제5장 문맥 자유 문법(CFG)

5.1 서론

- 정규문법 : token 인식에 적합, 언어의 syntax 기술하기에는 부적합

- CFG(type 2) : $A \rightarrow \alpha$

- P.L.의 구조는 CFG로 기술하는 장점
  ① 간단하고 이해하기가 쉽다
  ② 문법으로부터 자동적으로 인식기를 구현 가능
  ③ 입력된 program의 구조를 생성규칙에 의해 분석

- 문법기호들의 notational convention

  1. Terminal symbol
     (1) *a, b, c* 등 알파벳 시작부분의 소문자와 숫자 0, 1, 2, …
     (2) +, - 연산자와 semi-colon, comma, 괄호 등 delimiter
     (3) 'if', 'then'과 같이 인용부호 사이에 표기된 문법기호
  2. Nonterminal symbol
     (1) *A, B, C* 등 알파벳 시작부분의 대문자
     (2) *S*는 보통 시작기호로 사용
     (3) <stmt>, <expr>과 같이 <와 >로 묶어서 나타낸 문법기호
  3. 만약 아무런 언급이 없으면 첫번째 생성규칙의 LHS 기호가 시작기호임.
  4. $A \rightarrow \alpha_1$, $A \rightarrow \alpha_2$, …, $A \rightarrow \alpha_k$와 같이 생성규칙의 LHS가 모두 *A*인 경우에 $A \rightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_k$로 간단히 표기할 수 있다.
     (A에 대한 택일(alternation) 규칙)

- 형식언어 이론에서 사용하는 각 기호들의 의미

  1. *X, Y, Z*와 같은 알파벳 끝부분의 대문자는 보통 문법기호, 즉 nonterminal 혹은 terminal 기호를 나타낸다($X, Y, Z \in V$).
  2. 알파벳 끝부분의 소문자, 주로 *u, v,* …, *z*는 terminal들로 이루어진 string을 나타낸다. ω 역시 terminal string을 나타내는 대표적인 문자이다($\omega \in V^*$).

3. α,β,γ와 같은 그리스체 소문자는 문법기호로 구성된 string을 나타낸다 ($\alpha,\beta,\gamma \in V^*$).

예1) 일반적인 표기법에 따라 문법기호를 분류해 보자.

$E \rightarrow EOE \mid (E) \mid -E \mid a$
$O \rightarrow + \mid - \mid * \mid /$

E, O : nonterminal
E : start symbol
그외 : terminal symbol
생성규칙의 수는 8개

예2) <if_statement> → 'if' <condition> 'then' <statement>에서 <>안에 기술된 기호는 nonterminal이고 '과 '사이에 기술된 기호는 terminal이다.

5.2 유도와 유도트리(derivation tree)

5.2.1 유도(derivation)

• CFG에서 문장의 생성 : 유도 과정

sentential form의 string에서 생성규칙을 반복 적용하여 nonterminal을 확장

예3) 산술식을 나타내는 문법

$E \rightarrow E+E \mid E*E \mid (E) \mid -E \mid a$

$-(a)$의 유도 : $E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(a)$

※ CFG에서는 생성규칙의 RHS에 nonterminal이 여러 개 올 수 있으므로 같은 문장을 유도하는 과정이 여러 개 있을 수 있다. 즉, 문장형태에서 유도시 대치해야 할 nonterminal을 선택하는데 여러 가지 경우가 있을 수 있다.

[정의 5.1] 좌단유도와 우단유도

- 좌단유도(leftmost derivation)

: 유도 과정의 각 단계에서 sentential form의 가장 왼쪽에 있는 nonterminal을 대치. $\overset{lm}{\Rightarrow}$으로 표기.

- 우단유도(rightmost derivation)

: 유도 과정의 각 단계에서 sentential form의 가장 오른쪽에 있는 nonterminal 을 대치. $\overset{rm}{\Rightarrow}$으로 표기.

※ 좌단유도에 의한 sentential form을 left-sentential form,
우단유도에 의한 sentential form을 right-sentential form이라 한다.

예4) 문장 (a+a)가 예3의 문법에 의해 유도되는 과정

$E \rightarrow E+E \mid E*E \mid (E) \mid -E \mid a$

(1) 좌단유도

E ⇒ (E) ⇒ (E+E) ⇒ (a+E) ⇒ (a+a).

(2) 우단유도

E ⇒ (E) ⇒ (E+E) ⇒ (E+a) ⇒ (a+a).

[정의 5.2] left parse와 right parse

left parse -- 좌단유도에서 적용된 생성규칙의 순서

right parse -- 우단유도에서 적용된 생성규칙의 **역순**

예5) 문장 a+a*a의 left parse와 right parse를 구하시오.

