튜링기계

제 1 절 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 1을 보라.

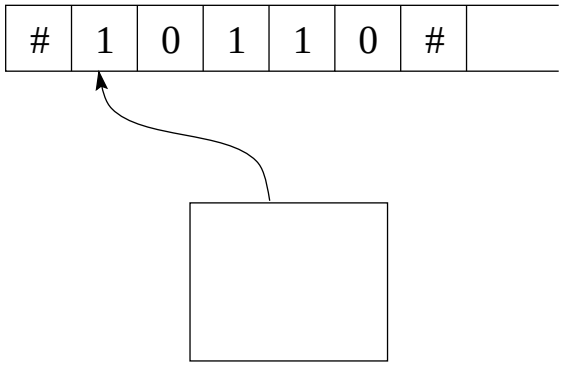

그림 1:

정의 1 *튜링 기계 M은 여섯 가지 요소 $(Q, \Sigma, \Gamma, \delta, q_0, H)$로 구성된다. 여기에서*

1. *Q는 상태들의 유한 집합이고*
2. *Σ는 입력 알파벳이고*
3. *Γ는 테입 알파벳이고 공백 글자 $\#$를 포함한다.*
4. *δ는 $Q' \times \Gamma$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 δ가 전이 함수이므로 결정 오토마타이다. δ의 정의에서 L은 헤드가 왼쪽으로 R은 헤드가 오른쪽으로 한 칸 움직이는 것을, S는 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 u이고 헤드 상의 글자가 a이고 헤드 오른쪽의 스트링이 v이면 (v는 오른쪽 끝의 공백 아닌 글자까지 포함한다), 현재 상황은 $(q, u\underline{a}v)$로 표시된다. 입력 스트링이 w일 때, 시작 상황은 $(q_0, \underline{\#}w)$이다.

예제 1 *테입 구성이 $\#aa\underline{a}aa$에서 헤드가 오른쪽으로 가면 $\#aaaa\underline{a}a$가 되고, 이어서 왼쪽으로 가면 원래의 모습이 된다. $\#aa\underline{a}$에서 오른쪽으로 가면 $\#aaa\underline{\#}$가 되고, 다시 오른쪽으로 가면 $\#aaa\#\underline{\#}$이 된다. 이어서 왼쪽으로 두 번 가면 원래 모습이 된다.*

정의 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 M을 정지하게 만드는 스트링의 집합이다. L을 정의하는 TM M(즉 $L = L(M)$)이 존재하는 언어 L을 재귀열거 언어라고 부른다.

예제 2 $L = \{0^n 1^n : n \geq 1\}$을 정의하는 TM M을 구하라.

M은 오른쪽으로 한 칸 가서, 맨 왼쪽의 0을 x로 바꾸고 (q_0), 오른쪽으로 진행하여 맨 왼쪽의 1을 찾아 y로 바꾸고 (q_1), 왼쪽으로 진행하여 맨 오른쪽의 x를 찾는 (q_2) 작업을 반복적으로 수행한다. 또한 q_0은 0이 없을 때 오른쪽으로 진행하여 1도 없으면 (q_3) 정지한다. 예: $000111 \ldots x00y11$.

구체적으로 $M = (\{q_0, \ldots, q_5\}, \{0, 1\}, \{0, 1, x, y, \#\}, \delta, q_0, \{q_5\})$이고

δ는 다음과 같다.

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

여기에서 모든 $q \in Q, a \in \Gamma$에 대하여 빈칸은 $\delta(q, a) = (q_4, a, S)$를 의미한다. 즉 q_4는 무한 순환하는 상태이다. 앞으로 무한 순환 상태는 표시하지 않기로 한다. M을 그림으로 나타내면 그림 2과 같다.

