□特 輯□

형식언어와 오토마타

경성대학교 전산통계학과 김 성 권*

목 차

Ⅰ. 서 론

언어는 크게 두 가지로 나눌 수 있는데, 하나는 자연 언어(natural language)이고 다른 하나는 형식 언어(formal language)이다. 우리가 언어를 배우거나 다룰 때 문법을 생각하는데, 어떤 특정 언어에 대해서 언어와 문법의 관계를 보면, 한국어나 영어 같은 자연 언어는 언어가 먼저 자연 발생적으로 생기고 나서 나중에 그 언어를 설명하기 위한 도구로서 문법을 만들었는데 반해, 형식 언어의 경우는 언어 자체를 정의하기 위해 문법을 만든다. 따라서 형식 언어의 경우는 언어가 문법적으로 정확하게 정의되지만, 자연 언어는 그렇지 못하고 애매모호한 경우가 많다.

형식 언어를 정의하는데 사용되는 문법의 일반적인 구조는 아래와 같다. 문법은 4-튜플 (V, Σ, R, S)로 나타낼 수 있는데, 여기서,

V는 비단말(nonterminal)들의 유한 집합이고,

Σ는 단말(terminal)들의 유한집합이고,

R은 생성 규칙(production rule)들의 유한 집합이며,

S는 V의 원소로서 시작 비단말(start nonterminal)이다.

Σ는 언어에서 사용되는 심볼들을 나타내며, 흔히 그 언어의 알파벳이라 부른다. 이 언어에 속하는 모든 스트링들은 Σ에 있는 심볼들로 구성된다. R에 있는 각 규칙은 $\alpha \rightarrow \beta$의 형태로 이루어져 있는데, α와 β는 각각 $V \cup \Sigma$의 원소들의 결합으로 이루어져 있으며, α에는 반드시 비단말이 한 개 이상 들어 있어야 한다. 시작 비단말 S로부터 시작하여 생성 규칙의 왼쪽에 나오는 패턴을 생성 규칙의 오른쪽에 나오는 패턴으로 연속적으로 대치해 나가서 단말들로만 이루어진 스트링이 나오면, 이 문법이 그 스트링을 생성한다고 말한다. 또 어떤 문법 G가 생성할 수 있는 모든 스트링의 집합을 그 문법이 생성하는 언어라 한다.

예를 들어, V={S, T}, Σ={a, b, c},

R={S→abTSc,

S→λ,

bTa→abT,

* 종신회원

bTc→bc,
bTb→bbT]

인 문법은 $\{a^nb^nc^n \mid n\geq 0\}$을 생성한다. aabbcc는 L에 속하는데, S⇒abTSc⇒abTabTScc⇒abTabTcc⇒abTabcc⇒aabTbcc⇒aabbTcc⇒aabbcc를 통해서 생성된다.

α와 β에 여러 제약을 가함으로써 여러 개의 다른 계층의 언어를 정의할 수 있다. 이들에 대해서는 앞으로 설명하겠다.

형식 언어를 정의하는 다른 방법은 정의하고자 하는 언어에 속하는 스트링에 대해서 '예'라고 대답하고 그렇지 않은 스트링에 대해서는 '아니오'라고 대답하는 개념적인 기계를 설계하는 것이다. 이 기계는 스트링을 입력으로 받아서, 그것이 그 기계에 의해 정의되는 언어에 속하는지 아닌지를 판단한다. 이 기계를 흔히 오토마타(automata)라 부르는데, 기계가 어느 정도로 복잡하느냐에 따라 여러 계층의 언어를 정의한다. 이들 기계에 대해서도 앞으로 설명하겠다.

문법은 어떤 언어를 정의하여 그 언어에 속하는 스트링들을 생성한다는 관점에서 출발한 반면에, 오토마타는 어떤 스트링이 어떤 언어에 속하느냐 아니냐를 판단하는 관점에서 출발한 것이다. 두 가지 서로 다른 방법에서 출발했지만 한쪽 방법으로 정의되는 언어는 언제든지 다른 쪽 방법으로 동등하게 변환될 수 있다는 면에서 문법과 오토마타는 유사점이 많다.

본고에서는 여러 종류의 오토마타를 설명하고, 그 오토마타가 정의하는 언어와 같은 언어를 생성하는 문법들을 설명한다. 이런 내용들은 이미 많은 책에 나와 있는 것이지만 이를 다시 소개한다는 입장에서 본고에 정리하여 본다. II장에서는 오토마타 중에서 가장 간단한 구조를 갖는 유한 오토마타(finite automata)를 소개하고, 이와 동일한 언어를 생성하는 문법인 정규 문법(regular grammar)을 소개한다. III장에서는 푸시다운 오토마타(pushdown automata; PDA)를 소개하고 이와 동일한 언어를 생성하는 문맥 자유 문법(context-free grammar; CFG)을 소개한다. IV장에서는 우리가 생각할 수 있는 가장 일반적인 구조를 갖는 오토마타인 튜링 기계(Turing machine)와 역시 가장 일반적인 문법인 무제한 문법(unrestricted grammar)을 소개하여, 둘이 결국은 같은 언어를 표현하고 있음을 소개한다. 마지막 장에서는 간단히 결론을 맺는다.