1. $E \rightarrow E+T$ 2. $E \rightarrow T$
3. $T \rightarrow T*F$ 4. $T \rightarrow F$
5. $F \rightarrow (E)$ 6. $F \rightarrow a$

left parse: 1 2 4 6 3 4 6 6.

$$
\begin{aligned}
E &\Rightarrow E+T \quad (1)\\
&\Rightarrow T+T \quad (2)\\
&\Rightarrow F+T \quad (4)\\
&\Rightarrow a+T \quad (6)\\
&\Rightarrow a+T*F \quad (3)\\
&\Rightarrow a+F*F \quad (4)\\
&\Rightarrow a+a*F \quad (6)\\
&\Rightarrow a+a*a \quad (6)
\end{aligned}
$$

right parse: 6 4 2 6 4 6 3 1.

$$
\begin{aligned}
E &\Rightarrow E+T \quad (1)\\
&\Rightarrow E+T*F \quad (3)\\
&\Rightarrow E+T*a \quad (6)\\
&\Rightarrow E+F*a \quad (4)\\
&\Rightarrow E+a*a \quad (6)\\
&\Rightarrow T+a*a \quad (2)\\
&\Rightarrow F+a*a \quad (4)\\
&\Rightarrow a+a*a \quad (6)
\end{aligned}
$$

5.2.2 유도트리(derivation tree)

- 유도 트리

  - 문장이 유도되는 과정을 tree형태로 표현한 것.
  - 생성규칙에 의해 적용되는 문장의 계층적 구조를 나타냄.

[정의 5.3] CFG G=(V_N, V_T, P, S)에 대한 유도 tree

1. 각 node는 문법기호를 label로 갖는다.
2. root label은 시작기호이다.
3. 어떤 node가 하나 이상의 children node를 가지면, 이 node는 nonterminal symbol을 label로 가진다.
4. 왼쪽부터 순서대로 label이 $A_1A_2\cdots A_k$인 node들이 어떤 node A의 children node라면, 생성규칙 $A \rightarrow A_1A_2\cdots A_k$가 존재한다.

예) 어떤 유도 과정에서 생성규칙 $A \rightarrow XYZ$가 적용되었다면 유도 tree는 다음과 같은 subtree를 갖는다.

$$\begin{tikzcd}[column sep=small] & A \arrow[dl] \arrow[d] \arrow[dr] & \\ X & Y & Z \end{tikzcd}$$

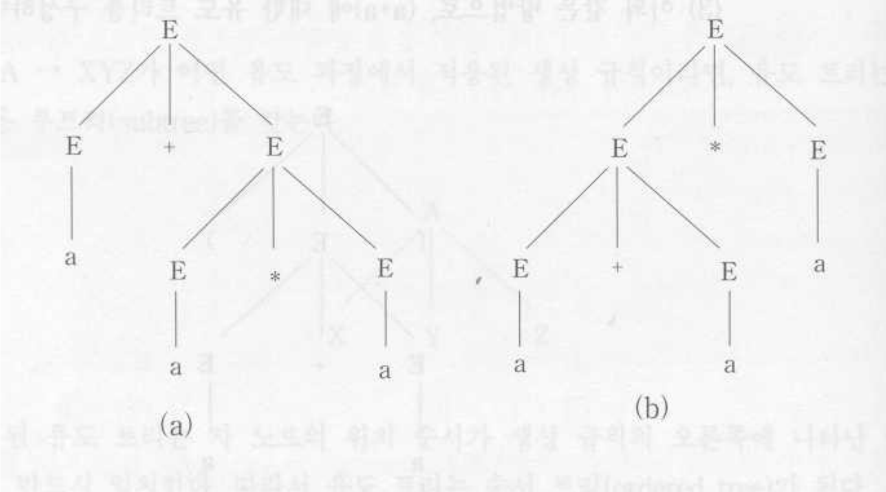
※ 유도 tree는 children node의 위치 순서가 생성규칙의 오른쪽에 나타난 symbol의 순서와 반드시 일치한다. 따라서 유도 tree는 순서 트리(ordered tree)가 된다.

예6) <예4> (1)의 유도 과정에 대하여 각 단계에 따라 유도 트리를 만들어 보자.
(좌단유도, 우단유도 모두 같은 모양의 tree로 구성됨)

$$E \rightarrow E+E \mid E*E \mid (E) \mid -E \mid a$$

※ 문장 $a+a*a$를 이 문법에 의해 좌단 유도를 하면 두 가지의 유도 과정 발생
$\Longrightarrow$ 이런 문법을 모호한(ambiguous) 문법이라고 한다.

$$
\begin{aligned}
E &\Rightarrow E + E \\
&\Rightarrow a + E \\
&\Rightarrow a + E * E \\
&\Rightarrow a + a * E \\
&\Rightarrow a + a * a
\end{aligned}
\qquad\qquad
\begin{aligned}
E &\Rightarrow E * E \\
&\Rightarrow E + E * E \\
&\Rightarrow a + E * E \\
&\Rightarrow a + a * E \\
&\Rightarrow a + a * a
\end{aligned}
$$



[그림 5.2] a+a*a를 위한 유도 트리

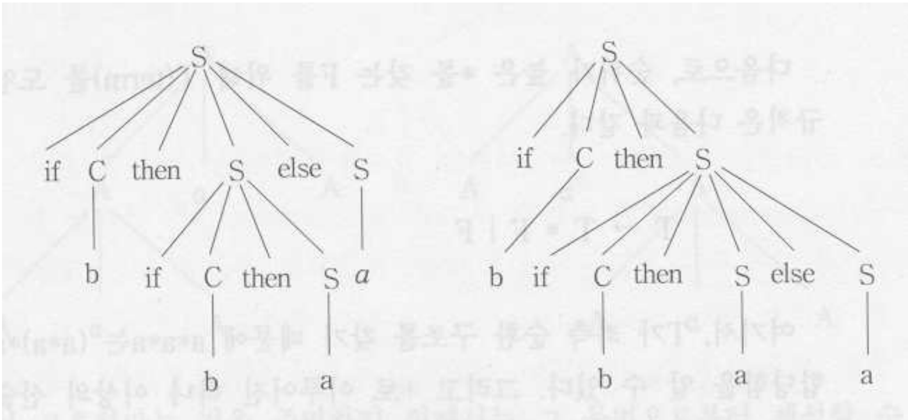
5.2.3 모호성(ambiguity)

