헤드에 있는 글자를 읽고 나면,  
 $M$ 의 전이를 알 수 있다.
2. 테입을 지나가면서  $M$ 의 전이를 반영한다. 만약  $M$ 이 어떤 헤드를 오른쪽으로 움직여서  $M'$ 이  $\#$ 이 들어있는 칸으로 옮겨야 하면  $M'$ 은 먼저  $\#$ 를  $(\#, \#)$ 로 바꾼 후  $M$ 의 전이를 반영한다.
3.  $M$ 이 정지하면,  $M'$ 은 첫째 트랙에 있는 내용을 원래 테입의 형태로 바꾸고 정지한다.

□

양쪽 열린 테입 TM은 양방향으로 열린 테입을 한 개 가지고 있다. 초기에 입력이 테입에 주어지고 헤드는 입력 바로 왼쪽의 공백에 놓여 있다고 가정한다. 이 TM은 2테입 TM에 의해 흉내낼 수 있다. 양쪽 열린 테입을 입력의 바로 왼쪽에서 절반으로 접어서 오른쪽의 내용은 첫째 테입에 저장하고 왼쪽의 내용은 둘째 테입에 역으로 저장한 후 흉내내면 된다. 물론 이 2테입 TM은 다시 원래 TM으로 흉내낼 수 있다.

$k$ 헤드 TM은 한 개의 테입에  $k$ 개의 헤드를 가지고 있고  $k$  헤드에 있는  $k$  개의 글자를 읽고 전이를 결정한다.  $k$ 헤드 TM을 이용하면  $\#w$ 를  $\#w\#w$ 로 바꾸는 작업을 더 간단히 할 수 있다.

1. 두 헤드를  $\#w\#$ 의 위치에 놓는다.
2. 두 헤드를 오른쪽으로 움직이면서 첫째 헤드의 내용을 둘째 헤드에 적는다. 즉  $\#w\#w\#$ 로 바꾼다.

$k$ 헤드 TM도  $k$ 테입 TM과 마찬가지로 헤드의 위치를 나타내는  $\bar{\Gamma}$ 를 이용하여 원래 TM이 흉내낼 수 있다.

### 4.3 비결정 튜링기계

비결정 TM은  $(Q, \Sigma, \Gamma, \Delta, q_0, H)$ 로 구성되고, 결정 TM과 다른 점은  $\Delta$ 가  $(Q' \times \Gamma) \times (Q \times \Gamma \times \{L, R, S\})$ 의 부분집합인 전이관계이다.

**정의 4.5** 비결정 TM  $M = (Q, \Sigma, \Gamma, \Delta, q_0, H)$ 이 정의하는 언어  $L(M)$ 은 다음과 같다.

$$L(M) = \{w \in \Sigma^* : (q_0, \underline{\#}w) \vdash_M^* (h, *)\}$$

$M$ 이 비결정이므로 정지 상태로 들어가는 계산 과정이 하나라도 있으면  $w \in L(M)$ 이다.

**정리 4.3** 비결정 TM  $M = (Q, \Sigma, \Gamma, \Delta, q_0, H)$ 와 동등한 원래 TM이 존재한다.

**증명.**  $M$ 은 비결정 TM이므로 한 상황  $C$ 에서 여러 개의 상황  $C_1, \dots, C_i$ 로 전이할 수 있다. 여기에서  $i$ 는 비결정 TM의 정의에 의해  $r = |Q| \times |\Gamma| \times 3$  이하의 값을 갖는다.

$M$ 의 계산과정을, 상황을 정점으로 (시작상황을 루트로) 전이를 간선으로 표시하여 트리  $T$ 로 나타낼 수 있다. 이 트리에서 한 정점의 자식의 개수는 최대  $r$ 이다. 따라서 이 트리의 모든 정점을 정수  $1, \dots, r$ 의 스트링으로 표시할 수 있다. 루트는 1이고 루트의 자식은  $11, \dots, 1r$ 이고  $11$ 의 자식은  $111, \dots, 11r$ 이다. 물론  $11$ 의 자식이  $r$ 보다 적으면  $11r$ 에 해당하는 정점(상황)은 없게 된다. 그럼 4.7을 보라.

이제  $M$ 의 계산과정을 3테입 TM  $M'$ 이 흉내내고자 한다. 즉  $M'$ 은 트리  $T$ 의 정점에 해당하는 상황을 모두 만들어 가면서 그 중에서 정지상황을 만나면 정지하려고 한다.  $T$ 가 유한 트리가 아닐 수 있으므로  $T$ 를 깊이 우선 탐색으로 찾아가면  $T$ 에 정지상황이 있는데도 불구하고 이를 찾지 못할 수 있다. 따라서  $M'$ 은  $T$ 를 너비 우선 탐색으로 찾아간다.

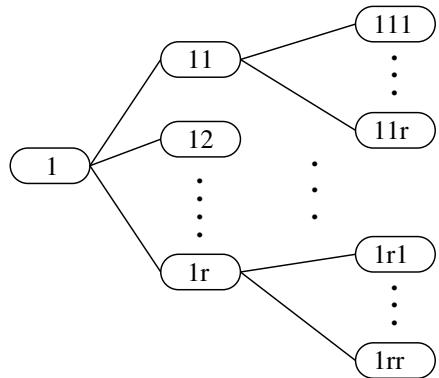


Figure 4.7:

$T$ 를 너비 우선 탐색으로 찾아가기 위해서는 일반적으로 큐(queue)가 필요한데, 다음과 같이 큐를 사용하지 않고 시간을 더 사용하여 해결할 수 있다. 즉  $T$ 의 각 정점을 나타내는  $\{1, \dots, r\}$  상의 스트링을 너비 우선 탐색의 순서로 하나씩 만든다. 둘째 테입에 어떤 스트링  $a_1 \dots a_i$ 가 주어지면 셋째 테입을 이용하여 이 스트링에 해당하는 상황을 만든다. 즉 첫째 테입에 있는  $w$ 를 셋째 테입에 복사한 후 시작 상황에서  $M$ 의  $a_1$ 번째 전이를 적용하여 다음 상황을 만들고 여기에 다시  $a_2$ 번째 전이를 적용하는 식으로 진행한다.  $a_1 \dots a_i$ 에 해당하는 상황이 정지상황이면  $M'$ 도 정지하고, 그렇지 않으면 둘째 테입에 다음 스트링을 만들고 반복한다.  $\square$

$M'$ 은 3테입 TM인데, 첫째 테입에는 입력  $w$ 를 보관하고 첫째 테입의 내용은 변하지 않는다. 둘째 테입에는  $\{1, \dots, r\}$  상의 스트링을 너비 우선 탐색의 순서로 하나씩 만든다. 둘째 테입에 어떤 스트링  $a_1 \dots a_i$ 가 주어지면 셋째 테입을 이용하여 이 스트링에 해당하는 상황을 만든다. 즉 첫째 테입에 있는  $w$ 를 셋째 테입에 복사한 후 시작 상황에서  $M$ 의  $a_1$ 번째 전이를 적용하여 다음 상황을 만들고 여기에 다시  $a_2$ 번째 전이를 적용하는 식으로 진행한다.  $a_1 \dots a_i$ 에 해당하는 상황이 정지상황이면  $M'$ 도 정지하고, 그렇지 않으면 둘째 테입에 다음 스트링을 만들고 반복한다.  $\square$

## 4.4 무제한 문법

무제한 문법  $G = (V, \Sigma, S, P)$ 은 생성규칙  $x \rightarrow y$ 가 임의의 형태  $x \in (V \cup \Sigma)^*V(V \cup \Sigma)^*$ ,  $y \in (V \cup \Sigma)^*$ 를 가진다.

**예제 4.9** 다음 문법은  $L = \{a^n b^n c^n : n \geq 1\}$ 을 생성한다.

$$\begin{aligned} S &\rightarrow abc \mid aDbc \\ Db &\rightarrow bD \\ Dc &\rightarrow Ebcc \\ bE &\rightarrow Eb \\ aE &\rightarrow aa \mid aaD \end{aligned}$$

예를 들면

$$\begin{aligned} S &\Rightarrow aDbc \\ &\Rightarrow abDc \\ &\Rightarrow abEbcc \\ &\Rightarrow aEbbcc \\ &\Rightarrow aabbcc \end{aligned}$$

위의 예가 보여주듯이 무제한 문법은 문맥무관이 아닌 언어도 만들 수 있다. 무제한 문법이 생성하는 언어의 종류는 재귀열거 언어 종류임을 보이자.

**정리 4.4** 무제한 문법  $G = (V, \Sigma, S, P)$ 에 대하여  $L(G)$ 를 정의하는 TM  $M$ 이 존재한다.

**증명.**  $M$ 은 비결정 2테입 TM이다.  $M$ 의 첫째 테입에는 입력  $w$ 가 주어진다. 둘째 테입에는  $G$ 의 문장형태가 만들어진다. 처음에 둘째 테입에  $S$ 를 쓰고 다음을 반복한다.

1. 비결정적으로 둘째 테입의 모든 위치  $i$ 를 선택한다. (즉 왼쪽 끝에서 시작하여 오른쪽으로 진행하면서 현재 위치를 선택하는 작업을 반복한다.)
2. 비결정적으로  $G$ 의 모든 생성규칙  $x \rightarrow y$ 을 선택한다.
3. 둘째 테입의  $i, \dots, i + |x| - 1$  위치의 글자가  $x$ 이면,  $x$ 를  $y$ 로 바꾼다. 이 과정에서  $|x| \neq |y|$ 이면  $x$ 의 오른쪽에 있는 글자를 왼쪽이나 오른쪽으로 이동시켜야 된다.
4. 둘째 테입에 있는 문장형태를  $w$ 와 비교하여 같으면 정지한다.