II. 정규 언어와 유한 오토마타

2.1 유한 오토마타

유한 오토마타를 간단히 설명하면, 입력 테이프와 제어장치로 이루어진 개념적인 기계이다. 입력 테이프는 필요한 입력 데이타를 저장하고 있는데, 이는 왼쪽은 끝이 있지만 오른쪽으로는 끝이 없는 무한 테이프이다. 이 테이프는 셀(cell)로 나뉘어 있는데 입력 테이프의 맨 왼쪽 n개의 셀에는 n개의 입력 심볼이 각 셀에 하나씩 들어 있다. 또, 셀에 있는 심볼을 읽을 수 있는 입력 헤드가 테이프에 설치되어 있다. 제어 장치는 기계의 동작 과정 하나 하나를 제어하는 부분인데, 이를 위한 제어 정보가 기억되어 있으며, 기계의 현재 상태(state)를 나타내는 부분이 있는데, 여기에는 미리 정의된 여러 개의 상태 중 하나가 표시된다. 이 상태 중에는 기계가 시동될 때를 표시하는 시작 상태(start state)와 아래에서 설명할 채택 상태(accept state)를 포함하고 있다.

유한 오토마타는 다음과 같은 방법으로 동작한다. 먼저, 기계는 시작 상태에 들어가고, 입력 헤드는 테이프의 맨 왼쪽 셀에 위치한다. 헤드가 위치한 셀의 내용을 읽은 후, 기계의 상태가 무엇이냐에 따라 제어 장치는 기계의 상태를 변화시킨다. 그리고 헤드는 오른쪽으로 한 셀 움직인다. 따라서, 제어 장치에는 현재의 상태와 입력된 심볼의 내용에 따라 기계의 다음 상태를 결정할 수 있는 정보가 있어야 한다. 이런 식으로 심볼을 하나 읽고 상태를 변화시키는 과정을 반복하여 입력 테이프에 있는 마지막 심볼을 읽은 후, 기계의 상태가 채택 상태에 있으면 우리는 그 입력을 채택(accept)한다고 말하고, 만약 입력을 다 읽은 후에도 기계가 채택 상태에 있지 않으면 그 입력은 기각(reject)되었다고 말한다. 유한 오토마타는 제어 장치에 들어 있는 정보에 따라서 어떤 입력은 채택하고 어떤 입력은 기각한다.

유한 오토마타는 아래와 같이 5-튜플$(S, \Sigma, \delta, s_0, F)$로 정의할 수 있다. 여기서,

S는 기계가 택할 수 있는 상태들의 유한 집합이고,

Σ는 기계의 테이프에 사용하는 심볼들의 유한 집합으로서 알파벳이라 부르며,

δ는 $S\times\Sigma$에서는 S로 가는 함수로서 전이(transition) 함수라 부르며,

s_0는 S에 속하며, 시작 상태를 나타내며,

F는 S의 부분 집합으로서 채택 상태들의 집합이다.

δ를 자세히 설명하면, 이는 제어 장치에 기억되어 있는 정보로서 만약 $\delta(p, x)=q$이면 이는 기계가 상태 p에 있고 입력된 심볼이 x이면 기계의 다음 상태는 q가 된다는 것을 의미한다. 따라서 입력 스트링이 $x_1x_2\cdots x_n$일 경우 이 입력이 채택되기 위해서는 $s_n\in F$이고 각 i에 대해서 $\delta(s_{i-1}, x_i)=s_i$가 되는 상태 순서 $s_0, s_1, \cdots, s_n$이 있어야

한다.

Σ^*를 Σ에 들어 있는 심볼들을 임의로 결합하여 얻을 수 있는 스트링들의 집합이라 하자. 예를 들어 $\Sigma=\{a, b\}$이면 $\Sigma^*=\{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \cdots\}$이다. λ는 공 스트링(empty string)을 나타낸다. 이 때, Σ^*의 임의의 부분 집합을 Σ를 알파벳으로 하는 언어라 한다. $M=(S, \Sigma, \delta, s_0, F)$을 유한 오토마타라 하면, M이 채택하는 모든 스트링들의 집합은 Σ^*의 부분 집합이므로 Σ를 알파벳으로 하는 언어이며, 이를 L(M)이라 표시한다. 유한 오토마타 M에 대해서 L(M)이 되는 언어를 우리는 정규 언어(regular language)라 부른다. 즉, 주어진 Σ에 대해서 어떤 언어가 정규 언어라는 것을 증명하기 위해서는 그 언어를 채택하는 유한 오토마타를 만들어 보이면 된다.

그러면 모든 언어는 정규 언어인가? 물론 정규 언어가 아닌 언어들도 많다. 그 중 하나가 $L=\{a^n b^n | n\geq 0\}$이다. 이 언어에 속하는 스트링들은 먼저 a가 몇 개 나오고 이어서 같은 갯수의 b가 나오는 것이다. 이 언어를 채택하기 위해서는 a가 몇 개 나오는가를 세는 기능이 있어야 하는데 유한 오토마타에는 그런 기억 기능이 없다. 쉽게 생각하여, a가 하나이면 상태 s_1, 둘이면 s_2, …, i개 이면 s_i로 표시하면 될 것 같지만, S는 유한 집합이므로 a가 |S| 개 이상 들어 있는 스트링은 처리할 능력이 없다.