[정의 5.4] 문법의 모호성
문법 G에 의해 생성되는 어떤 문장이 두 개 이상의 유도 트리를 갖는다면, 문법 G는 모호하다(ambiguous)고 한다. 즉, 모호한 문법은 어떤 문장에 대해 좌단유도나 우단유도 중 어느 한 가지를 이용하더라도 두 개 이상의 유도 과정이 존재하는 경우이다. 따라서 어떤 문법이 모호함을 증명하려면 그 문법에 속하는 임의의 문장을 선택하여 두 개 이상의 유도 트리를 구성하면 된다.

예7) 다음 문법은 모호한 문법이다.

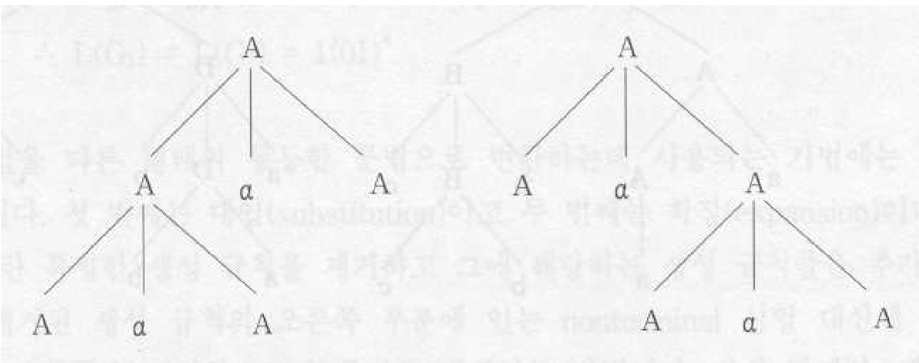
$$
\begin{aligned}
&S \rightarrow if\ C\ then\ S\ else\ S \\
&S \rightarrow if\ C\ then\ S \\
&S \rightarrow a \\
&C \rightarrow b
\end{aligned}
$$

증명: *if b then if b then a else a*에 대해 두 개의 유도 트리



※ 구문분석기(parser)의 출력 : parse tree
어떤 문장에 대한 parse tree가 모호성(ambiguity)을 가지면 어떤 object code를 생산할지 알 수 없다.
⇒ 모호하지 않은 문법으로 변환 가능, 모든 경우에 변환이 가능하지는 않음

※ 생성규칙의 형태가 $A \rightarrow A\,\alpha A$, $A \rightarrow \beta$이면 ambiguous하다.
∵ $A\,\alpha A\,\alpha A$와 같은 문장에 대하여 2개의 tree가 생성 가능



⇒ 모호성 제거

① α가 좌측결합(left associative) 연산자인 경우

$A \rightarrow A\,\alpha A' \mid A'$

$A' \rightarrow \beta$

② α가 우측결합(right associative) 연산자인 경우

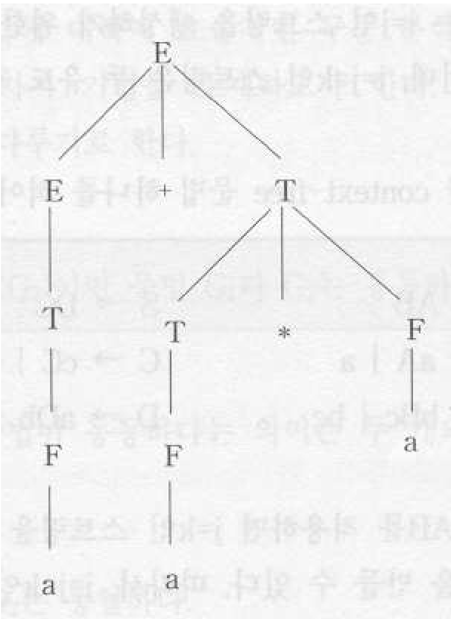
$A \rightarrow A' \alpha A \mid A'$

$A' \rightarrow \beta$

예) 다음 문법을 unambiguous grammar로 변환

$E \rightarrow E+E \mid E*E \mid (E) \mid a$

$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid a$



$E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid E\uparrow E \mid -E \mid (E) \mid id$

- 연산자의 우선순위
  +, - ≺ *, / ≺ ↑ ≺ -(unary)

① 우선순위가 가장 높은 것부터
<element> → (<expression>) | id

② 다음으로 높은 우선순위
<primary> → -<element> | <element>

③ 연산자 ↑는 우측결합(right associative)이므로
<factor> → <primary>↑<factor> | <primary>

④ 연산자 *, /는 좌측결합(left associative)이므로

<term> → <term> * <factor> | <term> / <factor> | <factor>

⑤ 연산자 +, -는 좌측결합(left associative)이므로

<expr> → <expr> + <term> | <expr> - <term> | <term>

[정의 5.5] 언어의 모호성

어떤 CFL을 생성하는 모든 문법이 모호하면, 이 언어는 본질적으로 모호하다.