$G$ 가  $w$ 를 생성하면  $M$ 은 언젠가 정지하고 그렇지 않으면  $M$ 은 정지하지 않는다. 따라서  $L(M) = L(G)$ 이다.  $\square$

**정리 4.5**  $TM M = (Q, \Sigma, \Gamma, \delta, q_0, H)$ 가 정의하는 언어  $L(M)$ 을 생성하는 무제한 문법  $G$ 가 존재한다.

**증명.**  $M$ 에서  $H = \{h\}$ 이고,  $M$ 은 정지할 경우 항상 상황  $(h, \#)$ 에서 정지한다고 가정하자. 임의의 TM을 위의 조건을 만족하는 TM으로 바꿀 수 있다.

$M$ 의 전이를 반대로 흉내내는 문법  $G$ 를 만들려고 한다. 즉  $M$ 이 시작상황  $(q_0, \#w)$ 에서  $(h, \#)$ 로 전이할 때,  $G$ 는  $[\#h]$ 에서  $[\#q_0w]$ 를 유도하려고 한다. 이를 위해  $M$ 의 각 전이에 대하여  $G$ 는 다음의 생성규칙을 갖는다.

- $\delta(q, a) = (p, b, S)$ :  $G$ 는  $bp \rightarrow aq$ 를 갖는다. 그림 4.8을 보라.
- $\delta(q, a) = (p, b, R)$ :  $G$ 는 모든  $c \in \Sigma \cup \{\#\}$ 에 대하여  $bcp \rightarrow aqc$ 를 갖는다. 또한  $b\#p] \rightarrow aq]$ 를 갖는다. 그림 4.9, 4.10을 보라.
- $\delta(q, a) = (p, b, L)$ :  $b \neq \#$ 이면  $G$ 는  $pb \rightarrow aq$ 를 갖는다.  $b = \#$ 이면 모든  $c \in \Sigma$ 에 대하여  $p\#c \rightarrow aqc$ 를 갖고 또한  $p] \rightarrow aq]$ 를 갖는다. 그림 4.11, 4.12, 4.13을 보라.

처음과 마지막을 위하여  $G$ 는

$$S \rightarrow [\#h]$$

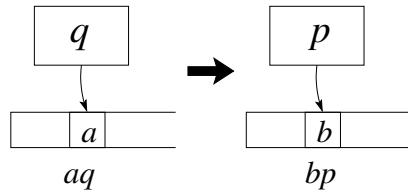


Figure 4.8:

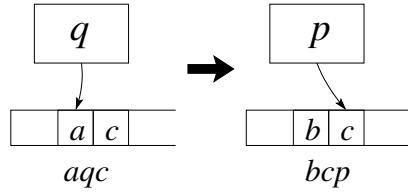


Figure 4.9:

$$\begin{aligned} [\#q_0 &\rightarrow \epsilon \\ ] &\rightarrow \epsilon \end{aligned}$$

을 갖는다.

그리면  $M \circ [ (q_0, \underline{\#}w) \vdash^* (h, \underline{\#}) ]$ 로 전이할 때,  $G$ 는

$$S \Rightarrow [\#h] \xrightarrow{*} [\#q_0 w] \xrightarrow{*} w$$

를 유도하게 된다.  $\square$

## 4.5 Church-Turing의 명제

먼저 문제를 언어로 바꾸어 생각할 수 있음을 보이자. 문제에는 크게 결정 문제(decision problem)와 최적화 문제(optimization problem)의 두 가지 유형이 있다. 결정 문제란 답이 예 또는 아니오인 문제이고, 최적화 문제란 어떤 조건을 만족하는 최소값 또는 최대값을 구하는 문제이다.

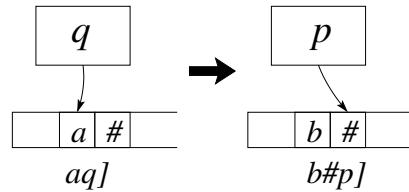


Figure 4.10:

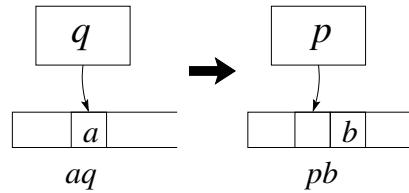


Figure 4.11:

**정의 4.6** 그래프에서 모든 정점을 포함하는 단순 사이클을 해밀톤 사이클이라고 한다. 해밀톤 사이클 문제는 그래프  $G$ 가 주어졌을 때  $G$ 가 해밀톤 사이클을 갖고 있느냐는 문제이다.

**예제 4.10** 그림 4.14에서 왼쪽 그래프는 해밀톤 사이클을 갖고 있지 않고, 오른쪽 그래프는 해밀톤 사이클을 갖고 있다.

**정의 4.7** 외판원(*traveling salesman*) 문제는 완전 그래프  $G$ 와 모든 간선  $e$ 의 거리  $d(e)$ 가 주어졌을 때, 거리가 최소인 해밀톤 사이클을 찾으라는 문제이다.

해밀톤 사이클 문제는 결정 문제이고 외판원 문제는 최적화 문제이다.

결정 문제는 다음과 같이 언어로 바꿀 수 있다. 즉 결정 문제의 입력  $I$ 를 스트링  $w$ 로 바꾸고,  $I$ 의 답이 예이면  $w \in L$ 이고 답이 아니오이면  $w \notin L$ 이다.

최적화 문제는 결정 문제를 이용하여 해결할 수 있다. 논의를 단순화하기 위하여, 최적화 문제에서 최적값만을 구한다고 가정한다.

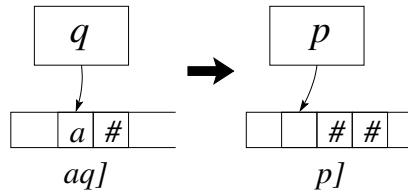


Figure 4.12:

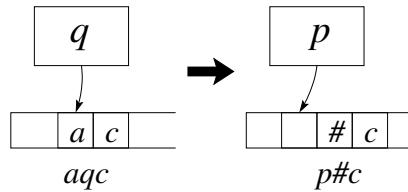


Figure 4.13:

외판원 문제에 대한 결정 문제는 다음과 같다. 완전 그래프  $G$ 와 간선의 거리  $d(e)$ 와 양의 정수  $A$ 가 주어졌을 때, 거리가  $A$  이하인 해밀톤 사이클이 있는가. 결정 문제를 풀 수 있으면 이진 탐색(binary search)에 의해 최적화 문제를 풀 수 있다. 따라서 모든 문제를 언어로 바꾸어 생각할 수 있다.

**정의 4.8 (Church-Turing 명제)** 재귀 언어를 컴퓨터로 풀 수 있는 문제라고 하자.

재귀 언어를 결정하는 TM을 알고리즘이라고 부른다.

## 4.6 Chomsky 체계

정규언어와 문맥무관 언어 외에 문맥민감 언어가 있다. 문맥무관 언어의 표준형에서와 같이 다음의 언어는  $\epsilon$ 을 포함하지 않는다고 가정 한다.

**정의 4.9** 문법  $G = (V, \Sigma, S, P)$ 의 모든 생성규칙  $x \rightarrow y$ 가  $|x| \leq |y|$ 를 만족하면  $G$ 는 문맥민감 문법(context-sensitive grammar)이라고 부른다. (여기에서  $x \in (V \cup \Sigma)^* V (V \cup \Sigma)^*$ 이고  $y \in (V \cup \Sigma)^*$ 이다.)

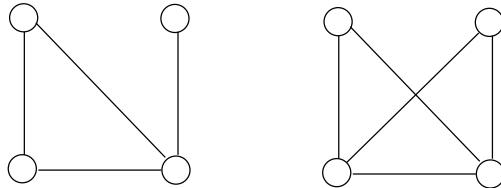


Figure 4.14:

**정의 4.10** 문맥민감 문법이 생성하는 언어를 문맥민감 언어(*context-sensitive language*)라고 부른다.

따라서 문맥민감 문법의 유도과정에서 문장형태의 길이는 단조증가한다. 문맥민감 문법이라고 불리는 이유는 모든 생성규칙을 다음의 표준형으로 바꿀 수 있기 때문이다.

$$uAv \rightarrow uwv$$

여기에서  $w \neq \epsilon$ 이다. 이 생성규칙은  $u$ 와  $v$ 의 문맥에서  $A$ 가  $w$ 로 바뀔 수 있음을 나타낸다. Chomsky 표준형이 문맥민감 문법의 형태이므로 문맥무관 언어의 종류는 문맥민감 언어 종류의 부분집합이다.