2.2 비결정적 유한 오토마타

지금까지 설명한 오토마타는 결정적(deterministic)이다. 현재의 상태가 p이고 입력 심볼이 x일 경우 기계가 갈 수 있는 다음 상태는 오직 하나라는 의미이다. 즉, 만약 $\delta(p, x)=q$이고 $\delta(p, x)=r$이면, 반드시 $q=r$이어야 한다는 것이다. 이 제한을 완화시키면 비결정적(nondeterministic) 유한 오토마타를 얻는다. 앞의 경우에 $q\neq r$일 수도 있다는 것이다. 좀 더 정확하게 설명하면, 비결정적 유한 오토마타$(S, \Sigma, \delta, s_0, F)$는 다른 모든 것을 결정적 오토마타의 경우와 같고, 다만 δ가 $S\times\Sigma$에서 P(S)로 가는 함수이다. P(S)는 S의 부분 집합들의 집합(power set)이다. 주어진 상태 p와 심볼 x에서 기계가 택할 수 있는 상태가 여러 개가 될 수 있다는 것이다. 예를 들어, $\delta(p, x)=\{q, r\}$이면 기계가 이 시점에서 택할 수 있는 상태가 q와 r, 둘이라는 의미이다. 이것은 q와 r 둘 중에 하나를 임의로 택하라는 의미가 아니라 q, r 두 가지를 모두 고려하라는 뜻이다. 비결정성(nondeterminism)과 임의성(randomness)은 전혀 의미가 다르므로 구별해야 한다. 입력 스트링이 $x_1x_2\cdots x_n$일 경우 이 입력이 비결정적 유한 오토마타에 의해 채택되기 위해서는 $s_n\in F$이고 각 i에 대해서 $s_i\in\delta(s_{i-1}, x_i)$가 되는 상태 순서 $s_0, s_1, \cdots, s_n$이 있어야 한다.

δ를 $S\times(\Sigma\cup\{\lambda\})$에서 P(S)로 가는 함수로 확대 정의하여, 비결정적 유한 오토마타가 입력 심볼을 읽지 않고, 단지 상태만 바꿀 수 있도록 만들 수 있다. 즉, $\delta(p, \lambda)=\{q\}$인 경우에는 현 상태 p에서 입력을 읽지 않고 상태만 q로 바꾼다는 것을 나타낸다. 이런 것을 λ-전이라 부르고, 그런 오토마타를 λ-전이를 갖춘 오토마타라 한다.

정의만 고려하면 비결정적인 오토마타가 더 광범위하여 결정적 오토마타보다 성능이 강할 걸로 예상되지만, 실제로는 모든 비결정적 유한 오토마타는 그와 성능이 똑 같은 결정적 오토마타로 변환될 수 있다. 따라서 비결정적 유한 오토마타와 결정적 유한 오토마타는 성능면에서는 전혀 차이가 없음이 증명되었다. 물론 유한 오토마타에 λ-전이를 허용한다하더라도 그로 인해 생기는 능력의 차이는 없다.

2.3 정규 문법

여기서는 유한 오토마타와 능력 면에서 동일한 문법을 소개한다. I장에서도 설명했듯이, 문법은 일반적으로 4-튜플(V, Σ, R, S)로 나타낼 수 있는데, R에 있는 각 규칙 $\alpha\rightarrow\beta$에서 α와 β에 제약을 가하여, 반드시 $|\alpha|=1$이고 β는 반드시 하나의 단말과 하나의 비단말로 이루어지거나, 하나의 단말로만 이루어지거나, 공 스트링으로 이루어지거나 하도록 하면 정규 문법(regular grammar)이 얻어진다. 즉, R에 속하는 각 생성 규칙이

$$A\rightarrow aB$$
$$A\rightarrow a$$
$$A\rightarrow\lambda$$

중의 하나의 형태가 되도록 하면 정규 문법을 얻는다. 여기서 A, $B\in V$이고 $a\in\Sigma$이다.

모든 정규 문법 G에 대해서 L(G)와 동일한 언어를 채택하는 유한 오토마타를 만들 수 있고, 역으로 모든 유한 오토마타 M에 대해서는 L(M)과 동일한 언어를 생성하는 정규 문법을 만들 수 있다. 따라서 정규 문법과 유한 오토마타 모두 정규 언어를 표현하는 데 사용하지만, 다만 그 접근 방법이 다르다고 말할 수 있다.

2.4 정규식

L_1과 L_2가 모두 정규 언어이면 합집합 $L_1\cup L_2$도 정규

언어이고, 연결(concatenation) $L_1 \cdot L_2$도 역시 정규 언어이다. $L_1 \cdot L_2$는 L_1에 속하는 각 스트링을 L_2에 속하는 각 스트링과 연결하여 얻어지는 언어이다. $L_1=\{a, ab\}$이고 $L_2=\{aa, b\}$이면, $L_1 \cdot L_2=\{aaa, ab, abaa, abb\}$이다. 또, L이 정규 언어일 경우, 클리네 스타(Kleene star) L^*도 역시 정규 언어이다. L^*는 L에 속하는 스트링을 0개 이상 연결하여 얻어지는 언어이다. 당연히 $\lambda \in L^*$이다. 합집합, 연결, 클리네 스타, 이 세 연산이 있으면 우리는 얼마든지 새로운 정규언어를 만들 수 있다.

주어진 알파벳 Σ에 대해서 정규식은 아래와 같이 정의된다. 원소 $a \in \Sigma$에 대해서 $\mathbf{a}=\{a\}$라 하자.

(1) ϕ는 정규식이다.

(2) Σ의 각 원소 a에 대해서 **a**는 정규식이다.

(3) 만약 p와 q가 정규식이면, $p \cup q$도 정규식이다.

(4) 만약 p와 q가 정규식이면, $p \cdot q$도 정규식이다.

(5) 만약 q가 정규식이면 p^*도 정규식이다.