예8) $L=\{a^i b^j c^k \mid i=j \text{ 또는 } j=k\}$ 는 본질적으로 모호한 CFL이다.

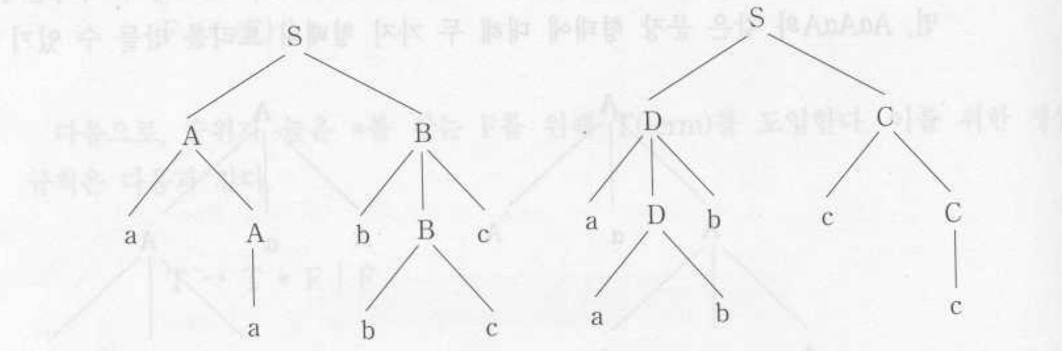
<이유>

i=j인 스트링을 생성하려면 j=k인 경우의 유도과정과 다른 과정을 거쳐야 한다. 마찬가지로, j=k인 스트링을 생성하려면 i=j인 경우의 유도과정과 다른 과정을 거쳐야 한다. 그런데, i=j=k의 경우는 두 유도과정이 모두 가능하기 때문이다.

언어 L을 위한 context-free 문법 하나를 적어 보면 다음과 같다.

S → AB	S → DC
A → aA | a	C → cC | c
B → bBc | bc	D → aDb | ab

처음에 S → AB를 적용하면 j=k인 스트링을 생성할 수 있고, S → DC를 적용하면 i=j인 스트링을 만들 수 있다. 따라서, i=j=k인 스트링 aabbcc에 대한 유도 트리는 다음과 같이 두 개를 만들 수 있다.



5.3 문법 변환

[정의 5.6] $L(G_1)$ = $L(G_2)$이면 문법 G_1과 G_2는 동등하다(equivalent)고 말한다.

예9) 다음 CFG G_1과 G_2는 동일하다.

G_1 : $A \rightarrow 1B$ $\quad$ G_2 : $X \rightarrow Y1$
$A \rightarrow 1$ $\quad$ $X \rightarrow 1$
$B \rightarrow 0A$ $\quad$ $Y \rightarrow X0$
$\therefore L(G_1) = L(G_2) = 1(01)^*$

※ 문법 변환 방법

① 대입(substitution) : 특정 생성규칙을 제거하고 그에 해당하는 생성규칙 추가
$A \rightarrow \alpha B \gamma$와 $B \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$에 대하여
$A \rightarrow \alpha\beta_1\gamma \mid \alpha\beta_2\gamma \mid \ldots \mid \alpha\beta_n\gamma$

② 확장(expansion) : 새로운 생성규칙을 도입하여 한 개의 생성규칙을 분해
$A \rightarrow \alpha\beta$ 에 대하여 $A \rightarrow \alpha X$ 와 $X \rightarrow \beta$

예10) $S \rightarrow aT$ / bT, $T \rightarrow S$ / Sb / c 에서 생성규칙 T를 제거하시오.
$\Longrightarrow$ T를 S에 대입

5.3.1 불필요한 생성규칙 제거

- 문장을 생성하는데 적용될 수 없는 생성규칙들은 모두 제거

[정의 5.7] nonterminal X가 유도과정에서 전혀 사용되지 않는다면 X는 불필요한 symbol이다. 이 때, X를 포함한 모든 생성규칙을 제거해도 된다. 불필요한 symbol에는 non-terminating symbol과 inaccessible symbol이 있다.

[정의 5.8]
terminating nonterminal: terminal string을 생성할 수 있는 nonterminal (알고리즘 5.1)
accessible symbol: 시작기호로부터 도달할 수 있는 symbol (알고리즘 5.2)

[알고리즘 5.1] Terminating nonterminal 구하는 방법

Algorithm terminating
begin
    V_N' : = { A | A→ ω∈P, ω∈V_T^* };
    repeat
        V_N' := V_N' ∪ { A | A→ α∈P, α∈(V_N'∪V_T)* }
    until no change
end.

[알고리즘 5.2] Accessible symbol을 구하는 방법

Algorithm accessible
begin
    V′ := { S };
    repeat
        V′ := V′ { X | some A→ αXβ∈P, A∈V′ }
    until no change
end.

예11) $P = \{ S \rightarrow A, S \rightarrow B, B \rightarrow a \}$ 에서 terminating nonterminal 집합은?
$\{ S, B \}$

예12) $P = \{ S \rightarrow aB, A \rightarrow aB, A \rightarrow aC, B \rightarrow C, C \rightarrow b \}$ 에서 도달가능한 symbol 집합은?