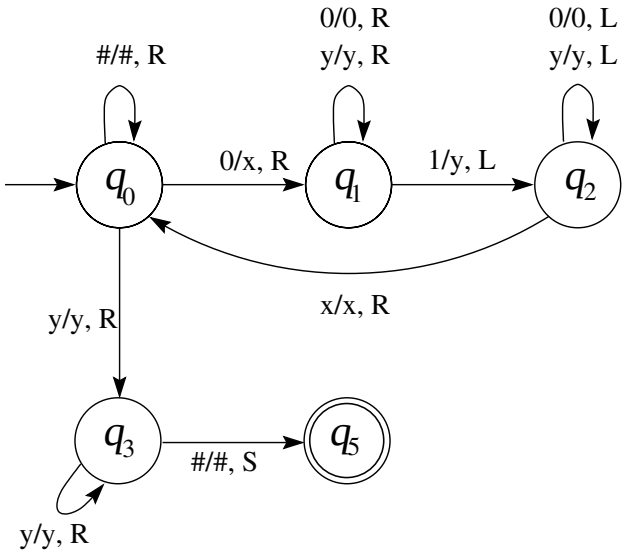

그림 2:

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 3 $L = \{ww : w \in \{0,1\}^+\}$*을 정의하는 TM을 구하라.*

먼저 입력 스트링을 앞부분과 뒷부분으로 나누기 위하여 앞부분의 $0, 1$*은 각각* x, y*로 뒷부분의* $0, 1$*은 각각* x', y'*로 바꾼다. 예:* $110011 \ldots yyxx'y'y'$*. 그림 3.*

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

이후 앞부분과 뒷부분을 맞추어 보는 것은 앞의 예제와 비슷하다.

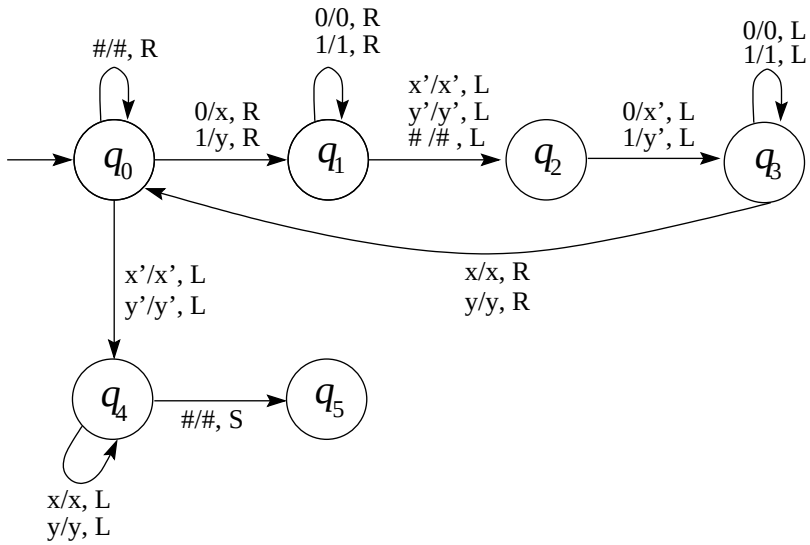

그림 3:

또한 TM은 모든 입력에 대해 정지하면서 정지상태에 따라 언어를

정의할 수 있다.

정의 3 *언어 L에 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 M이 L을 결정한다고 말한다. L을 결정하는 TM이 존재하면 L을 재귀 언어라고 부른다.

예제 4 *$L = \{0^n 1^n : n \geq 0\}$에 대하여 예 2의 TM에서*
$H = \{q_4, q_5\}$이고 $q_4 = n, q_5 = y$이면 이는 L을 결정하는 TM이다.

정리 1 *재귀언어 종류는 재귀열거언어 종류의 부분집합이다.*

증명. L을 재귀 언어라고 하면, L을 결정하는 TM M이 존재한다. M에서 상태 n을 무한 순환 상태로 만들고 y만을 정지 상태로 하면, L을 정의하는 TM을 얻는다. 따라서 L은 재귀열거 언어이다. □

TM은 입력에서 출력을 계산하는 도구로 사용될 수도 있다. 이 경

우 TM은 어떤 함수를 계산하게 된다. 자연수에서 자연수로의 함수를 다룰 때, 자연수를 단일진법으로 표시한다. 즉 자연수 n은 1^n으로 표시된다.