**예제 4.11**  $L = \{a^n b^n c^n : n \geq 1\}$ 은 예제 4.9의 문법에 의해 생성되므로 문맥민감 언어이다.

위 예제의 의해 문맥무관 언어의 종류는 문맥민감 언어 종류의 진부분집합이다.

**정의 4.11** 선형 오토마타는 입력  $w$ 에 대하여 테입의  $\#w\#$  부분만을 사용하는 비결정 튜링기계이다.

**정리 4.6** 선형 오토마타가 나타내는 언어의 종류는 문맥민감 언어 종류이다.

**증명.** 무제한 문법과 TM의 관계에 대한 증명과 비슷하다.

(문맥민감 문법  $G \rightarrow$  선형 오토마타  $M$ ) 선형 오토마타  $M$ 은 테입을 2개의 트랙으로 나누고 첫째 트랙에 입력  $w$ 를 보관하고 둘째 트랙에  $G$ 의 문장형태  $u$ 를 만든다. 문맥민감 문법의 문장형태의 길이는 단조증가하므로  $u \xrightarrow{*} w$ 일 경우  $|u| \leq |w|$ 이므로  $M$ 은 테입의  $\#w\#$  부분만을 사용하여  $w \in L(G)$ 인 경우  $w$ 를 받아들일 수 있다.

□

Chomsky는 언어의 종류를 다음 네 가지로 분류하였다. 이를 Chomsky 체계라고 부른다.

유형	언어 종류
0	재귀열거 언어
1	문맥민감 언어
2	문맥무관 언어
3	정규 언어

Figure 4.15:

이 외에 재귀 언어가 유형 0과 1 사이에 위치한다. 다섯 가지 언어 종류는 서로 진부분집합 관계를 가진다. 그림 4.16을 보라.

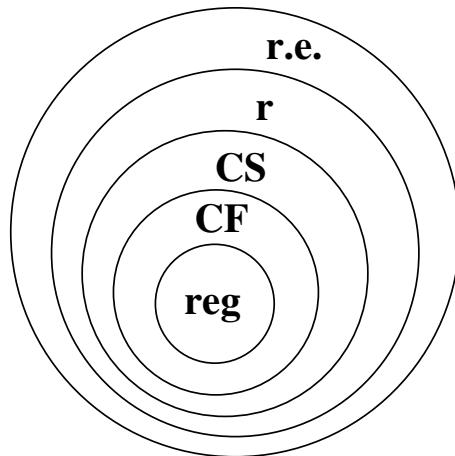


Figure 4.16:

## 4.7 연습 문제

### 1. 튜링기계

- 1.1. 단항 표기법(unary notation)으로 표현된 양의 정수  $x, y$ 에 대하여 다음 함수를 계산하는 TM을 구하라.

$$f(x, y) = \begin{cases} x - y & x \geq y \\ 0 & x < y \end{cases}$$

- 1.2. 언어  $\{a^n b^m : m = n^2, n \geq 1\}$ 를 결정하는 TM을 구하라.
- 1.3. 단항 표기법으로 표현된 정수  $n \geq 1$ 에 대해  $f(n) = n!$ 을 계산하는 TM을 구하라.
- 1.4. 단항 표기법으로 표현된 정수  $n \geq 0$ 에 대해  $f(n) = 2^n$ 을 계산하는 TM을 구하라.
- 1.5. 언어  $\{w \in \{a, b\}^* \mid w\text{는 적어도 } 1\text{개의 } a\text{를 포함한다}\}$ 을 결정하는 TM  $M$ 을 구하라. TM의 입력과 출력을 구체적으로 기술하라.
- 1.6.  $f(n, m) = n + m + 2$ 을 계산하는 TM을 구하라. TM의 입력과 출력을 구체적으로 기술하라.
- 1.7. 음이 아닌 정수  $x, y, z$ 에 대하여  $F(x, y, z) = x + y + z$ 를 계산하는 TM을 구하라.
- 1.8. 언어  $L = \{ww : w \in \{a, b\}^+\}$ 을 정의하는 TM을 구하라.
- 1.9.  $n \circ |$  단항 표기법으로 주어질 때  $\lceil \log_2 n \rceil$ 을 계산하는 TM을 구하라.
- 1.10.  $w \in \{0, 1\}^+$ 에 대하여 함수  $f(w) = w^R$ 를 계산하는 TM을 구하라.
- 1.11. 언어  $L = \{a^n b^n a^n b^n : n \geq 1\}$ 을 정의하는 TM을 구하라.

- 1.12. 언어  $L = \{w \in \{a, b\}^* : N_a(w) = N_b(w)\}$ 을 정의하는 TM을 구하라.  $N_a(w)$ 와  $N_b(w)$ 는  $w$ 에서 각각  $a$ 와  $b$ 의 수를 나타낸다.

2. 확장된 투링기계

3. 비결정 투링기계

- 3.1. 결정 TM이 비결정 TM을 어떻게 흉내내는지 설명하라.  
(같은 스트링 집합에 대하여 두 개의 TM이 정지해야함에 주의하라.)

- 3.2. 다음 언어를 정의하는 비결정 TM을 구하라.

$$L = \{a^n : n \text{은 소수가 아니다}\}$$

- 3.3. 다음 언어를 정의하는 효율적인 결정, 비결정 TM을 구하라.

$$L = \{ww : w \in \{0, 1\}^+\}$$

4. 무제한 문법

5. Church-Turing의 명제

6. Chomsky 체계

- 6.1. 재귀 언어 종류가 다음 연산에 닫혀있는지 증명하라.

- (a) 합집합
- (b) 접합

- 6.2.  $L(M)$ 을 TM  $M$ 이 정의하는 언어라고 하자. 다음 언어가 재귀인지 아닌지 증명하라.

- (a)  $\{\langle M \rangle \mid aab \in L(M)\}$
- (b)  $\{\langle M \rangle \mid L(M) \text{은 재귀 열거}\}$

6.3. 다음 언어에 대한 선형 오토마타를 구하라.

$$L = \{a^n : n = k^2, k \geq 1\}$$

- 6.4. 정규 언어 종류, 문맥민감 언어 종류, 문맥무관 언어 종류, 재귀 열거 언어 종류, 재귀 언어 종류, 모든 언어 종류의 관계를 벤 다이어그램으로 보여라. 언어 종류들의 진부분 집합 관계를 나타내는 예제를 적어도 3가지 보여라.
- 6.5. 문맥민감 언어 종류가 역행(reversal)에 닫혀 있음을 증명 하라.
- 6.6. 문맥무관 언어 종류가 교집합에 닫혀 있지 않음을 증명 하라.
- 6.7. 언어  $L = \{a^{n!} : n \geq 0\}$ 을 정의하는 선형 오토마타를 구하라.

# Chapter 5

## 계산불가

### 5.1 무한집합의 크기

무한집합은 크기에 의해 두 가지로 나눌 수 있다. 자연수의 집합  $N = \{0, 1, 2, \dots\}$ 과 1대1 대응관계를 가지는 집합을 셀 수 있는 무한집합이라고 부른다. 그 외의 무한집합을 셀 수 없는 집합이라고 부른다. 집합  $A$ 가 셀 수 있는 무한집합이면,  $N$ 과 1대1 대응관계를 가지므로 이 대응관계에 의해  $A$ 의 원소를  $a_0, a_1, a_2, \dots$  차례로 나열할 수 있다.

**예제 5.1**  $A, B$ 가 셀 수 있는 무한집합이면,  $A \cup B$ 도 셀 수 있는 무한집합이다.

단순하게 하기 위하여  $A \cap B = \emptyset$ 이라고 가정하자.  $A, B$ 가 셀 수 있는 집합이므로 그 원소를 차례로 나열할 수 있다. 즉  $A = \{a_0, a_1, \dots\}$ ,  $B = \{b_0, b_1, \dots\}$ . 이제  $A \cup B$ 의 원소를 차례로 나열하는 방법을 보이면 된다.  $A \cup B = \{a_0, b_0, a_1, b_1, \dots\}$ 과 같이  $A$ 와  $B$ 의 원소를 번갈아 가면서 나열하면 된다.

**예제 5.2**  $N \times N$ 은 셀 수 있는 무한집합이다.

$N \times N$ 의 모든 원소를 유한 번 이내에 셀 수 있는 순서를 찾아야 한다.  $(0, 0), (0, 1), (0, 2), \dots$  순서로 나열하면 유한 번 이내에  $(1, 0)$ 을 나열할 수 없으므로 안 된다.  $(i, j)$ 를  $i + j$ 가 증가하는 순서(즉  $(0, 0), (0, 1), (1, 0), (0, 2), (1, 1), (2, 0), (0, 3), \dots$ )로 나열하면 된다.

정리 5.1  $2^N$ 은 셀 수 없는 집합이다.

**증명.**  $2^N$ 이 셀 수 있는 무한집합이라고 가정하자. 그러면  $2^N$ 의 원소 ( $N$ 의 부분집합)를 나열하는 방법이 존재한다.

$$2^N = \{A_0, A_1, A_2, \dots\}$$

각 집합  $A_i$ 를 표로 나타내면 다음과 같다. 이 표의 행은 집합  $A_i$ 를, 열은 원소  $a$ 를 나타내고  $a \in A_i$ 이면 표의 값이 1이고,  $a \notin A_i$ 이면 0이다.