알고리즘 5.2에 따라 구하면 : $\{ S, a, B, C, b \}$

예13) $P = \{ S \rightarrow aS, S \rightarrow A, S \rightarrow B, A \rightarrow aA, B\rightarrow a, C\rightarrow aa \}$ 에서 불필요한 생성규칙 제거.

5.3.2 ε-생성규칙 제거 --- [정의 5.9] ε-*free* 문법

정의 5.9 CFG $G=(V_N, V_T, P, S)$가 다음과 같은 조건을 가질 때 ε-free라 한다.

(1) P가 ε-생성 규칙을 가지지 않거나,

(3) S 하나만이 ε-생성 규칙을 가지며, 다른 생성 규칙의 오른쪽에 S가 나타나지 않아야 한다.

생성규칙의 형태가 $A \rightarrow \varepsilon$ 인 것 제거 --- 알고리즘 5.3

[알고리즘 5.3] ε-free 문법으로 변환하는 방법

알고리즘 5.3 ε-free 문법으로 변환하는 방법:

```
Algorithm ε-free;
begin
  P' := P - {A→ε| A∈V_N};
  V_Nε := {A| A ⇒+ ε, A∈V_N};
  for A→α0B1α1B2 ··· Bkαk∈P', where αi≠ε and Bi∈V_Nε do
    만약 Bi-생성 규칙이 P'에 존재하면,
        A→α0X1α1X2 ··· Xkαk에 대하여 Xi=ε 혹은 Xi=Bi의 조합에 의해
        나올 수 있는 모든 생성 규칙을 P'에 추가한다.
    그렇지 않은 경우,
        A→α0X1α1X2 ··· Xkαk에서 Xi=ε인 생성 규칙을 P'에 추가한다.
  end for
  if S∈V_Nε then P' := P' ∪ {S'→ε|S}
end.
```

예14) $S \rightarrow aSbS$ / $bSaS$ / ε

① 각 생성규칙에 대하여 ε 제거

$S \rightarrow aSbS \;\Rightarrow\; S \rightarrow aSbS \mid aSb \mid abS \mid ab$

$S \rightarrow bSaS \;\Rightarrow\; S \rightarrow bSaS \mid bSa \mid baS \mid ba$

② 시작기호가 ε으로 생성되면 $S' \rightarrow S$ / ε를 추가

알고리즘 5.4 $V_{N\varepsilon}$를 구하는 방법:

```
Algorithm Compute_VNε;
begin VNε := {A | A→ε∈P};
      repeat
         VNε := VNε ∪ {B | B→α∈P, α∈VNε*}
      until no change
end.
```

5.3.3 단일 생성규칙 제거 --- 알고리즘 5.5

$A \rightarrow B$ 와 같이 생성규칙의 오른쪽에 한 개의 nonterminal만 오는 경우
⇒ 불필요한 유도과정이 생기며 parsing 속도가 느려진다.

알고리즘 5.5 단일 생성 규칙 제거 방법:

```
Algorithm Remove_Single_Production;
begin P' := P - {A→B | A, B∈VN};
   for each A∈VN do
      VNA := {B | A⇒+B};
      for each B∈VNA do
         for each B→α∈P' do (* not single production *)
            P' := P' ∪ {A→α}
         end for
      end for
   end for
end.
```

알고리즘 5.6 V_{NA}를 구하는 방법 :

```
Algorithm Compute_VNA;
begin VNA := {B | A→B∈P};
   repeat
      VNA := VNA ∪ {C | B→C∈P, B∈VNA}
   until no change
end.
```

예15) 아래 문법에 대해 단일 생성규칙을 제거해 보자.

$E \rightarrow E+T \mid T$
$T \rightarrow T*F \mid F$
$F \rightarrow (E) \mid a$

$E \rightarrow E+T \mid T*F \mid (E) \mid a$
$T \rightarrow T*F \mid (E) \mid a$
$F \rightarrow (E) \mid a$

[정의 5.10] cycle-free이고 ε-free인 문법을 proper하다고 한다.

cycle-free 문법 : A⇒...⇒A 형태의 유도과정이 존재하지 않는 문법

5.3.4 문법의 기본적인 형태(Canonical Forms of Grammar)

[정의 5.11] Chomsky Normal Form : CNF

CFG $G=(V_N,\ V_T,\ P,\ S)$의 생성규칙이 다음 형태로 이루어져 있을 때 G를 정규형태(normal form) 혹은 CNF라 한다.

(1) $A\rightarrow BC$, 여기서 $A,B,C\in V_N$
(2) $A\rightarrow a,\ a\in V_T$
(3) 만약 $\varepsilon\in L(G)$라면, $S \rightarrow \varepsilon$이고 S는 생성규칙의 RHS에 나타나지 않음

※ 모든 CFG는 CNF로 바꿀 수 있다.

방법 : ① CFG를 ε-free 문법으로 변환

② $A\rightarrow\alpha$, $|\alpha|>2$인 생성규칙의 RHS를 두 개의 symbol을 갖도록 변환

$A\rightarrow X_1X_2\cdots X_k,\ k>2$
$\Rightarrow\ A\rightarrow X_1'\langle X_2\cdots X_k\rangle$
$\langle X_2\cdots X_k\rangle\rightarrow X_2'\langle X_3\cdots X_k\rangle$
$\cdots$
$\langle X_{k-1}X_k\rangle\rightarrow X_{k-1}'X_k'$

※ $\langle X_i\cdots X_k\rangle$는 *nonterminal*이고,
X_i'은 *nonterminal*이면 X_i', *terminal*이면 생성규칙 $X_i'\rightarrow X_i$가 추가된다.