정의 4 *f를 Σ^*에서 Σ^*로의 함수라고 하자. TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \{h\})$이*

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

을 만족하면 M이 f를 계산한다고 말한다. 이 경우 f를 계산가능 함수라고 부른다.

함수 f가 입력으로 k 개의 자연수를 취할 때 즉 $f(n_1, \ldots, n_k)$일 때, 입력은 $1^{n_1}0\cdots01^{n_k}$로 표시된다.

예제 5 *더하기 함수 $f(n, m) = n + m$을 계산하는 TM을 구하라.*

입력이 1^n01^m이므로 0을 1로 바꾸고 맨 끝의 1을 공백으로 바꾸면

된다. TM의 전이함수는 다음과 같다. 그림 4.

	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		$(q_3, 1, L)$	$(h, \#, S)$

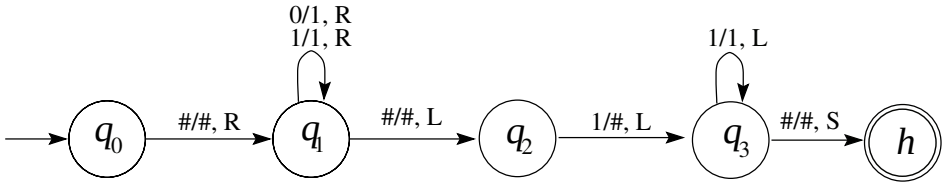

그림 4:

예제 6 *곱하기 함수 $f(n, m) = nm$을 계산하는 TM을 구하라.*

입력 중 n이 0이면 테입의 내용을 다 지우고 끝난다 (즉 0을 출력). n이 2 이상이면 m을 $n-1$ 번 복사한 후 테입에 있는 0을 지우면서 1을 왼쪽으로 이동시킨다. 그림 5.