	0	1	2	3	...
$A_0$	0	0	0	1	
$A_1$	0	1	1	0	
$A_2$	1	0	1	0	
$A_3$	1	0	0	0	
...					

이제 집합  $D = \{i \in N : i \notin A_i\}$ 를 고려하자. 즉  $D$ 는 위의 표에서 주대각선이 0인 원소들의 집합이다.  $D$ 는 자연수의 집합이므로  $2^N$ 의 원소이어야 한다. 그러나  $D$ 는 어느 집합  $A_i$ 와도 같지 않다 (자연수  $i$ 가  $A_i$ 에 속하면  $D$ 에 속하지 않고,  $A_i$ 에 속하지 않으면  $D$ 에 속하므로). 즉 모순에 도달하므로,  $2^N$ 는 셀 수 없는 집합이다.  $\square$

알파벳  $\Sigma$  상의 모든 스트링의 집합  $\Sigma^*$ 는 셀 수 있는 무한집합이다.  $\Sigma = \{0, 1\}$ 인 경우, 모든 스트링을 길이와 사전식 순서(lexicographic order)에 의해  $\epsilon, 0, 1, 00, 01, 10, 11, \dots$ 의 순서로 나열할 수 있다.

언어는  $\Sigma^*$ 의 부분집합이므로, 언어의 전체 집합은  $2^{\Sigma^*}$ 이다. 즉 언어의 전체 집합은 셀 수 없는 집합이다. 다음 장에서 재귀열거 언어 종류는 셀 수 있는 무한집합임을 보인다. 따라서  $2^{\Sigma^*}$ 에는 재귀열거가 아닌 언어(즉 튜링기계로 표시할 수 없는 언어)가 무한히 많이 존재한다. 이후에 재귀열거가 아닌 언어의 예를 보게 될 것이다.

## 5.2 범용 튜링기계

앞으로 언어의 알파벳  $\Sigma$ 를  $\{0, 1\}$ 로, TM의 테입 알파벳  $\Gamma$ 를  $\{0, 1, \#\}$ 로 제한한다. 제한된 문제(언어)가 계산불가임을 보이면, 더 일반적인 문제는 당연히 계산불가가 된다.

먼저 튜링기계를 스트링으로 바꾸는 방법을 보자. TM  $M = (Q, \{0, 1\}, \{0, 1, \#\}, \delta, q_1, H)$ 를 다음과 같이  $\{0, 1\}$  상의 스트링으로 부호화한다.

1.  $Q = \{q_1, \dots, q_n\}$ 일 때,  $q_i$ 는  $0^i$ 으로 표시한다.
2.  $\Sigma = \{0, 1\}$ 에서 0과 1은 각각  $0$ 과  $00$ 으로 표시한다.
3.  $\Gamma = \{0, 1, \#\}$ 에서  $0, 1, \#$ 는 각각  $0, 00, 000$ 으로 표시한다.
4. 방향을 나타내는  $L, R, S$ 는 각각  $0, 00, 000$ 으로 표시한다.
5. 전이  $\delta(q, a) = (p, b, d)$ 는  $q, a, p, b, d$ 의 부호들 사이에 1을 두고 차례로 나열한다. 즉  $\delta(q_i, 0) = (q_j, 1, L)$ 은  $0^i 1 010^j 1 0010$ 으로 표시된다.
6. TM  $M$ 의 부호는 전이의 부호들 사이에 11을 두고 사전 순서로 나열하고, 앞뒤에 111을 붙인 것이다.
7. 초기 상태는 항상  $q_1$ 이다.
8. 정지 상태는 전이의 첫째 원소에 나타나지 않는 상태이다. 언어를 결정할 경우  $H = \{y, n\}$ 인데, 이때  $y$ 는 두 정지상태 중 사전 순서가 작은 것이고  $n$ 은 큰 것이다.

$M$ 의 부호를  $\langle M \rangle$ 으로 표시한다.

TM  $M$ 의  $\Sigma$ 와  $\Gamma$ 가 일반적인 경우에도 이들이 유한집합이므로  $M$ 을  $\{0, 1\}$  상의 스트링으로 표시할 수 있다. 따라서 모든 TM의 집합(즉 재귀열거 언어의 집합)은  $\{0, 1\}^*$ 의 부분집합이므로 셀 수 있는 무한집합이다.

**예제 5.3** TM  $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, \#\}, \delta, q_1, \{q_3\})$  이고

$$\begin{aligned}\delta(q_1, \#) &= (q_2, \#, R) \\ \delta(q_2, 0) &= (q_2, 1, R)\end{aligned}$$

$$\delta(q_2, 1) = (q_2, 0, R)$$

$$\delta(q_2, \#) = (q_3, \#, S)$$

이면  $\langle M \rangle$ 은

11101000100100010011001010010010010011

001001001010011001000100010001000111

이다.

튜링기계  $M$ 의 입력 스트링  $w$ 에 대해서도 0과 1을 각각 0과 00으로 표시하고 글자 사이에 1을 넣어 부호  $\langle w \rangle$ 를 만든다. 즉  $\langle 1001 \rangle = 001010100$ .  $\langle M \rangle$ 과  $\langle w \rangle$ 를 접합한 것을  $\langle M, w \rangle$ 로 표시한다.

지금까지 고려한 튜링기계는  $\{0^n 1^n : n \geq 1\}$ 을 결정하는 TM, 덧셈하는 TM, 곱셈하는 TM 등 특정한 기능을 수행하는 TM이었다. 그런데 지금 우리가 사용하는 컴퓨터는 여러 가지 기능을 수행할 수 있는 범용컴퓨터(general-purpose computer)이다. 범용 컴퓨터를 가지고 특정한 기능을 수행하고자 할 때는 그 기능을 수행하는 프로그램을 작성하여 이 프로그램을 범용컴퓨터가 수행하면 된다. 즉 지금까지 고려한 TM은 프로그램에 해당한다. 범용컴퓨터에 해당하는 것이 범용튜링기계이다.

범용튜링기계  $M_u$ 는  $\langle M, w \rangle$ 를 입력으로 받아서  $M$ 을 입력  $w$ 를 가지고 돌린 후 그 결과를 출력하는 TM이다 (즉  $M$ 이 정지하면  $M_u$ 도 정지한다).  $M_u$ 는 3테입 TM이다.  $M_u$ 는 다음과 같이 첫째 테입에  $\langle M \rangle$ 을, 둘째 테입에  $M$ 의 테입 내용을, 셋째 테입에  $M$ 의 상태를 저장하고  $M$ 을  $w$ 에 대하여 작동시킨다.

- 첫째 테입에 주어진 입력  $\langle M, w \rangle$  중에서  $\langle M \rangle$ 에 해당하는 부분이 제대로 된 TM의 부호인지 점검한다 (syntax error 점검). 즉 111로 시작하고 0에 의해 구분된 1이 4개 나오고 11이 나오는 것이 반복된 후에 111이 나오는지, 전이  $0^a 10^b 10^c 10^d 10^e$

에서  $1 \leq b, d, e \leq 3$ 인지, 전이 중에서 같은  $0^a 1 0^b 1$ 로 시작되는 것이 없는지 점검한다.

2. 첫째 테입에  $\langle M \rangle$ 을 남기고  $\langle w \rangle$ 를 둘째 테입에 옮겨 적는다. 셋째 테입에  $M$ 의 시작상태  $q_1$ 에 해당하는 0을 적는다. 모든 테입의 헤드는 맨 왼쪽에 위치한다.
3. 셋째 테입의 내용이  $0^a$ 이고 둘째 테입의 현재 글자가  $0^b$ 이면 첫째 테입에서  $1 1 0^a 1 0^b 1$ 로 시작되는 전이가 있는지 찾는다. 그런 전이가 없으면 ( $0^a$ 가  $M$ 의 정지상태이다)  $M_u$ 는 정지한다. 그런 전이가 있으면 이를 셋째 테입과 둘째 테입에 반영한다.

### 5.3 정지문제

$L_1$ 이 재귀열거 언어일 때  $L_1$ 을 정의하는 TM  $M_1$ 과  $L_2$ 가 재귀 언어일 때  $L_2$ 를 결정하는 TM  $M_2$ 를 다음 그림과 같이 표시한다.

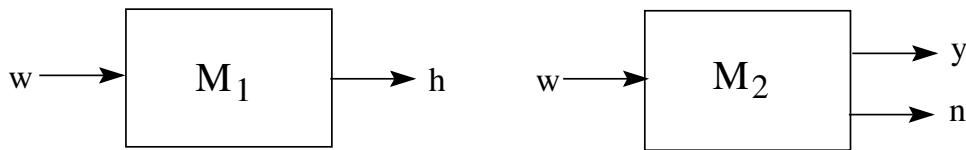


Figure 5.1:

**정리 5.2** 재귀 언어  $L$ 의 여집합은 재귀이다.

**증명.**  $L$ 을 결정하는 TM에서  $y$ 와  $n$ 을 서로 바꾸어 주면  $\bar{L}$ 을 결정하는 TM이 된다.  $\square$

**정리 5.3** 언어  $L$ 과 그 여집합  $\bar{L}$ 이 둘 다 재귀열거이면  $L$ 은 재귀이다.

**증명.**  $L$ 과  $\bar{L}$ 이 재귀열거이므로 이를 정의하는 TM  $M_1$ 과  $M_2$ 가 존재한다. 입력  $w$ 에 대하여  $M_1$ 과  $M_2$ 를 동시에 돌리면서  $M_1$ 의 정지 상태를  $y$ 로,  $M_2$ 의 정지상태를  $n$ 으로 갖는 TM이  $L$ 을 결정하는 TM이다.  $\square$

정리 5.2과 5.3에 의해서 한 쌍의 언어  $L$ 과  $\bar{L}$ 는 다음의 경우들만을 가질 수 있다.