예16) 다음 문법을 CNF로 변환하시오.

$S \rightarrow aAB \mid BA$

$A \rightarrow BBB \mid a$

$B \rightarrow AS \mid b$

$S \rightarrow aAB$에 대하여

$S \rightarrow a'\langle AB \rangle$

$a' \rightarrow a$

$\langle AB \rangle \rightarrow AB$

[정의 5.12] 표준형태(standard form) 또는 Greibach 정규형태

CFG $G=(V_N, V_T, P, S)$가 ε-free이고 ε-생성규칙이 존재하지 않는 $A \rightarrow a\alpha$, $a \in V_T$, $\alpha \in V_N^*$ 형태의 생성규칙을 갖는 문법

5.4 CFG 표기법 : BNF와 EBNF

- 명칭의 정의

<id> ::= <letter> | <id><letter> | <id><digit>
<letter> ::= *a* / *b* / *c* / … / *y* / *z*
<digit> ::= *0* / *1* / *2* / … / *8* / *9*

- comma로 구분되는 명칭의 정의

① BNF

<id_list> ::= <id_list>, <id> | <id>

② EBNF --- 반복되는 부분을 {}로 표현

<id_list> ::= <id> {, <id> }

예18) compound statement의 CFG 표현

① BNF
<compound_statement> ::= begin <statement_list> end
<statement_list> ::= <statement_list>; <statement> | <statement>

② EBNF --- 반복되는 부분을 {}, 선택은 []로 표현
<compound_statement> ::= begin <statement> {; <statement>} end

※ EBNF : 반복횟수 표현 $\{\}_0^7$ --- 최소 0번, 최대 7번
[x] --- $\{x\}_0^1$과 같은 의미

예) if 문 : <if_stmt> ::= if <condition> then <statement> [else <statement>]

예19) 단순변수와 1차원 배열에 대한 BNF와 EBNF

① BNF
<variable> ::= <id> | <id> '[' <exp> ']'

② EBNF

<variable> ::= <id> ['[' <exp> ']']

※ EBNF에서 사용되는 meta symbol들을 terminal로 사용할 때는 ‘’로 묶어 표현

※ 괄호와 택일기호 |를 사용하여 여러 개의 생성규칙들을 묶어서 간단히 표현

예) <exp> ::= <exp>+<exp> | <exp>-<exp> | <exp>*<exp> | <exp>/<exp>
⇒ <exp> ::= <exp> (+ | - | * | /) <exp>

예20) BNF의 정의

※ CFG는 syntax diagram(구문 도표)으로 표현하기도 한다.

5.5 pushdown automata (PDA)

- finite automata의 제약

  기억장소가 없으므로 처리된 input symbol의 수를 헤아릴 수 없음

- PDA

  finite-state control, input tape, stack으로 구성 (그림 5.3)

[정의 5.13] PDA는 7개 요소로 구성된다.

P = $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

Q : *finite set of states*

Σ : *finite set of input alphabets*

Γ : *finite set of stack symbols*

δ : 상태전이함수 $Q\times(\Sigma\cup\{\varepsilon\})\times\Gamma \rightarrow Q\times\Gamma^*$

$q_0\in Q$: *start state*

$Z_0\in\Gamma$: *stack*의 *start state*

$F\in Q$: *set of final states*

- 상태전이 함수 δ는 다음과 같은 형태를 갖는다.

  $\delta(q, a, Z) = \{(p_1, \alpha_1), (p_2, \alpha_2), \cdots, (p_n, \alpha_n)\}$

- 현재상태 q, input symbol a, stack의 top symbol Z에서 (p_i, α_I)가 선택되었다면,

  (1) next state는 p_i

  (2) stack의 top symbol Z를 α_i로 대치

[정의 5.14] PDA의 작동 상태 : $(q, \omega, \alpha) \Leftarrow Q\times\Sigma^*\times\Gamma^*$의 triple

q : 현재 상태

ω : 읽지 않은 input 부분

α : stack의 내용 (α의 왼쪽 1-st symbol이 stack의 top symbol)

- 상태전이의 예

$(q,\ a\omega,\ Z\alpha) \vdash (q',\omega,\gamma\alpha)$ --- $\delta(q,\ a,\ Z) = \{(q',\gamma),\ \cdots\}$에 대하여

$\vdash^*,\ \vdash^+$

- q에서 q'으로 상태전이
- input head가 한 칸 이동
- stack의 top Z가 γ로 대치 (γ가 ε이면 pop)
- a가 ε이면 ε-move

- PDA P의 시작 형태 : $\omega\in\Sigma^*$일 때 $(q_0,\ \omega,\ Z_0)$
- PDA P의 종결 형태 : (q,ε,α), $q\in F$이고 $\alpha\in\Gamma^*$

[정의 5.15] PDA P에 의해 accept되는 언어 L(P)

· input string ω를 다 본 상태가 ε이고 final state에 있으면, string ω는 P에 의해 인식된다고 말한다.