(6) 만약 p가 정규식이면 (p)도 정규식이다.

예를 들어, $\Sigma=\{a, b, c\}$이면, $\mathbf{a} \cup (\mathbf{b} \cdot \mathbf{c})$은 정규식으로서, 정규 언어 {a, bc}를 나타낸다. 따라서, 하나의 정규식은 하나의 정규 언어를 표현한다.

모든 정규식에 대해서 그것이 표현하는 언어를 채택하는 유한 오토마타를 만들 수 있고, 반대로 모든 유한 오토마타에 대해서 그것이 채택하는 언어를 표현할 수 있는 정규식이 있다. 따라서 정규식은 정규 언어를 표현하는 또 다른 방법이며, 지금까지 설명한 결정적 유한 오토마타, 비결정적 유한 오토마타, 정규 문법, 정규식은 정규 언어를 표현하는 서로 다른 네 방법이지만, 모두 정규 언어를 표현한다는 데 있어서는 동일하다.

2.5 정규 언어의 성질

앞에서도 언급했지만, L_1과 L_2가 모두 정규 언어이면 $L_1 \cup L_2$과 $L_1 \cdot L_2$도 정규 언어이다. 또, L이 정규 언어이면 L^+도 정규 언어이고, 보집합 $\Sigma^* - L$도 정규 언어이다. 각 정규 언어에 해당하는 유한 오토마타를 적당히 조합하면, 결과에 해당하는 정규 언어를 채택하는 유한 오토마타를 만들 수 있다.

주어진 유한 오토마타에 대해서 그 오토마타가 채택하는 언어가 공집합인지, 유한 집합인지, 아니면 무한 집합인지를 알아내는 알고리즘이 있다. 또, 두 개의 유한 오토마타가 있을 때, 이 둘이 채택하는 언어가 동일한지 아닌지도 알아낼 수 있다. 또 어떤 정규 언어를 채택하는 유한 오토마타가 있으면 똑 같은 언어를 채택하면서 상태의 수가 가장 작은 오토마타로 항상 변환시킬 수 있다.

III. 문맥 자유 언어와 푸시다운 오토마타

3.1 푸시다운 오토마타

푸시다운 오토마타(PDA)는 구조 면에서는 유한 오토마타와 거의 동일하나 차이점은 PDA는 기억 장치로 스택을 갖고 있다는 점이다. 스택이 있기 때문에 여기에 필요한 정보를 기억시키고 나중에 이를 다시 이용할 수 있다. 스택의 성질 때문에 스택에 대한 자료의 입출력은 반드시 스택의 top을 통해서만 이루어진다는 점을 유의하기 바란다.

PDA의 동작 과정은 입력 테이프에서 심볼을 하나 읽고, 스택에서 하나의 심볼을 읽은(pop) 후, 기계의 상태를 바꾸고, 스택에 하나의 심볼을 쓴다(push). 즉, 현재의 상태, 입력 심볼, 그리고 스택 심볼에 따라 기계의 다음 상태와 스택에 기억시킬 심볼을 결정한다. 테이프에서 심볼을 읽은 후에는 헤드는 오른쪽으로 한 셀 움직인다.

PDA는 아래와 같이 6-튜플 $(S, \Sigma, \Gamma, \delta, s_0, F)$로 정의할 수 있다. 여기서,

S는 기계가 택할 수 있는 상태를 나타내는 유한 집합이고,

Σ는 입력 테이프에 사용하는 심볼들의 집합이고,

Γ는 스택에 사용하는 심볼들의 집합이며,

δ는 $S \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})$에서 $P(S \times (\Gamma \cup \{\lambda\}))$로 가는 함수이고,

s_0는 S에 속하며, 시작 상태를 나타내며,

F는 S의 부분 집합으로서 채택 상태를 나타낸다.

여기서 δ를 자세히 보면 위에서 정의한 PDA는 비결정적임을 알 수 있다. 나중에 설명하겠지만 유한 오토마타의 경우와 달리, PDA에서는 결정적 PDA와 비결정적 PDA는 성능이 서로 다르므로 양자를 구별할 필요가 있다. 그래서 그냥 PDA라 함은 비결정적 PDA를 의미하고, 결정적 PDA는 DPDA(deterministic PDA)라 한다. 또, 입력 심볼이나 스택 심볼 자리에 λ가 올 수 있도록 하여 입력이나 스택을 사용하지 않고도 PDA가 움직일 수 있도록 λ-전이를 갖추고 있다. 예를 들어, $\delta(p, \lambda, \lambda)=\{(q, \lambda)\}$는 입력이나 스택을 전혀 사용하지 않고 상태만 p에서 q로 바꾸는 것이며, $\delta(p, \lambda, c)=\{(q, \lambda)\}$는 스택에서 c를 읽고 상태를 바꾸는 것이며, $\delta(p, \lambda, \lambda)=\{(q, c)\}$는 스택에 c를 저장하고 상태를 바꾸는 것이다.

주어진 입력에 대해서 PDA가 시작 상태에 있고, 스택은 비어 있으며, 헤드가 테이프의 맨 왼쪽 셀에 있는

상황에서 시동하여, PDA가 입력을 모두 읽은 후, PDA가 채택 상태에 있으면, 그 입력은 PDA에 의해서 채택되었다고 말한다. M이 PDA일 때, M에 의해서 채택되는 모든 입력 스트링들의 집합을 L(M)이라 표시한다. PDA에 의해서 채택되는 모든 언어를 문맥 자유 언어(context-free language; CFL)라 한다. M에서 스택을 전혀 사용하지 않으면 M은 바로 유한 오토마타와 기능이 꼭 같으므로, 모든 정규 언어는 PDA에 의해 채택될 수 있고, 따라서 CFL이다.