$n = 2, m = 3$일 때 테입의 변화

$\#1\underline{1}0111\#$

$\#\underline{1}00111\#$

$\#a0011\underline{1}0\#$

$\#a0011b\underline{0}1\#$

$\#a0\underline{0}1110111\#$

$\#000111111\underline{1}\#\#$

$\underline{\#}111111\#$

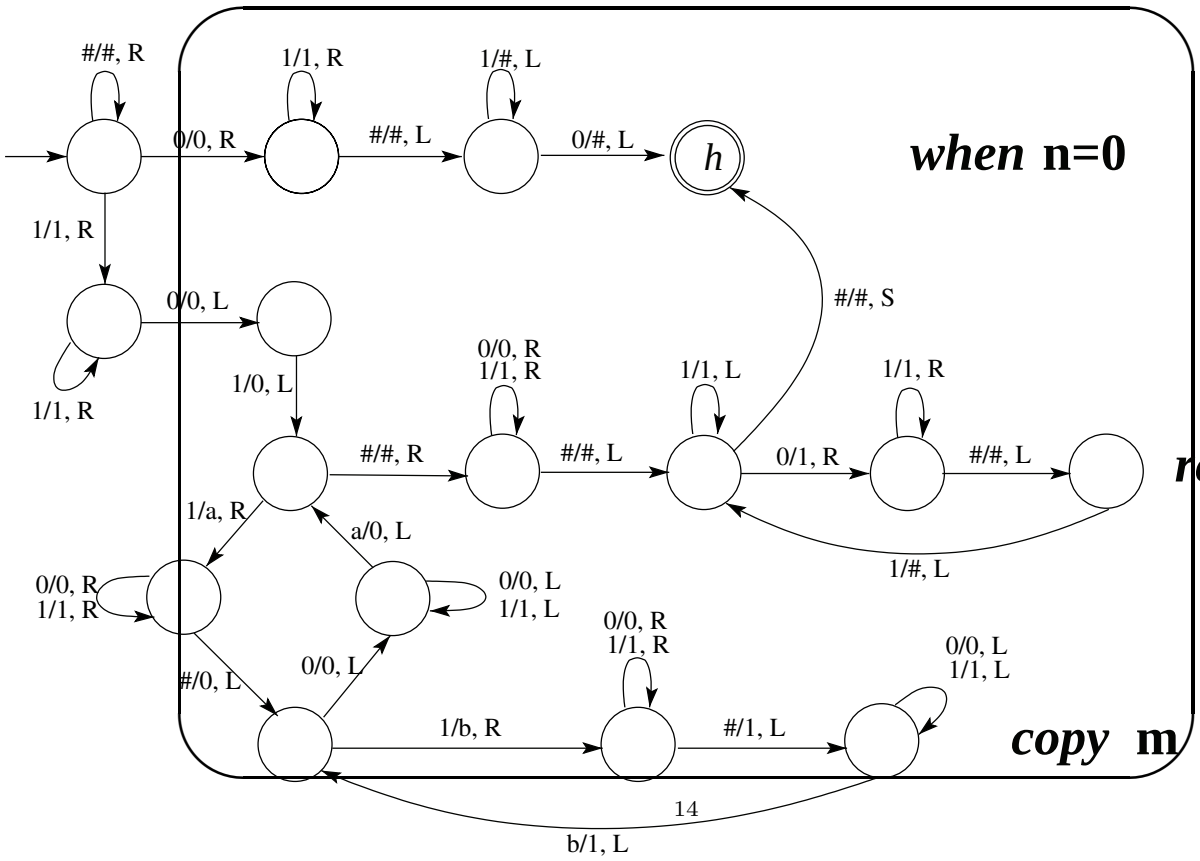

그림 5:

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.

제 2 절 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.

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

예제 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 8 *k테입 TM은 원래 TM보다 간단하게 계산을 할 수 있다. 예를 들어 $\underline{\#}w$를 $\underline{\#}w\#w$로 바꾸는 계산을 고려하자. 원래 TM은 좌우로 움직이면서 w를 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 $(\underline{\#}w, \underline{\#})$에서 $(\#w\underline{\#}, \#w\underline{\#})$로 바꾼다.*
2. *둘째 테입의 헤드를 왼쪽으로 옮긴다. 즉 $(\#w\underline{\#}, \underline{\#}w\#)$로 바꾼다.*
3. *두 헤드를 오른쪽으로 움직이면서 둘째 테입의 내용을 첫째 테입에 적는다. 즉 $(\#w\#w\underline{\#}, \#w\underline{\#})$로 바꾼다.*
4. *첫째 테입의 헤드를 왼쪽으로 옮긴다.*

이와 같이 특정한 계산을 하는 편이성(또는 시간)에 있어서는 k테입 TM이 원래 TM보다 나을 수 있지만, 어떤 계산을 할 수 있느냐 없느냐의 관점에서는 두 개가 동일하다. 즉 k테입 TM이 할 수 있는 계산은 원래 TM도 할 수 있다.

정리 2 *k테입 TM $M = (Q, \Sigma, \Gamma, \delta, q_0, H)$와 동등한 원래 TM M'이 존재한다 (즉 $L(M) = L(M')$).*

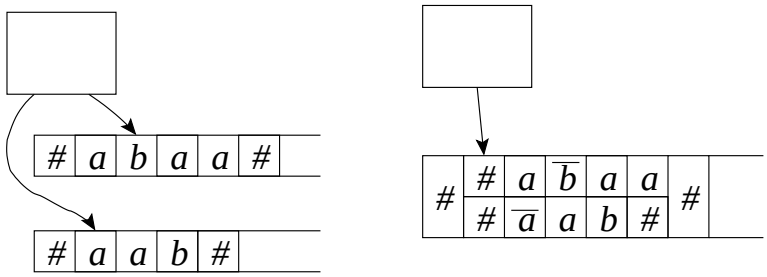

그림 6:

증명. 원래 TM M'이 M을 흉내내기 위해서는 k 테입의 내용을 보관하고 있어야 한다. 이를 위해 M'의 테입을 k 개의 트랙(track)으로 나눈다. 그림 6. $\bar{\Gamma} = \{\bar{a} : a \in \Gamma\}$라고 하자. $\bar{a}$는 M에서 헤드가 a글자에 있음을 나타낸다. 그림과 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 Γ'으로 $\Gamma \cup (\Gamma \cup \bar{\Gamma})^k$을 사용하는 것뿐

이다. 즉 M'이 $(\bar{a}, b)$를 읽으면 첫째 테입의 이 위치에 a가 있고 여기에 헤드가 있으며 둘째 테입에는 b가 있다는 것을 알 수 있다.