· $L(P) = \{\ \omega /\ (q_0,\ \omega,\ Z_0) \vdash^* (q,\ \varepsilon,\ \alpha),\ q\in F,\ \alpha\in\Gamma^*\}$

예26) 언어 $L = \{\ 0^n1^n \mid n\geq 1\ \}$을 인식하는 PDA

$$P = (\{q_0,\ q_1,\ q_2\},\ \{0,\ 1\},\ \{Z,\ 0\},\ \delta,\ q_0,\ Z,\ \{q_0\})$$

$$\delta(q_0,\ 0,\ Z) = \{\ (q_1,\ 0Z)\ \}$$
$$\delta(q_1,\ 0,\ 0) = \{\ (q_1,\ 00)\ \}$$
$$\delta(q_1,\ 1,\ 0) = \{\ (q_2,\ \varepsilon)\ \}$$
$$\delta(q_2,\ 1,\ 0) = \{\ (q_2,\ \varepsilon)\ \}$$
$$\delta(q_2,\ \varepsilon,\ Z) = \{\ (q_0,\ \varepsilon)\ \}$$

· input string 0011에 대하여 P가 인식하는 과정

$$\begin{aligned}(q_0,\ 0011,\ Z) &\vdash (q_1,\ 011,\ 0Z)\\ &\vdash (q_1,\ 11,\ 00Z)\\ &\vdash (q_2,\ 1,\ 0Z)\\ &\vdash (q_2,\ \varepsilon,\ Z)\\ &\vdash (q_0,\ \varepsilon,\ \varepsilon)\end{aligned}$$

· input string 0^n1^n에 대하여 인식하는 과정

$$
\begin{aligned}
(q_0,\ 0^n1^n,\ Z) &\vdash (q_1,\ 0^{n-1}1^n,\ 0Z) \\
&\vdash^{n-1} (q_1,\ 1^n,\ 0^nZ) \\
&\vdash (q_2,\ 1^{n-1},\ 0^{n-1}Z) \\
&\vdash^{n-1} (q_2,\ \varepsilon,\ Z) \\
&\vdash (q_0,\ \varepsilon,\ \varepsilon)
\end{aligned}
$$

※ stack이 empty이면 더 이상 move가 발생하지 않는다.

[정의 5.16] Extended PDA

$P = (Q,\ \Sigma,\ \Gamma,\ \delta,\ q_0,\ Z_0,\ F)$

δ : 상태전이함수 $Q\times(\Sigma\cup\{\varepsilon\})\times\Gamma^* \rightarrow Q\times\Gamma^*$

예27) $L = \{\omega\omega^R \mid \omega\in\{a,b\}^+\}$를 인식하는 확장된 PDA를 정의해 보자.

$P = (\{q\},\ \{a,\ b\},\ \{a,\ b,\ S,\ Z\},\ \delta,\ q,\ Z,\ \{q\})$

$$
\begin{aligned}
\delta(q,\ a,\ \varepsilon) &= \{\ (q,\ a)\ \} \\
\delta(q,\ b,\ \varepsilon) &= \{\ (q,\ b)\ \} \\
\delta(q,\ \varepsilon,\ \varepsilon) &= \{\ (q,\ S)\ \} \\
\delta(q,\ \varepsilon,\ aSa) &= \{\ (q,\ S)\ \} \\
\delta(q,\ \varepsilon,\ bSb) &= \{\ (q,\ S)\ \} \\
\delta(q,\ \varepsilon,\ SZ) &= \{\ (q,\ \varepsilon)\ \}
\end{aligned}
$$

· input string *aabbaa*에 대하여 인식하는 과정

$$
\begin{aligned}
(q,\ aabbaa,\ Z) &\vdash (q,\ abbaa,\ aZ) \\
&\vdash (q,\ bbaa,\ aaZ) \\
&\vdash (q,\ baa,\ baaZ) \\
&\vdash (q,\ baa,\ SbaaZ) \\
&\vdash (q,\ aa,\ bSbaaZ) \\
&\vdash (q,\ aa,\ SaaZ) \\
&\vdash (q,\ a,\ aSaaZ) \\
&\vdash (q,\ \varepsilon,\ aSaZ) \\
&\vdash (q,\ \varepsilon,\ SZ) \\
&\vdash (q,\ \varepsilon,\ \varepsilon)
\end{aligned}
$$

[정의 5.17] stack을 empty로 만드는 확장된 PDA P에 의해 인식되는 언어

$$Le(P) = \{ \omega \mid (q_0, \omega, Z_0) \vdash^{+} (q, \varepsilon, \varepsilon), q \in Q \}$$

[정리 5.1] PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$에 의해 인식되는 언어 $L(P)$를 L이라 할 때, $Le(P') = L$인 PDA P'을 만들 수 있다.

5.6 Context-free 언어와 PDA 언어

[정리 5.2] 문법 $G = (V_N, V_T, P, S)$가 CFG일 때, G로부터 $Le(R) = L(G)$인 PDA R을 만들 수 있다.