앞에서 정규 언어가 아니라고 언급했던 $L=\{a^nb^n \mid n\geq 0\}$은 PDA에 의해서 채택될 수 있다. 입력이 주어졌을 때, 먼저 스택에 #를 저장한다. 이는 스택의 바닥을 표시하기 위함이다. 그 다음에는 a를 하나씩 읽어서 스택에 저장한다. a들이 끝나고, b가 시작되면 b를 하나 읽을 때마다 a를 하나 스택에서 빼낸다. 이렇게 하여 b를 모두 읽었을 때, 스택에 남아 있는 것이 #뿐이면 우리는 그 입력을 채택한다. 이 과정 중에 하나라도 계획대로 실행되지 않으면 그 입력은 기각한다. 약간 더 복잡한 언어 $L=\{a^nb^nc^n \mid n\geq 0\}$은 CFL이 아니다. 이는 자세하게 증명할 수 있지만, 스택의 성질상 항상 top을 통해서만 입출력이 가능하므로 a의 수와 b의 수를 비교하고, 또 이를 c의 수와 비교하는 PDA는 제작이 불가능하다.

3.2 문맥 자유 문법

본절에서는 앞절에서 정의한 PDA와 동일한 언어를 표현하는 문법인 문맥 자유 문법(CFG)을 소개한다. 2.3절의 정규 문법과 거의 동일한데, 차이는 생성 규칙이 $A\rightarrow\beta$의 형태로 되어 있다는 것이다. A는 비단말이고, β는 단말과 비단말의 조합으로서 길이에는 제한이 없으며, λ가 될 수도 있다. $L=\{a^nb^n \mid n\geq 0\}$은 CFG

$$S\rightarrow aSb$$
$$S\rightarrow\lambda$$

에 의해서 생성될 수 있음을 쉽게 확인할 수 있다. 예를 들어 aaabbb는 L에 속하는데, 이는 $S\Rightarrow aSb\Rightarrow aaSbb\Rightarrow aaaSbbb\Rightarrow aaabbb$를 통해서 생성된다.

모든 CFG에 대해서 그것이 생성하는 언어를 채택하는 PDA를 만들 수 있으며, 반대로 모든 PDA에 대해서도 그것이 채택하는 언어를 생성하는 CFG를 만들 수 있다. 따라서 CFG와 PDA 모두 CFL을 표현하는 데 사용하지만 접근 방법이 다를 뿐이다.

3.3 결정적 푸시다운 오토마타

앞에서 설명한 PDA는 비결정적으로 동작한다. 유한 오토마타의 경우에는 결정적인 경우와 비결정적인 경우 모두 동일한 언어, 정규 언어를 채택하므로 두 가지를 굳이 구별할 필요가 없었다. 그러나 PDA는 경우가 다르다. 결정적 PDA(DPDA)는 PDA의 정의 부분에서 δ를 $S\times(\Sigma\cup\{\lambda\})\times(\Gamma\cup\{\lambda\})$에서 $S\times(\Gamma\cup\{\lambda\})$로 가는 함수로 다시 정의함으로써 얻어지는데, DPDA는 현재의 상태, 입력 심볼과 스택의 심볼에 따라서 DPDA의 다음 상태와 스택에 입력할 심볼이 유일하게 정해진다.

이렇게 DPDA를 정의한 후 생길 수 있는 의문은, 과연 모든 CFL에는 그것을 채택하는 DPDA를 만들 수 있는가 하는 것이다. 불행하게도, $L=\{a^nb^n \mid n\geq 0\}\cup\{a^nb^{2n} \mid n\geq 0\}$은 PDA에 의해서는 채택되지만 DPDA에 의해서는 채택되지 않는 언어이다. 입력에서 a들을 모두 읽어 스택에 입력한 후, b들이 시작될 때, 스택에 들어 있는 a 하나에 b를 한 개씩 대응시켜야 하는지 두개씩 대응시켜야 하는지를 정해야 하는데 이것을 DPDA는 결정할 수 없다. 즉, PDA와 DPDA 사이에는 능력의 차가 있다는 것이다. 그래서 DPDA가 채택하는 언어들을 DCFL(deterministic CFL)이라 하고, DCFL은 CFL의 진부분 집합이다.

3.4 문맥 자유 언어의 성질

L_1과 L_2가 CFL이면 $L_1\cup L_2$도 CFL이다. 그러나 이들의 교집합 $L_1\cap L_2$는 CFL이 아닐 수 있다. 예를 들어, $L_1=\{a^nb^nc^m \mid n\geq 0,\ m\geq 0\}$과 $L_2=\{a^mb^nc^n \mid n\geq 0.\ m\geq 0\}$ 모두 CFL이나, $L_1\cap L_2=\{a^nb^nc^n \mid n\geq 0\}$은 CFL이 아니다. L이 CFL이면 L^*은 CFL이나, 보집합 Σ^*-L은 CFL이 아니다. 교집합과 합집합은 드모르간의 법칙에 의해서 보집합을 이용하면 서로 표시 가능한데, 합집합은 CFL이고 교집합은 CFL이 아니므로 당연히 보집합도 CFL이 아니다. 그러나 L이 CFL이고 R이 정규 언어이면 $L\cap R$은 CFL이다.