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

1. 입력을 하나씩 오른쪽으로 옮긴 후 테입을 M의 k 테입 형태로 바꾼다. $w = abaa$이고 M이 2 테입 TM이면, M'의 테입을 $\#abaa$에서 $\#\#abaa$로, 그리고 $\#(\bar{\#}, \bar{\#})(a, \#)(b, \#)(a, \#)(a, \#)$으로 바꾼다.
2. M의 계산을 흉내낸다. M의 한 전이를 흉내내기 위해 M'은 다음의 움직임을 한다.
   1. 테입을 지나가면서 M의 k 헤드에 있는 글자를 읽는다. M의 상태를 기억하고 k 헤드에 있는 글자를 읽고 나면, M의 전이를 알 수 있다.
   2. 테입을 지나가면서 M의 전이를 반영한다. 만약 M이 어떤 헤드를 오른쪽으로 움직여서 M'이 $\#$이 들어있는 칸으로 옮겨야 하면 M은 먼저 $\#$를 $(\#, \#)$로 바꾼 후 M의 전이를 반영한다.

3. M이 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.

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

1. 두 헤드를 $\underline{\#}w\underline{\#}$의 위치에 놓는다.

2. 두 헤드를 오른쪽으로 움직이면서 첫째 테입의 내용을 둘째 테입에 적는다. 즉 $\#w\underline{\#}w\underline{\#}$로 바꾼다.

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

제 3 절 비결정 튜링기계

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

정의 5 *비결정 TM $M = (Q, \Sigma, \Gamma, \Delta, q_0, H)$이 정의하는 언어 $L(M)$은 다음과 같다.*

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

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

정리 3 *비결정 TM $M = (Q, \Sigma, \Gamma, \Delta, q_0, H)$와 동등한 원래 TM이 존재한다.*

증명. M은 비결정 TM이므로 한 상황 C에서 여러 개의 상황 $C_1, \ldots, C_i$로 전이할 수 있다. 여기에서 i는 비결정 TM의 정의에 의해 $r = |Q| \times |\Gamma| \times 3$ 이하의 값을 갖는다.

M의 계산과정을, 상황을 정점으로 (시작상황을 루트로) 전이를 간선으로 표시하여 트리 T로 나타낼 수 있다. 이 트리에서 한 정점의 자식의 개수는 최대 r이다. 따라서 이 트리의 모든 정점을 정수 $1, \ldots, r$의 스트링으로 표시할 수 있다. 루트는 1이고 루트의 자식은 $11, \ldots, 1r$이고 11의 자식은 $111, \ldots, 11r$이다. 물론 11의 자식이 r보다 적으면 $11r$에 해당하는 정점(상황)은 없게 된다. 그림 7.

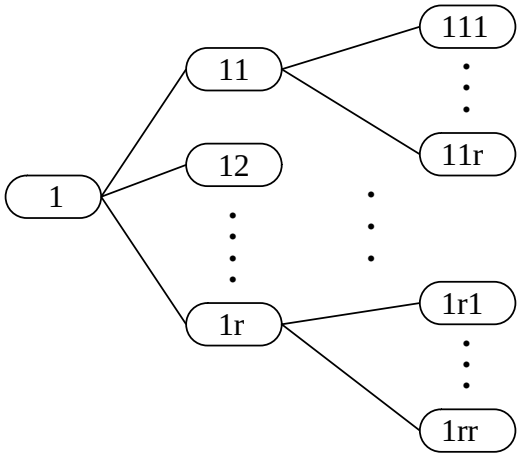

그림 7:

이제 M의 계산과정을 3테입 TM M'이 흉내내고자 한다. 즉 M'은 트리 T의 정점에 해당하는 상황을 모두 만들어 가면서 그 중에서 정지상황을 만나면 정지하려고 한다. T가 유한 트리가 아닐 수 있

으므로 T를 깊이 우선 탐색으로 찾아가면 T에 정지상황이 있는데도 불구하고 이를 찾지 못할 수 있다. 따라서 M'은 T를 너비 우선 탐색으로 찾아간다.