- **좌단유도를 행하는 PDA R을 구성**

$$R = (\{q\}, V_T, V_N \cup V_T, \delta, q, S, \phi)$$

(1) $A \rightarrow \alpha \in P$ 이면, $\delta(q, \varepsilon, A)$에 (q, α)를 추가

(2) 모든 $a \in V_T$에 대하여, $\delta(q, a, a) = \{ (q, \varepsilon) \}$

(증명) $A \Rightarrow \omega$와 $(q, \omega, a) \vdash^{n} (q, \varepsilon, \varepsilon)$이 서로 필요충분 조건이 됨을 증명.

예28) 문법 G에 대하여 좌단유도를 행하는 PDA R을 구성.

$$G = (\{E, T, F\}, \{a, *, +, (,)\}, P, E)$$

$$\begin{aligned} P : E &\rightarrow E+T \mid T \\ T &\rightarrow T*F \mid F \\ F &\rightarrow (E) \mid a \end{aligned}$$

(1) $\delta(q, \varepsilon, E) = \{ (q, E+T), (q, T) \}$

(2) $\delta(q, \varepsilon, T) = \{ (q, T*F), (q, F) \}$

(3) $\delta(q, \varepsilon, F) = \{ (q, (E)), (q, a) \}$

(4) $\delta(q, x, x) = \{ (q, \varepsilon) \}$, 여기서 $x \in \{ a, +, *, (,) \}$

· input string $a+a*a$에 대한 PDA R의 인식 과정

$$
\begin{aligned}
(q,\ a+a*a,\ E) &\vdash (q,\ a+a*a,\ E+T) \\
&\vdash (q,\ a+a*a,\ T+T) \\
&\vdash (q,\ a+a*a,\ F+T) \\
&\vdash (q,\ a+a*a,\ a+T) \\
&\vdash (q,\ +a*a,\ +T) \\
&\vdash (q,\ a*a,\ T) \\
&\vdash (q,\ a*a,\ T*F) \\
&\vdash (q,\ a*a,\ F*F) \\
&\vdash (q,\ a*a,\ a*F) \\
&\vdash (q,\ *a,\ *F) \\
&\vdash (q,\ a,\ F) \\
&\vdash (q,\ a,\ a) \\
&\vdash (q,\ \varepsilon,\ \varepsilon)
\end{aligned}
$$

- **우단 역유도(reverse derivation)를 행하는 PDA R을 구성**

$R = (\ \{q,\ r\},\ V_T,\ V_N \cup V_T,\ \delta,\ q,\ \$,\ \{r\}\ )$

(1) $A \rightarrow \alpha \in P$ 이면, $\delta(q, \varepsilon, \alpha)$에 $(q,\ A)$를 추가

(2) 모든 $a \in V_T$에 대하여, $\delta(q,\ a,\ \varepsilon) = \{\ (q,\ a)\ \}$

(3) $\delta(q, \varepsilon,\ \$S) = \{\ (r, \varepsilon)\ \}$

예29) 예28의 문법에 대하여 우단 역유도를 행하는 PDA R을 구성.

(1) $\delta(q, x, \varepsilon) = \{\ (q,\ x)\ \}$, 여기서 $x \in \{\ a,\ +,\ *,\ (,\ )\ \}$

(2) $\delta(q, \varepsilon,\ E+T) = \{\ (q,\ E)\ \}$

$\delta(q, \varepsilon,\ T) = \{\ (q,\ E)\ \}$

$\delta(q, \varepsilon,\ T*F) = \{\ (q,\ T)\ \}$

$\delta(q, \varepsilon,\ F) = \{\ (q,\ T)\ \}$

$\delta(q, \varepsilon,\ (E)) = \{\ (q,\ F)\ \}$

$\delta(q, \varepsilon,\ a) = \{\ (q,\ F)\ \}$

(3) $\delta(q, \varepsilon,\ \$E) = \{\ (r, \varepsilon)\ \}$

· input string $a+a*a$에 대한 인식 과정

$$
\begin{aligned}
(q,\ a+a*a,\ \$) &\vdash (q,\ +a*a,\ \$a) \\
&\vdash (q,\ +a*a,\ \$F) \\
&\vdash (q,\ +a*a,\ \$T) \\
&\vdash (q,\ +a*a,\ \$E) \\
&\vdash (q,\ a*a,\ \$E+) \\
&\vdash (q,\ *a,\ \$E+a) \\
&\vdash (q,\ *a,\ \$E+F) \\
&\vdash (q,\ *a,\ \$E+T) \\
&\vdash (q,\ a,\ \$E+T*) \\
&\vdash (q,\ \varepsilon,\ \$E+T*a) \\
&\vdash (q,\ \varepsilon,\ \$E+T*F) \\
&\vdash (q,\ \varepsilon,\ \$E+T) \\
&\vdash (q,\ \varepsilon,\ \$E) \\
&\vdash (r,\ \varepsilon,\ \varepsilon)
\end{aligned}
$$

※ top-down parsing vs. bottom-up parsing

top-down parsing : 좌단유도
bottom-up parsing : 우단 역유도

[정리 5.3] $R = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$가 PDA라면 $L(G) = Le(R)$인 CFG를 만들 수 있다.

※ 다음은 모두 동등한 의미를 갖는다.
(1) CFG G에 의해 생성되는 언어 $L(G)$
(2) PDA P에 의해 생성되는 언어 $L(P)$
(3) PDA P에 의해 생성되는 언어 $Le(P)$
(4) Extended PDA P에 의해 생성되는 언어 $L(P)$