- (둘 중의 하나라도 재귀인 경우)  $L$ 과  $\bar{L}$ 가 둘 다 재귀이다. (나머지 경우 중, 둘 다 재귀가 아닌 재귀열거일 수 없으므로 다음의 두 가지가 남는다.)
- $L$ 과  $\bar{L}$ 가 둘 다 재귀열거가 아니다.
- $L$ 과  $\bar{L}$  둘 중의 하나는 재귀가 아닌 재귀열거이고, 다른 하나는 재귀열거가 아니다.

정지문제란 입력으로 프로그램  $P$ 와 입력  $w$ 를 주고  $P$ 가  $w$ 에 대하여 정지할지 안 할지를 결정하는 문제이다. 정지문제를 언어로 바꾸면

$$L_u = \{\langle M, w \rangle : w \in L(M)\}$$

이 된다. 정지문제가 계산불가임을 보이기 위하여  $L_u$ 가 재귀가 아님을 보이고자 한다.

다음 한 쌍의 언어를 정의한다.

$$L_d = \{\langle M \rangle : \langle M \rangle \in L(M)\}$$

즉,  $w \in \{0, 1\}^*$ 가 어떤 튜링기계  $M$ 의 부호이고  $M$ 이  $\langle M \rangle$ 에 대하여 정지하는 경우에만  $w$ 는  $L_d$ 의 원소이다.

$\bar{L}_d$ 는  $L_d$ 의 여집합이다. 즉,  $w \in \{0, 1\}^*$ 가 튜링기계의 부호가 아니거나  $w$ 가 어떤 튜링기계  $M$ 의 부호인데  $M$ 이  $\langle M \rangle$ 에 대하여 정지하지 않는 경우에만  $w$ 는  $\bar{L}_d$ 의 원소이다.

정리 5.4  $\bar{L}_d$ 는 재귀열거가 아니다.

**증명.**  $\{0, 1\}^*$ 의 원소를 길이와 사전식 순서로 행과 열에 나열한 표를 생각하자. 이 표의 행은 튜링기계의 부호  $\langle M \rangle$ 이고, 열은 튜링기계에 대한 입력 단어  $w$ 이고,  $w \in L(M)$ 이면 표의 값은 1이고, 아니면 표의 값은 0이다.  $L_d$ 는 주대각선이 1인 단어들의 집합이고,  $\bar{L}_d$ 는 주대각선이 0인 단어들의 집합이다.

$\bar{L}_d$ 가 재귀열거이기 위해서는  $\bar{L}_d$ 를 정의하는 TM(즉  $L(M) = \bar{L}_d$ 인 TM  $M$ )이 존재하여야 한다. 모든 TM은 위의 표의 행에 나열되어 있는데 이 중 어떤 TM  $M$ 도  $L(M) \neq \bar{L}_d$ 이다. 왜냐하면  $\langle M \rangle \in L(M)$ 인 경우에만  $\langle M \rangle \notin \bar{L}_d$ 이기 때문이다.  $\bar{L}_d$ 를 정의하는 튜링기계가 없으므로  $\bar{L}_d$ 는 재귀열거가 아니다.  $\square$

$\bar{L}_d$ 는 재귀열거가 아닌 언어의 첫번째 예이다. 위의 세 가지 경우에 의해서  $L_d$ 는 재귀가 아닌 재귀열거이거나 재귀열거가 아니어야 하는데,  $L_u$ 를 고려함으로써  $L_d$ 가 재귀가 아닌 재귀열거임을 밝히고자 한다.

**정리 5.5**  $L_u$ 는 재귀열거이다.

**증명.** 범용튜링기계  $M_u$ 는  $\langle M, w \rangle$ 의 입력에 대하여  $w$ 를 가지고  $M$ 을 돌려보아서  $M$ 이 정지하면  $M_u$ 도 정지한다. 즉  $L(M_u) = L_u$ 이다.  $L_u$ 를 정의하는 튜링기계가 존재하므로  $L_u$ 는 재귀열거이다.  $\square$

**정리 5.6**  $L_d$ 는 재귀열거이다.

**증명.** 입력  $w \in \{0, 1\}^*$ 를  $w\langle w \rangle$ 로 바꾸어서 범용튜링기계  $M_u$ 에 보낸 후  $M_u$ 의 결과를 출력하는 튜링기계를  $M_d$ 라고 하자. 그림 5.2를 보라. 그러면  $w$ 가 어떤 튜링기계  $M$ 의 부호이고  $M$ 이  $\langle M \rangle$ 에 대하여 정지하는 경우에만  $M_d$ 는 정지한다. 즉  $L(M_d) = L_d$ 이므로  $L_d$ 는 재귀열거이다.  $\square$

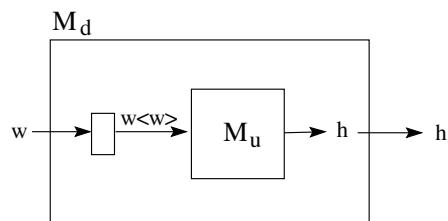


Figure 5.2:

정리 5.7  $L_u$ 와  $L_d$ 는 재귀가 아니다.

**증명.**  $L_u$ 가 재귀라고 가정하자. 그러면  $L_u$ 를 결정하는 튜링기계  $M_0$ 이 존재한다. 즉  $M_0$ 은 입력  $\langle M, w \rangle$ 에 대하여  $w \in L(M)$ 이면  $y$ 에서 정지하고,  $w \notin L(M)$ 이면  $n$ 에서 정지한다.

이제  $M_0$ 을 이용하여  $L_d$ 를 결정하는 튜링기계  $M_1$ 을 만들고자 한다.  $M_1$ 은 입력  $w \in \{0, 1\}^*$ 에 대하여  $w\langle w \rangle$ 를 만들어서  $M_0$ 에게 보낸 후,  $M_0$ 의 결과를 출력한다. 그림 5.3을 보라. 그러면  $w$ 가 어떤 튜링기계  $M$ 의 부호이고  $\langle M \rangle \in L(M)$ 이면  $M_1$ 은  $y$ 에서 정지하고, 그렇지 않으면  $M_1$ 은  $n$ 에서 정지한다. 즉  $M_1$ 은  $L_d$ 를 결정하는 튜링기계이다.

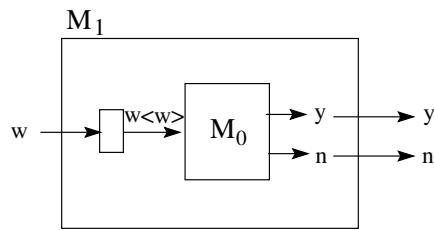


Figure 5.3:

따라서  $L_u$ 가 재귀이면  $L_d$ 도 재귀이고, 정리 5.2에 의해서  $L_d$ 가 재귀이면  $\bar{L}_d$ 도 재귀이고, 그래서  $\bar{L}_d$ 는 재귀열거이다. 이것은 정리 5.4에 모순된다.  $\square$

이상과 같이 재귀가 아닌 재귀열거 언어의 두 예  $L_u$ 와  $L_d$ 를 얻었다. Church-Turing 명제에 의해 재귀 언어를 컴퓨터로 풀 수 있는 문제라고 정의하였기 때문에, 정지문제는 계산불가 문제이다.

## 5.4 계산불가 문제

계산불가 문제인 정지문제를 찾았으므로 이를 이용하여 여러 가지 문제가 계산불가임을 보일 수 있다. 이를 위해 문제의 변환(reduction)이라는 기법을 사용하고자 한다.

정의 5.1 언어  $L_1$ 에서 언어  $L_2$ 로의 변환이란

$$w \in L_1 \text{인 경우에만 } f(w) \in L_2$$

를 만족하는 계산가능 함수  $f$ 이다.

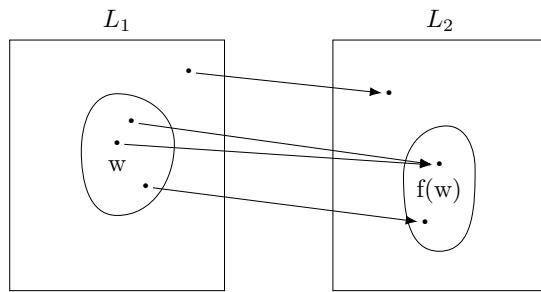


Figure 5.4:

그림 5.4를 보라. 여기에서 함수  $f$ 는 one-to-one 이나 onto일 필요는 없다.