T를 너비 우선 탐색으로 찾아가기 위해서는 일반적으로 큐(queue)가 필요한데, 다음과 같이 큐를 사용하지 않고 시간을 더 사용하여 해결할 수 있다. 즉 T의 각 정점을 나타내는 $\{1, \ldots, r\}$ 상의 스트링을 너비 우선 탐색의 순서로 하나씩 만들면서 각 스트링에 해당하는 상황을 시작상황에서부터 (이전 상황을 기억하지 않고 있으므로) 만들면 된다.

M'은 3테입 TM인데, 첫째 테입에는 입력 w를 보관하고 첫째 테입의 내용은 변하지 않는다. 둘째 테입에는 $\{1, \ldots, r\}$ 상의 스트링을 너비 우선 탐색의 순서로 하나씩 만든다. 둘째 테입에 어떤 스트링 $a_1 \cdots a_i$가 주어지면 셋째 테입을 이용하여 이 스트링에 해당하는 상황을 만든다. 즉 첫째 테입에 있는 w를 셋째 테입에 복사한 후 시작상황에서 M의 a_1번째 전이를 적용하여 다음 상황을 만들고 여기에 다시 a_2번째 전이를 적용하는 식으로 진행한다. $a_1 \cdots a_i$에 해당

하는 상황이 정지상황이면 M'도 정지하고, 그렇지 않으면 둘째 테입에 다음 스트링을 만들고 반복한다. □

제 4 절 무제한 문법

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

예제 9 *다음 문법은* $L = \{a^n b^n c^n : n \geq 1\}$*을 생성한다.*

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

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

증명. M은 비결정 2테입 TM이다. M의 첫째 테입에는 입력 w가 주어진다. 둘째 테입에는 G의 문장형태가 만들어진다. 처음에 둘째 테입에 S를 쓰고 다음을 반복한다.

1. 비결정적으로 둘째 테입의 모든 위치 i를 선택한다. (즉 왼쪽 끝에서 시작하여 오른쪽으로 진행하면서 현재 위치를 선택하는 작업을 반복한다.)
2. 비결정적으로 G의 모든 생성규칙 $x \to y$을 선택한다.
3. 둘째 테입의 $i, \ldots, i + |x| - 1$ 위치의 글자가 x이면, x를 y로 바꾼다. 이 과정에서 $|x| \neq |y|$이면 x의 오른쪽에 있는 글자를 왼쪽이나 오른쪽으로 이동시켜야 된다.
4. 둘째 테입에 있는 문장형태를 w와 비교하여 같으면 정지한다.

G가 w를 생성하면 M은 언젠가 정지하고 그렇지 않으면 M은 정지하지 않는다. 따라서 $L(M) = L(G)$이다. □

정리 5 *TM $M = (Q, \Sigma, \Gamma, \delta, q_0, H)$가 정의하는 언어 $L(M)$을 생성하는 무제한 문법 G가 존재한다.*

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

M의 전이를 반대로 흉내내는 문법 G를 만들려고 한다. 즉 M이 시작상황 $(q_0, \underline{\#}w)$에서 $(h, \underline{\#})$로 전이할 때, G는 $[\#h]$에서 $[\#q_0w]$를 유도하려고 한다. 이를 위해 M의 각 전이에 대하여 G는 다음의 생성규칙을 갖는다.

- $\delta(q, a) = (p, b, S)$: G는 $bp \to aq$를 갖는다. 그림 8.
- $\delta(q, a) = (p, b, R)$: G는 모든 $c \in \Sigma \cup \{\#\}$에 대하여 $bcp \to aqc$를 갖는다. 또한 $b\#p] \to aq]$를 갖는다. 그림 9, 10.
- $\delta(q, a) = (p, b, L)$: $b \neq \#$이면 G는 $pb \to aq$를 갖는다. $b = \#$이면 모든 $c \in \Sigma$에 대하여 $p\#c \to aqc$를 갖고 또한 $p] \to aq]$를 갖는다. 그림 11, 12, 13.