CFG에 의해 표현되는 CFL이 있을 때, 이것이 공집합인지, 유한 집합인지, 아니면 무한 집합인지를 알아내는 알고리즘이 존재한다. 그러나 두 개의 CFG가 있을 때, 이 둘이 동일한 언어를 생성하는지를 알아내는 알고리즘은 없다.

3.5 결정적 문맥 자유 언어의 성질

DCFL은 여러 면에서 CFL과는 다른 성질을 갖고 있다. L_1과 L_2가 DCFL이더라도 $L_1\cup L_2$과 $L_1\cdot L_2$는 DCFL이 아니다. 예를 들어 $L_1=\{a^nb^n \mid n\geq 0\}$와 $L_2=\{a^nb^{2n} \mid n\geq 0\}$은

모두 DCFL이나, $L_1 \cup L_2$은 DCFL이 아니라는 것은 3.3절에서 언급했다. L이 DCFL이더라도 L*는 DCFL이 아니지만, 보집합 $\Sigma^* - L$은 DCFL이다. 두 개의 DPDA가 동일한 언어를 생성하는지를 알아내는 알고리즘이 있는지 없는지는 아직 증명되지 않은 중요한 미해결 문제이다.

IV. 튜링 기계

4.1 튜링 채택 언어

II, III장에서 사용한 여러 기계들과 마찬가지로 튜링 기계도 테이프를 갖고 있는데, 한쪽으로만 무한이며, 유한 오토마타나 PDA와는 달리 테이프에서 입력하는 것뿐 아니라 테이프에 출력을 할 수도 있다. 즉, 테이프 헤드가 입출력 겸용이라는 것을 뜻한다. 따라서 테이프에 입출력할 수 있는 테이프 심볼들이 필요하고, 테이프에 초기 데이타를 기억시켜 놓기 위해 필요한 심볼들인 기계의 알파벳이 필요하다.

튜링 기계 역시 제어 장치를 갖고 있으며, 이 제어 장치는 기계의 현 상태를 가르키는데 이 중에는 시작 상태와 채택 상태가 있다. 시작 상태는 다른 기계와 마찬가지로 기계가 시동할 때의 상태이며, 채택 상태는 튜링 기계가 동작 중 여기에 이르면 기계는 입력 스트링을 채택한다.

시동시 기계는 시작 상태에 있으며, 헤드는 테이프의 맨 왼쪽 셀 위에 놓이고, 입력 스트링은 테이프의 맨 왼쪽 셀들에 저장되어 있다. 튜링 기계의 동작은 먼저 테이프에서 심볼을 하나 입력하면 현재의 상태에 따라서 기계의 다음 상태가 결정되고, 테이프에 심볼 하나를 출력할 것인가 아니면 헤드가 한 셀 만큼 왼쪽이나 오른쪽으로 이동할 것인가를 정한다. 이런 식으로 동작하여 기계가 채택 상태에 이르면 그 스트링은 채택된다. 일반적으로 스트링을 채택하면 기계는 동작을 멈춘다고 가정한다. 따라서 채택되지 않는 스트링에 대해서는 기계에 따라 멈출 수도 있고, 멈추지 않고 계속 동작할 수 있다.

튜링 기계는 아래와 같이 6-튜플 $(S, \Sigma, \Gamma, \delta, s_0, F)$로 정의할 수 있다. 여기서,

S는 기계가 택할 수 있는 상태를 나타내는 유한 집합이고,

Σ는 테이프에 입력 자료를 기억시키는 데 사용하는 심볼들의 유한 집합으로서 알파벳이라 부르며,

Γ는 테이프에 사용하는 심볼들의 집합이며, $\Sigma \subseteq \Gamma$이고,

δ는 $S \times \Gamma$에서 $S \times (\Gamma \cup \{L, R\})$로 가는 함수이고,

s_0는 S에 속하며, 시작 상태를 나타내며,

F는 S의 부분 집합으로 채택 상태들의 집합을 나타낸다.

δ에서 $\delta(p, x) = (q, y)$는 현 상태가 p이고 테이프에서 입력되는 심볼이 x이면 다음 상태는 q가 되고 x 대신 y를 테이프에 기억시킨다는 것을 의미하고, $\delta(p, x) = (q, L)$는 앞의 경우와 똑 같은데 y를 기억시키는 것이 아니라 헤드를 왼쪽으로 한 셀 옮긴다는 것을 의미하고, $\delta(p, x) = (q, R)$ 는 헤드를 오른쪽으로 한 셀 옮긴다는 것을 의미한다. 여기에서 정의하는 튜링 기계는 다른 데에서 사용되는 정의와 다를 수 있으나 전체적으로 보면 차이가 없다는 점에 유의하기 바란다.

튜링 기계의 중요성은 아래와 같이 표현하는데 이를 흔히 Church Turing's thesis라 부른다. 튜링 기계는 우리가 생각할 수 있는 어떤 계산 시스템과도 동등한 계산 능력을 갖는다. 이는 물론 아직 증명은 확실히 되고 있지 않지만 거의 모든 사람들이 그렇게 믿고, 또한 그럴 가능성이 아주 높기 때문에 가정이라 하지 않고 thesis라 한다. 예로서 알고리즘이나 계산 이론에서 많이 사용하는 RAM(random access machine)과 튜링 기계는 능력면에서 동일함이 증명되었다. 또, 튜링 기계는 여러 면에서 확장 가능한데, 테이프를 양쪽으로 무한인 것으로 대체할 수도 있고, 테이프의 수를 하나에서 여러 개로 늘릴 수도 있으며, 비결정성을 도입하여 비결정적 튜링 기계를 만들 수도 있다. 이런 것을 도입하더라도 튜링 기계의 능력은 증가하지 않는다는 것은 이미 증명되어 있으며, 이는 앞의 thesis에 비추어 당연하다.

튜링 기계 M에 의해서 채택된 언어를 L(M)이라 표시하고, 이런 언어를 튜링 채택(Turing acceptable) 언어라 부른다. 이를 r.e.(recursively enumerable) 집합이라 부르기도 한다. 튜링 채택 언어 중의 하나가 $L = \{a^n b^n c^n \mid n \geq 0\}$이다. 이는 CFL은 아니다. 그러면 튜링 채택 언어를 생성하는 문법은 무엇인가하는 의문점은 다음 절에서 다룬다.

4.2 무제한 문법

문법 중에서 가장 일반적인 것으로 무제한(unrestricted) 문법이 있다. 이는 type 0 문법, phrase-structure 문법 등으로 불리기도 한다. 무제한 문법 역시 4-튜플 (V, Σ, R, S)로 나타낼 수 있는데, R에 있는 각 규칙 $\alpha \rightarrow \beta$에 대해, α와 β는 각각 $V \cup \Sigma$의 원소들의 결합으로 이루어져 있고, α에는 반드시 비단말이 적어도 하나 이상

들어 있어야 한다는 것 외에는 다른 제한이 없다. I장에서 예를 든 문법이 무제한 문법이면서, CFL이 아닌 $\{a^nb^nc^n \mid n\geq 0\}$을 채택한다.

모든 무제한 문법에 대해서 그것이 생성하는 것과 동일한 언어를 채택하는 튜링 기계를 만들 수 있고, 반대로 모든 튜링 기계에 대해서 그것이 채택하는 언어와 동일한 언어를 생성하는 무제한 문법을 만들 수 있다. 따라서, 무제한 문법과 튜링 기계는 튜링 채택 언어를 표현하는 서로 다른 방법이다.

4.3 튜링 기계의 한계

그러면 무제한 문법으로 생성할 수 없는 언어는 존재하는가? 다시 말해서, 튜링 기계로서 채택할 수 없는 언어가 있는가라는 의문점이 생긴다. 이 의문점을 해결하기 위해 앞의 튜링 기계 정의를 다시 한번 보면, $M=(S, \Sigma, \Gamma, \delta, s_0, F)$에서 S, Σ, Γ 모두 유한 집합이고, 따라서 δ도 유한이다. 그러므로 Σ나 Γ에 속하는 심볼들은 ASCII 코드처럼 이진 코드화할 수 있다. 또 S 역시 유한이므로 S의 상태들도 코드화할 수 있다. 마찬가지로, δ의 각 규칙은 S, Σ, Γ의 코드를 이용하여 표시할 수 있으므로 (즉, 비트 스트링으로 표시 가능하므로), δ의 모든 내용도 역시 비트 스트링으로 표시 가능하다. 따라서 튜링 기계의 모든 것을 쉽게 하나의 아주 긴 비트 스트링으로 나타낼 수 있다. 이를 M의 인코딩(encoding)이라 부른다. 이것은 프로그래밍 언어로 쓰여진 어떤 프로그램이더라도 프로그램에 쓰인 각 글자를 ASCII 코드로 변환시키면 그 프로그램은 하나의 긴 비트 스트링으로 나타낼 수 있다는 것과 같은 맥락에서 이해하면 된다.

그래서 인코딩 w로 표시될 수 있는 튜링 기계를 앞으로는 M_w라고 표시한다. 언어 $L_0=\{w \mid M_w$가 w를 채택하지 않는다$\}$를 정의하자. 즉, 어떤 스트링 w가 L_0에 속할 필요충분 조건은 기계 M_w가 입력 w를 채택하지 않는다는 것이다. 그러면 L_0는 튜링 채택 언어가 아니다. 이를 증명하기 위해 L_0가 어떤 기계 M_0에 의해 채택된다고 가정하자. 즉, $L_0=L(M_0)$이다. M_0의 인코딩을 w_0라 하자. 문제는 w_0는 L_0에 속하는가 아닌가? L_0의 정의에 의해서 $w_0\in L_0$이면 $M_{w0}=M_0$가 w_0를 채택하지 않으므로 $w_0 \notin L_0$이고, $w_0 \notin L_0$이면 M_{w0}가 w_0를 채택하므로 $w_0\in L_0$이다. 두 경우를 종합하면 모순이 생기므로 L_0가 튜링 채택 언어라고한 가정이 잘못된 것이다. 따라서 L_0를 채택하는 튜링 기계는 존재하지 않는다.

4.4 튜링 결정 언어

L_0의 보집합, $L_1=\{w \mid M_w$가 w를 채택한다$\}$는 어떤가? L_1은 튜링 채택 언어이다. L_1을 채택하는 튜링 기계는 w를 입력으로 받아서 w를 인코딩으로 갖는 M_w에 입력하여 M_w가 w를 채택하면 w를 채택한다.