$L_2$ 가 계산불가임을 보이기 위해서는 이미 계산불가임이 알려진 언어  $L_1$ 을 선택하여  $L_1$ 을  $L_2$ 로 변환한다. 변환의 의미는  $L_2$ 를 결정할 수 있으면  $L_2$ 의 답이  $L_1$ 의 답이 되므로  $L_1$ 을 결정할 수 있다. 그림 5.5를 보라. 대우에 의해  $L_1$ 이 계산불가이므로  $L_2$ 도 계산불가라는 결과를 얻는다. 여기에서 변환의 방향에 주의를 기울여야 한다.  $L_2$ 를  $L_1$ 로 변환하면 아무 결과도 얻지 못한다.

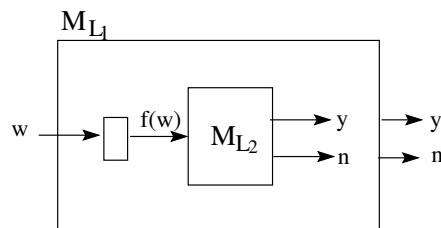


Figure 5.5:

**정리 5.8** “TM  $M$ 이 주어졌을 때,  $M$ 이  $\epsilon$ 에 대하여 정지하는가” 문제, 즉  $L_\epsilon = \{\langle M \rangle : \epsilon \in L(M)\}$ 은 계산불가이다.

**증명.**  $L_\epsilon$ 이 재귀라고 가정하자. 그러면  $L_\epsilon$ 을 결정하는 TM  $M_\epsilon$ 이 존재한다. 이제  $L_u$ 를  $L_\epsilon$ 으로 변환하자.  $L_u$ 의 입력  $\langle M, w \rangle$ 이 주어지면  $L_\epsilon$ 의 입력  $\langle M_w \rangle$ 로 변환한다.  $M_w$ 는 입력  $\epsilon$ 이 주어지면 테입에  $w$ 를 쓰고  $M$ 을 시작시키는 TM이다. 그림 5.6를 보라.  $\langle M, w \rangle$ 에서  $\langle M_w \rangle$ 를 얻기 위해서는 테입에  $w$ 를 쓰고  $M$ 의 초기상태로 들어가는 전이를  $\langle M \rangle$ 에 더해주면 된다. 그러면

- $M$ 이  $w$ 에 대하여 정지하면  $M_w$ 는  $\epsilon$ 에 대하여 정지하고
- $M$ 이  $w$ 에 대하여 정지하지 않으면  $M_w$ 는  $\epsilon$ 에 대하여 정지하지 않는다.

따라서  $M_\epsilon$ 이  $M_w$ 의  $\epsilon$ 에 대한 정지 여부를 알려 주면 그것이 바로  $M$ 이  $w$ 에 대한 정지 여부가 된다.  $\square$

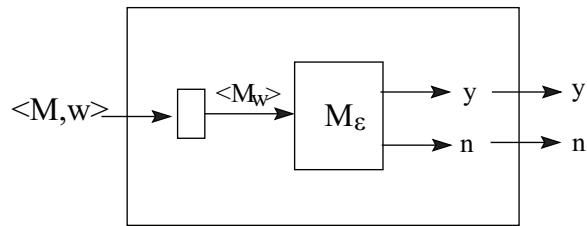


Figure 5.6:

**정리 5.9** “TM  $M$ 이 주어졌을 때,  $M$ 이 모든 입력에 대하여 정지하지 않는가” 문제, 즉  $L_\emptyset = \{\langle M \rangle : L(M) = \emptyset\}$ 은 계산불가이다.

**증명.**  $L_\emptyset$ 를 재귀라고 가정하면  $L_\emptyset$ 을 결정하는 TM  $M_\emptyset$ 이 존재한다. 이제  $L_\epsilon$ 을  $L_\emptyset$ 로 변환하자.  $L_\epsilon$ 의 입력  $\langle M \rangle$ 이 주어지면  $L_\emptyset$ 의 입력  $\langle M' \rangle$ 으로 변환한다.  $M'$ 은 주어진 입력을 지우고  $\epsilon$ 에 대하여  $M$ 을 시작시키는 TM이다.  $\langle M \rangle$ 에서  $\langle M' \rangle$ 를 얻기 위해서는 테입의 내용(주어진 입력)을 지우고  $M$ 의 초기상태로 들어가는 전이를  $\langle M \rangle$ 에 더해주면 된다. 그림 5.7을 보라. 그러면

- $M \circ | \epsilon$ 에 대하여 정지하면  $M'$ 은 입력이 무엇이든지 정지하게 되므로  $L(M') = \Sigma^*$ 이고
- $M \circ | \epsilon$ 에 대하여 정지하지 않으면  $L(M') = \emptyset$ 이다.

$M_\emptyset \circ | L(M') = \emptyset$ 인지 아닌지 알려주므로  $M \circ | \epsilon$ 에 대하여 정지하는지 아닌지 알 수 있다.  $\square$

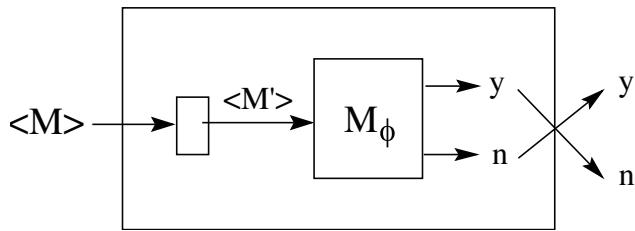


Figure 5.7:

**정리 5.10** “TM  $M \circ |$  주어졌을 때,  $L(M)$ 이 정규언어인가” 문제, 즉  $L_r = \{\langle M \rangle : L(M) \text{이 정규언어}\}$ 는 계산불가이다.

**증명.**  $L_r$ 을 재귀라고 가정하면  $L_r$ 을 결정하는 TM  $M_r \circ |$  존재한다.

이제  $L_\epsilon$ 을  $L_r$ 로 변환하자.  $L = \{0^n 1^n : n \geq 1\}$ 라고 하고  $M_L$ 을  $L$ 을 정의하는 TM이라고 하자. 즉  $L = L(M_L)$ .  $L_\epsilon$ 의 입력  $\langle M \rangle$ 이 주어지면  $L_r$ 의 입력  $\langle M' \rangle$ 으로 변환한다.  $M'$ 은

- 주어진 입력  $w$ 를 따로 (예를 들면, 둘째 테입에) 보관하고
- $\epsilon$ 에 대하여  $M$ 을 돌리고
- $M \circ |$  정지하면  $w$ 에 대하여  $M_L$ 을 돌리는

TM이다.  $\langle M \rangle$ 에서  $\langle M' \rangle$ 을 얻기 위해서는  $\langle M \rangle$ 과  $\langle M_L \rangle$ 에 다음의 전이를 더해주면 된다.

- 주어진 입력  $w$ 를 입력 테입에서 지우면서 따로 보관하고  $M$ 의 초기상태로 들어가는 전이

- $M$ 의 정지상태에서 시작하여  $w$ 를 다시 입력 테입에 적고  $M_L$ 의 초기상태로 들어가는 전이

그림 5.8을 보라. 그러면

- $M$ 이  $\epsilon$ 에 대하여 정지하면  $M'$ 은  $M_L$ 과 동일한 결과를 가지므로  $L(M') = L$ 이고
- $M$ 이  $\epsilon$ 에 대하여 정지하지 않으면  $M'$ 은  $w$ 가 무엇이든지 정지하지 않으므로  $L(M') = \emptyset$ 이다.

$\emptyset$ 는 정규언어이고  $L$ 은 정규언어가 아니므로  $M_r$ 이  $L(M') = \emptyset$ 인지 아닌지 알려준다. 그림 5.9을 보라. 그러므로  $M$ 이  $\epsilon$ 에 대하여 정지하는지 아닌지 알 수 있다.  $\square$

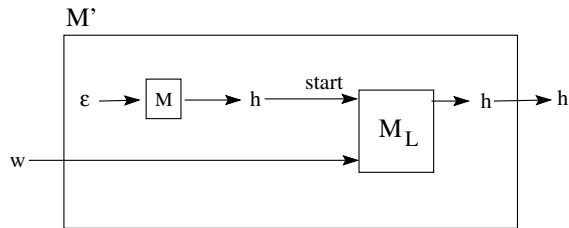


Figure 5.8:

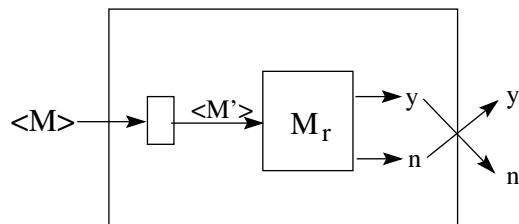


Figure 5.9:

이상과 같이 TM에 대한 대부분의 질문은 계산불가이다. 이를 하나로 요약한 것이 Rice의 정리이다.