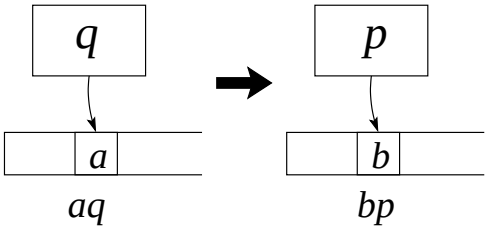

그림 8:

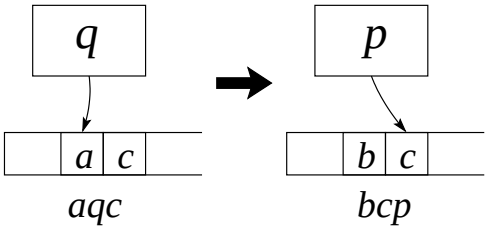

그림 9:

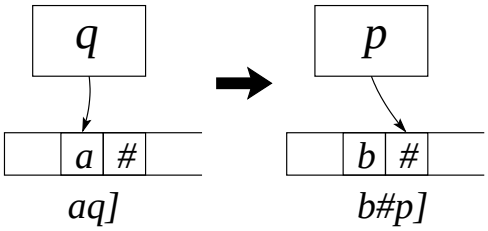

그림 10:

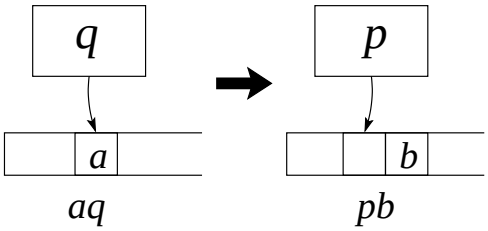

그림 11:

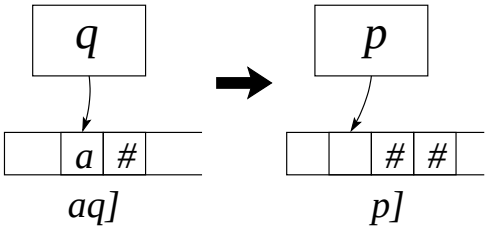

그림 12:

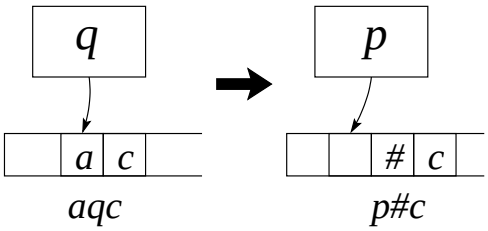

그림 13:

처음과 마지막을 위하여 G는

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

을 갖는다.

그러면 M이 $(q_0, \underline{\#}w) \vdash^* (h, \underline{\#})$로 전이할 때, G는

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

를 유도하게 된다. □

제 5 절 Church-Turing의 명제

먼저 모든 문제를 언어로 바꾸어 생각할 수 있음을 보이자. 모든 문

제는 결정 문제(decision problem)와 최적화 문제(optimization problem)의 두 가지 유형으로 분류할 수 있다. 결정 문제란 답이 예 또는 아니오인 문제이고, 최적화 문제란 어떤 조건을 만족하는 최소값 또는 최대값을 구하는 문제이다.