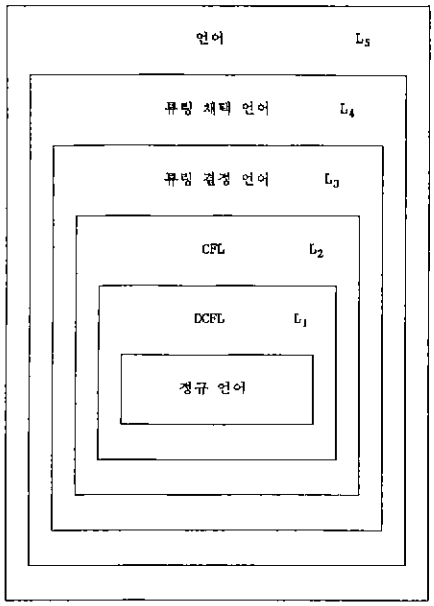
어떤 언어 L이 튜링 채택 언어이고, 그의 보집합 역시 튜링 채택 언어이면 우리는 L을 튜링 결정(Turing decidable) 언어라 부른다. 튜링 결정 언어는 흔히 recursive 언어라 하기도 한다. 튜링 결정 언어의 경우는 어떤 입력이 그 언어에 속하는지 아닌지를 정확하게 결정할 수 있는 반면에, 튜링 채택 언어는 그 언어에 속하는 입력에 대해서는 정확히 채택하지만 속하지 않는 것에 대해서는 잘 알 수가 없다. 튜링 기계를 가지고 이야기하면 튜링 결정 언어의 경우, 그 언어에 속하는 입력에 대해서는 '예'라고 답하고 속하지 않는 입력에 대해서는 '아니오'라고 답하는 기계를 만들 수 있다는 것이고, 튜링 채택 언어의 경우는 그 언어에 속하는 입력에 대해서는 '예'라고 답하지만 속하지 않는 입력에 대해서는 기계가 무한 루프에 빠져 정확히 '아니오'라는 답을 주지 못할 수도 있다는 것이다. 위의 L_1은 튜링 채택이지만 튜링 결정 언어는 아니다. 언어 L이 튜링 결정 언어이면 L을 채택하는 튜링 기계는 어떤 입력에 대해서도 그것이 L에 속하면 '예'라 답하고, 속하지 않으며 '아니오'라고 답한 후, 동작을 멈춘다. 이 개념은 알고리즘의 특성인 '알고리즘은 항상 끝나야 한다'는 유한성과 일맥 상통하며, 실제로 튜링 결정 언어들의 집합과 알고리즘에 의해 결정되는 문제들의 집합은 동등하다.

V. 결 론

본고에서는 여러 계층의 언어들을 살펴 보았고, 또 각 언어를 생성하는 문법과 그것을 채택하는 오토마타들에 대해서 살펴보았다. 이 언어 계층은 아래 그림 1과 같이 나타낼 수 있다. 각 계층은 그 아래 계층을 완전 포함한다. 그림 1에서 L_i는 계층 i에 속하는 언어 중에서 계층 i−1에는 속하지 않는 언어를 나타낸다. 각 L_i는 다음과 같다.

$L_1=\{a^nb^n \mid n\geq 0\}$
$L_2=\{a^nb^n \mid n\geq 0\}\cup\{a^nb^{2n} \mid n\geq 0\}$
$L_3=\{a^nb^nc^n \mid n\geq 0\}$
$L_4=\{w \mid M_w$가 w를 채택한다$\}$
$L_5=\{w \mid M_w$가 w를 채택하지 않는다$\}$

CFL과 튜링 결정 언어 사이에는 CSL(context-sensitive language)이라는 언어 계층이 더 있는데, 별로 흥



(그림 1) 언어 계층

미를 끌지 못하는 부분이므로, 여기서 간단히 소개한다. CSG(context-sensitive grammar)는 각 생성 규칙 $\alpha \rightarrow \beta$ 에서 $|\alpha| \leq |\beta|$ 라는 제약 조건만 붙는다. 또, CSL을 채택하는 오토마타로 LBA(linear bounded automata)가 있는데, 이는 튜링 기계와 모든 것이 동일한데, 테이프가 유한이라는 점만 다르다. 즉, 테이프에 입력 스트링이 원래 쓰여진 부분에만 헤드가 입출력을 할 수 있다.

유한 오토마타는 컴파일러의 첫 구성 요소인 lexical analyzer를 만드는 데 직접 응용될 수 있으며, DCFL과 CFL은 프로그래밍 언어의 문법을 기술하는 데 사용되고, parser를 작성하는 데는 DPDA의 변형인 LL(k)나 LR(k) parser 기법이 사용되고 있다. 튜링 기계는 Church Turing' thesis에 나타나 있듯이 전산학 분야에서 매우 중요한 위치를 차지하고 있으며, 특히 튜링 결정 언어는 우리가 알고리즘으로 해결할 수 있는 문제들의 범위와 동등하다는 면에서 중요하다.

참 고 문 헌

본고의 성질상 개별적인 논문보다 책을 많이 참고하여 작성하였다. 전체적인 줄거리는 [1]을 따라 전개했으며, 중간 중간 [2]를 참조했다.

1. J. G. Brookshear, *Formal Languages, Automata, and Complexity,* Redwood City, California: Benjamin/Cummings Publishing, 1989.
2. J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Menlo Park, California: Addison-Wesley, 1979.

김 성 권

1981 서울대학교 계산통계학과 졸업(이학사)
1983 한국과학기술원 전산학과 졸업(이학박사)
1990 미국 위싱턴대학교(시애틀) 전산학과 졸업(공학박사)
1983 ~1985 목포대학교 전산통계학과 교수재직
1991 ~현재 경성대학교 전산통계학과 교수재직중
관심 분야 : 컴퓨터 이론, 알고리즘