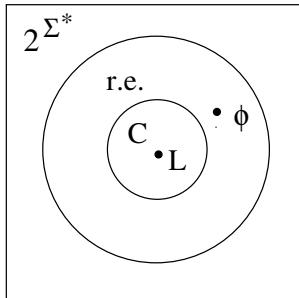


Figure 5.10:

**정리 5.11 (Rice)**  $C$ 를 재귀열거 언어 종류의 공집합이 아닌 진부분 집합이라고 하자.  $L_C = \{\langle M \rangle : L(M) \in C\}$ 는 계산불가이다.

**증명.** 먼저  $C$ 가 모든 재귀열거 언어의 집합이거나 공집합이면  $L_C$ 는 풀 수 있는 문제이다. 그럼 5.10를 보라.

- $C$ 가 모든 재귀열거 언어의 집합이면  $L_C$ 는  $\{\langle M \rangle : L(M) \text{은 재귀열거}\}$ 이다. 그런데 재귀열거의 정의에 의해  $L(M)$ 은 항상 재귀열거이다. 따라서 입력스트링  $w$ 가 TM의 부호이면  $w \in L_C$ 이고 TM의 부호가 아니면  $w \notin L_C$ 이다.  $w$ 가 TM의 부호인지는 쉽게 확인할 수 있으므로  $L_C$ 는 풀 수 있는 문제이다.
- $C$ 가 공집합이면  $L_C$ 는  $\{\langle M \rangle : L(M) \in \emptyset\}$ 이다. 재귀열거의 정의에 의해  $L(M)$ 이 재귀열거가 아닐 수 없다. 따라서 입력스트링  $w$ 가 TM의 부호이든지 아니든지  $w \notin L_C$ 이다.

그외의 경우에는  $L_C$ 가 계산불가임을 증명하자. 일반성을 잃지 않고 언어  $\emptyset$ 이  $C$ 에 포함되지 않는다고 가정하자. (그렇지 않으면 다음에서  $C$ 대신  $\bar{C}$ 를 고려하면 된다.)  $C$ 가 공집합이 아니므로  $C$ 에 속하는 언어  $L$ 이 존재한다.  $L$ 을 정의하는 TM을  $M_L$ 이라고 하자.

$L_C$ 를 재귀라고 가정하면  $L_C$ 를 결정하는 TM  $M_C$ 가 존재한다. 이제  $L_\epsilon$ 을  $L_C$ 로 변환하자. 그럼 5.11을 보라.  $L_\epsilon$ 의 입력  $\langle M \rangle$ 이 주어지면  $L_C$ 의 입력  $\langle M' \rangle$ 으로 변환한다.  $M'$ 은 앞의 증명에서와 같이

- 주어진 입력  $w$ 를 따로 보관하고

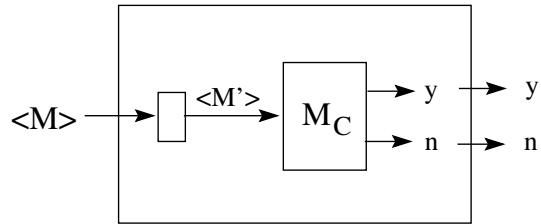


Figure 5.11:

- $\epsilon$ 에 대하여  $M$ 을 돌리고
- $M$ 이 정지하면  $w$ 에 대하여  $M_L$ 을 돌리는

TM이다. 그러면

- $M$ 이  $\epsilon$ 에 대하여 정지하면  $M'$ 은  $M_L$ 과 동일한 결과를 가지므로  $L(M') = L$ 이고
- $M$ 이  $\epsilon$ 에 대하여 정지하지 않으면  $M'$ 은  $w$ 가 무엇이든지 정지하지 않으므로  $L(M') = \emptyset$ 이다.

$L \in C$ 이고  $\emptyset \notin C$ 이므로  $M_C$ 가  $L(M') = L$ 인지 아닌지 알려준다. 그러므로  $M$ 이  $\epsilon$ 에 대하여 정지하는지 아닌지 알 수 있다.  $\square$

Rice 정리에 의하면 “컴퓨터 프로그램  $P$ 가 주어졌을 때,  $P$ 가 정렬(sorting)을 (제대로) 하는 프로그램인가” 문제도 계산불가이다. 또한 “컴퓨터 프로그램  $P$ 가 주어졌을 때,  $P$ 가 특정 specification을 만족하는 프로그램인가” 문제도 계산불가이다

TM에 대한 질문 외에 좀 더 구체적인 문제 중에 계산불가인 문제도 많이 있다.

**정의 5.2 Post 대응문제:**  $P = ((u_1, v_1), \dots, (u_n, v_n))$ 이 주어졌을 때 ( $u_i, v_i \in \Sigma^+$ ), 어떤  $i_1, \dots, i_r$ 에 대하여  $w = u_{i_1} \cdots u_{i_r} = v_{i_1} \cdots v_{i_r}$ 을 만족하는  $w$ 를  $P$ 의 대응이라고 부른다. Post 대응문제는  $P$ 의 대응이 존재하는가 하는 문제이다.

**예제 5.4**  $P = ((a, ab), (b, ca), (ca, a), (abc, c))$  일 때,

$$\begin{aligned} & abcaaabc \\ &= u_1u_2u_3u_1u_4 \\ &= v_1v_2v_3v_1v_4 \end{aligned}$$

이므로  $P$ 의 대응이 존재한다.

**정리 5.12** Post 대응문제는 계산불가이다.

문맥무관 문법에 대한 다음 질문들도 계산불가이다.

**정리 5.13** “문맥무관 문법  $G$ 가 주어졌을 때,  $G$ 가 애매한가” 문제는 계산불가이다.

**증명.** Post 대응 문제를 문맥무관 문법의 애매성 문제로 변환한다. Post 대응 문제의 알파벳  $\Sigma$ 에 속하지 않는 글자  $a_1, \dots, a_n$ 를 선택한다. 문법  $G_1$ 은

$$S_1 \rightarrow u_i S_1 a_i \mid u_i a_i$$

$(1 \leq i \leq n)$ 으로 구성되고, 문법  $G_2$ 는

$$S_2 \rightarrow v_i S_2 a_i \mid v_i a_i$$

로 구성되고, 문법  $G$ 는 위의 생성규칙에

$$S \rightarrow S_1 \mid S_2$$

를 더하여 얻는다.

문법  $G_1$ 은 애매하지 않고  $G_2$ 도 마찬가지이다. 따라서 문법  $G$ 가 애매한 경우는  $G_1$ 과  $G_2$ 가 같은 스트링을 만드는 경우이다. 즉 Post 대응문제가 대응을 가지는 경우에만  $G$ 는 애매하다.  $\square$

**정리 5.14** “문맥무관 문법  $G_1, G_2$ 가 주어졌을 때,  $L(G_1) \cap L(G_2) = \emptyset$ 인가” 문제는 계산불가이다.

**증명.** 역시 Post 대응문제를 변환한다. 앞의 증명과 같이 Post 대응문제에서  $G_1$ 과  $G_2$ 를 얻는다. 그러면 Post 대응문제가 대응을 가지는 경우에만  $L(G_1) \cap L(G_2) \neq \emptyset$ 이다.  $\square$

## 5.5 연습 문제

1. 무한집합의 크기
  - 1.1.  $N \times N \times N$ 이 셀 수 있는 집합인지 아닌지 증명하라.
  - 1.2. 양의 정수  $i, j, k$ 에 대하여 모든  $(i, j, k)$ 의 집합이 셀 수 있는 집합임을 증명하라.
  - 1.3.  $A$ 와  $B$ 가 셀 수 있는 무한집합이라고 하면  $A \times B$ 도 셀 수 있는 무한집합임을 증명하라.
2. 범용 튜링기계
  - 2.1.  $\langle M, w \rangle$ 을 입력으로 받아들이는 범용 TM  $M_u$ 의 동작을 설명하라.
3. 정지문제
  - 3.1.  $\Sigma = \{0, 1\}$ ,  $L_d = \{\langle M \rangle : \langle M \rangle \in L(M)\}$ 라고 하자.  $L_d$ 의 여집합은 재귀 열거가 아님을 증명하라.
  - 3.2. 재귀 언어 종류가 다음 연산에 닫혀 있는지 아닌지 증명하라.
    - (a) 합집합
    - (b) 접합
    - (c) 교집합

3.3. 언어  $L$ 과 여집합인  $\bar{L}$ 가 재귀 열거이면  $L \cap \bar{L}$ 이 재귀임을 증명하라.

#### 4. 계산불가 문제

4.1. 다음 문제를 언어로 표현하고 계산불가임을 증명하라.

- (a) TM  $M$ 이 주어졌을 때,  $L(M)$ 이 길이가 3인 스트링을 포함하는가
- (b) TM  $M_1$ 과  $M_2$ 가 주어졌을 때,  $L(M_1) \subseteq L(M_2)$  인가
- (c) TM  $M_1$ 과  $M_2$ 가 주어졌을 때,  $L(M_1) = L(M_2)$  인가
- (d) DFA와 정규식이 동등한가

4.2. 다음 언어가 계산불가임을 증명하라.

- (a)  $\{\langle M \rangle : L(M) = 0^*\}$
- (b)  $\{\langle M \rangle : L(M)$ 은 단일집합(singleton set) $\}$

4.3. 이진 알파벳( $\Sigma = \{0, 1\}$ )에서 Post 대응문제가 계산불가임을 증명하라.

4.4. TM  $M$ 과 정규언어  $L$ 이 주어졌을 때  $L(M) \cap L = \emptyset$ 이 계산불가임을 증명하라.

4.5.  $P = ((u_1, v_1), \dots, (u_n, v_n))$ 가 주어졌을 때 짹수인 정수  $i_1, \dots, i_r$ 에 대해  $w = u_{i_1} \cdots u_{i_r} = v_{i_1} \cdots v_{i_r}$ 이면  $w$ 를 짹수 대응이라고 한다.  $P$ 가 짹수 대응인지 아닌지 결정하는 문제는 계산불가임을 증명하라.

4.6. 다음 문제가 계산불가임을 증명하라.

- (a) TM  $M$ 이 주어졌을 때,  $L(M) = \Sigma^*$
- (b) TM  $M$ 이 주어졌을 때,  $L(M)$ 이 문맥무관 언어
- (c) TM  $M_1$ 과  $M_2$ 가 주어졌을 때,  $L(M_1) \cap L(M_2) = \emptyset$
- (d) TM  $M$ 이 주어졌을 때,  $L(M)$ 이 유한집합
- (e) TM  $M_1$ 과  $M_2$ 가 주어졌을 때,  $L(M_1) = L(M_2)$
- (f) TM  $M_1$ 과  $M_2$ 가 주어졌을 때,  $L(M_1) \cap L(M_2) \neq \emptyset$

- 4.7. 다음 문제를 Post 대응문제로부터 변환하여 계산불가임을 증명하라: 두 개의 문맥무관 문법  $G_1$ 과  $G_2$ 가 주어졌을 때,  $L(G_1) \cap L(G_2) = \emptyset$
- 4.8. 다음 문제가 계산가능한지 아닌지 증명하라.
- $\{\langle M \rangle : L(M) \text{은 재귀 열거}\}$
  - $\{\langle M \rangle : L(M) = a^*\}$
  - $\{\langle M \rangle : L(M) \text{은 결정 문맥무관 언어}\}$

# Chapter 6

## 계산 복잡도

지금까지 문제들을 모델(오토마타, 문법 등)의 표현 능력에 의해 분류하였다. 앞장에서는 이를 크게 계산가능 문제와 계산불가 문제로 나누었다.

본 장에서는 문제를 푸는데 소요되는 시간과 공간의 양에 의해 계산가능 문제를 분류하고자 한다. 계산 모델로는  $k$  테입 튜링기계를 사용한다. 정렬(sorting), 최소 신장트리(minimum spanning tree) 문제 등에 대한 정확한 시간복잡도를 얻기 위해서는 현재 컴퓨터와 거의 동일한 계산 모델이 필요한데, 이러한 모델을 random access machine이라고 부른다. 그러나 본 장에서는 다행 시간 이상의 문제 종류만을 다루므로  $k$  테입 튜링기계를 사용하여도 동일한 문제 종류를 얻을 수 있다.

입력의 길이를  $n$ 이라고 하자. 시간(공간)복잡도가  $n$ 에 대한 다항식 형태 즉  $O(n^c)$ 일 때, 이를 다행 시간(공간)이라고 한다. 시간(공간)복잡도가  $n$ 에 대한 지수식 형태 즉  $O(2^{n^c})$ 일 때, 이를 지수시간(공간)이라고 한다.

**정의 6.1** 모든 입력에 대하여 다행 시간 내에 정지하면서 언어  $L$ 을 결정하는 튜링기계가 존재하면  $L$ 은 언어 종류  $P$ 에 속한다.

**정의 6.2** 모든 입력에 대하여 다행 공간 만을 사용하면서 언어  $L$ 을 결정하는 튜링기계가 존재하면  $L$ 은 언어 종류  $PSPACE$ 에 속한다.

**정의 6.3** 모든 입력에 대하여 지수 시간 내에 정지하면서 언어  $L$ 을 결정하는 튜링기계가 존재하면  $L$ 은 언어 종류  $EXP$ 에 속한다.

다항 시간 내에는 다항 공간밖에 사용할 수 없으므로  $P \subseteq PSPACE$ 이다. 다항 공간을 사용하면서 만들어지는 튜링기계의 모든 상황은 지수 개이므로  $PSPACE \subseteq EXP$ 이다. 따라서

$$P \subseteq PSPACE \subseteq EXP$$

**정의 6.4** 모든 입력에 대하여 다항 시간 내에 정지하면서 언어  $L$ 을 결정하는 비결정 튜링기계가 존재하면  $L$ 은 언어 종류  $NP$ 에 속한다.

결정 튜링기계는 비결정 튜링기계의 일종이므로  $P \subseteq NP$ 이고, 비결정 튜링기계도 다항 시간 내에는 다항 공간밖에 못 쓰므로  $NP \subseteq PSPACE$ 이다. 따라서

$$P \subseteq NP \subseteq PSPACE \subseteq EXP.$$

각 언어 종류에 속하는 문제들의 예를 들면 다음과 같다.

- $P$ : 정렬, 최소 신장트리(minimum spanning tree) 등
- $NP$ : 만족가능성(satisfiability) 문제, 외판원 문제 등
- $PSPACE$ : quantified boolean formula (QBF), 게임 등

$P$ 가  $EXP$ 의 진부분집합임은 알려져 있지만, 그외의 진부분집합 관계는 알려져 있지 않다. 이외에도  $EXPSPACE$ , 이중 지수 시간 등 다양한 언어 종류의 계층을 생각할 수 있으나, 현실에서 접하는 대부분의 문제는  $P$ 와  $NP$ 에 속한다.  $P$ 를 컴퓨터로 빨리 풀 수 있는 문제의 집합으로 간주하기 때문에  $P$ 와  $NP$ 의 구분이 현실적으로 중요한 문제이다.

**정의 6.5** 다항시간 내에 함수  $f$ 를 계산하는 TM이 존재하면,  $f$ 를 다항시간 함수라고 부른다. 언어  $L_1$ 에서 언어  $L_2$ 로의 다항시간 변환이란

$$w \in L_1 \text{인 경우에만 } f(w) \in L_2$$

를 만족하는 다향시간 함수  $f$ 이다.

언어  $L_1$ 에서 언어  $L_2$ 로의 다향시간 변환이 존재하면  $L_1 \leq_P L_2$ 로 표시한다.

**정의 6.6** 다음 두 조건이 만족되면, 언어  $L$ 을  $NP$ 완전이라고 부른다.

- $L \in NP$ 이고
- $NP$ 에 속한 모든 언어  $A$ 가  $L$ 로 다향 시간에 변환된다 (즉,  $A \leq_P L$ ).

만족가능성 문제, 외판원 문제 등 많은 문제들이  $NP$ 완전이다.

실제 상황에서  $NP$ 완전 문제를 만났을 때, 이에 대응하는 방법은 아래와 같다.

- 제한된 문제를 푼다: 주어진 문제에 약간의 조건을 더하면 다향 시간에 풀 수 있는 경우가 있다.
- 근사 알고리즘(approximation algorithm)을 찾는다: 최적해에 가까운 근사해를 다향 시간에 찾는다.
- 백트래킹(backtracking): 상태공간(state space)을 깊이 우선 탐색으로 순회한다. 최악의 경우 지수 시간이 걸린다.
- 분지 한정(branch-and-bound): cost function을 사용하여 상태 공간의 일부만 탐색한다. 최악의 경우 지수 시간이 걸린다.
- 부분 개선: 임의의 해를 구한 후, 부분 개선을 통해 더 좋은 해를 구해 나간다.

## 연습 문제

1. 언어 종류  $P$ ,  $NP$ ,  $PSPACE$ ,  $NP$ -complete,  $EXP$ 를 정의하고 각 종류의 예제를 보여라. 각 종류간의 관계를 보이고 이유를 설명하라.
2. 문제  $A$ 가  $NP$ -complete임을 증명하는 방법을 설명하라.



# Bibliography

- [ASU] A.V. Aho, R. Sethi and J.D. Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, 1986.
- [DK] D.Z. Du and K.I. Ko, Problem Solving in Automata, Languages, and Complexity, John Wiley & Sons, 2001.
- [GJ] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, 1979.
- [HMU] J.E. Hopcroft, R. Motwani and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation, 3rd ed., Addison-Wesley, 2007.
- [Ka] R. H. Katz, Contemporary Logic Design, Benjamin/Cummings, 1994.
- [LP] H.R. Lewis and C.H. Papadimitriou, Elements of the Theory of Computation, 2nd ed., Prentice-Hall, 1998.
- [Li] P. Linz, An Introduction to Formal Languages and Automata, 4th ed., Jones and Bartlett, 2006.

## 용어

accept	받아들이다
automata	오토마타
binary relation	이원관계
closure	닫음
complement	여집합
context-free grammar	문맥무관 문법
context-sensitive grammar	문맥민감 문법
corollary	따름정리
derive	유도하다
encoding	부호
formal language	형식언어
grammar	문법
if and only if	경우에만
language	언어
leftmost derivation	좌측 유도
lemma	보조정리
production rule	생성규칙
pushdown automata	내리누름 오토마타
recursively enumerable	재귀열거
recursive	재귀
reject	거부하다
sentential form	문장 형태
simulate	흉내내다
uncomputable	계산불가
universal Turing machine	범용튜링기계
without loss of generality	일반성을 잃지 않고