정의 6 *그래프에서 모든 정점을 포함하는 단순 사이클을 해밀톤 사이클이라고 한다. 해밀톤 사이클 문제는 그래프 G가 주어졌을 때 G가 해밀톤 사이클을 갖고 있느냐는 문제이다.*

예제 10 *그림 14에서 왼쪽 그래프는 해밀톤 사이클을 갖고 있지 않고, 오른쪽 그래프는 해밀톤 사이클을 갖고 있다.*

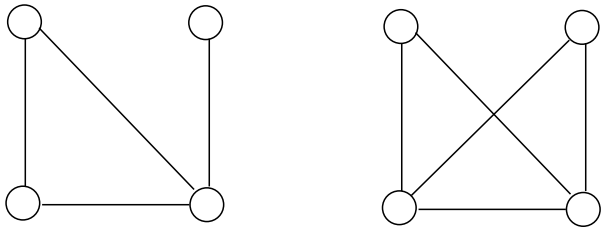

그림 14:

정의 7 *외판원(traveling salesman) 문제는 완전 그래프 G와 모든 간선 e의 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 I를 스트링 w로 바꾸고, I의 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 구한다면 가정한다. 외판원 문제에 대한 결정 문제는 다음과 같다. 완전 그래프 G와 간선의 거리 $d(e)$와 양의 정수 A가 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 8 (Church-Turing 명제) *재귀 언어를 컴퓨터로 풀 수 있는 문제라고 하자.*

재귀 언어를 결정하는 TM을 알고리즘이라고 부른다.

제 6 절 Chomsky 체계

정규언어와 문맥무관 언어 외에 문맥민감 언어가 있다. 문맥무관 언어의 표준형에서와 같이 다음의 언어는 ϵ을 포함하지 않는다고

가정한다.

정의 9 *문법 $G = (V, \Sigma, S, P)$의 모든 생성규칙 $x \to y$가 $|x| \leq |y|$를 만족하면 G는 문맥민감 문법(context-sensitive grammar)이라고 부른다. (여기에서 $x \in (V \cup \Sigma)^* V (V \cup \Sigma)^*$이고 $y \in (V \cup \Sigma)^*$이다.)*

정의 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.

$$uAv \to uwv$$

여기에서 $w \neq \epsilon$이다. 이 생성규칙은 u와 v의 문맥에서 A가 w로 바뀔 수 있음을 나타낸다. Chomsky 표준형이 문맥민감 문법의 형태이므로 문맥무관 언어의 종류는 문맥민감 언어 종류의 부분집합이다.

예제 11 *$L = \{a^n b^n c^n : n \geq 1\}$은 예제 9의 문법에 의해 생성되므로 문맥민감 언어이다.*

위 예제의 의해 문맥무관 언어의 종류는 문맥민감 언어 종류의 진부분집합이다.

정의 11 *선형 오토마타는 입력 w에 대하여 테입의 $\#w\#$ 부분만을 사용하는 비결정 튜링기계이다.*

정리 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 M은 테입을 2개의 트랙으로 나누고 첫째 트랙에 입력 w를 보관하고 둘째 트랙에 G의 문장형태 u를 만든다. 문맥민감 문법의 문장형태의 길이는 단조증가하므로 $u \stackrel{*}{\Rightarrow} w$일 경우 $|u| \leq |w|$이므로 M은 테입의 $\#w\#$ 부분만을 사용하여 $w \in L(G)$인 경우 w를 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

그림 15:

이 외에 재귀 언어가 유형 0과 1 사이에 위치한다. 다섯 가지 언어 종류는 서로 진부분집합 관계를 가진다. 그림 16.

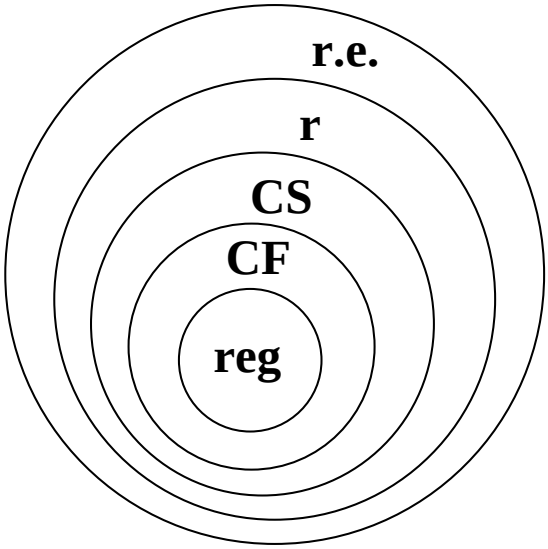

그림 16: