

# Top-down Parsing

컴파일러의 구문 분석 단계에서 중요한 역할을 하는 Top-down Parsing에 대해 알아보겠습니다. 이 강의에서는 이론적 배경인 Pushdown Automata부터 시작하여 실제 파서 구현에 필요한 LL(1) 파싱 테이블 생성까지의 전 과정을 다룹니다.

# 문맥 자유 문법 (Context Free Grammar)

## 문맥 자유 문법 정의

문맥 자유 문법  $G = (VN, VT, P, S)$ 로 정의됩니다.

- $VN$ : 비단말 기호 집합
- $VT$ : 단말 기호 집합
- $P$ : 생성 규칙 집합
- $S$ : 시작 기호

## 생성 규칙 형태

$\alpha \rightarrow \beta, \alpha \in V^+, \beta \in V^*$

특히 문맥 자유 문법에서는  $\alpha \in VN, \beta \in V^*$ 의 형태를 가집니다.

# 문법의 계층 구조

타입	문법	인식 장치	예시
0	무제한 문법	튜링 머신	모든 계산 가능한 언어
1	문맥 의존 문법(CSG)	선형 제한 오토마타(LBA)	제약이 있는 구문(타입 체킹 등)
2	문맥 자유 문법(CFG)	푸시다운 오토마타(PDA)	중첩 구조 파싱

# 푸시다운 오토마타(PDA)

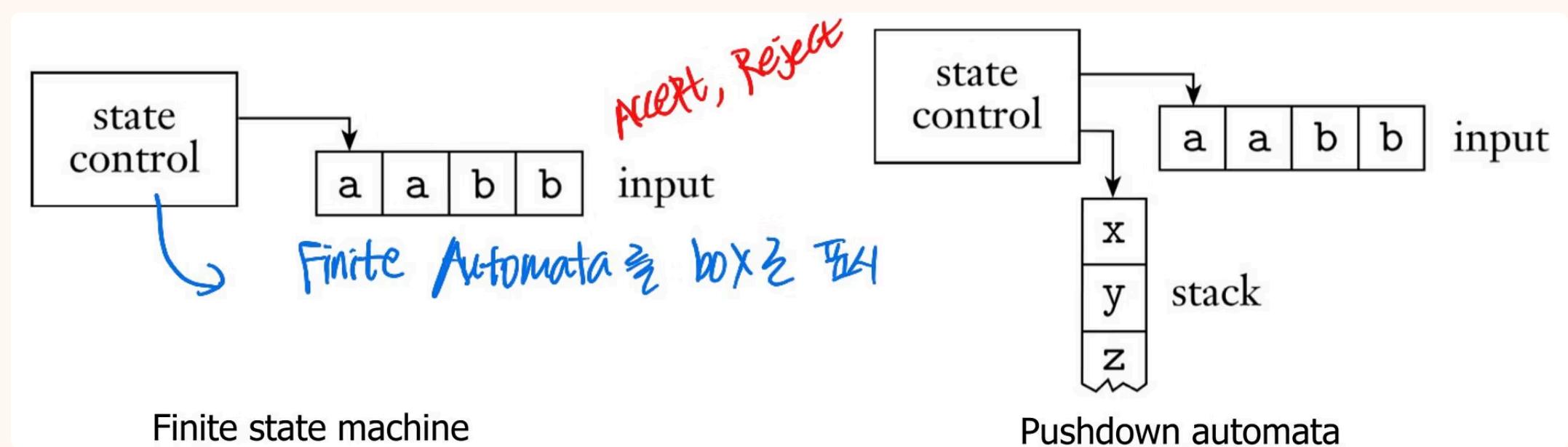
## 1 문맥 자유 언어(CFL)

푸시다운 오토마타에 의해 인식되는 언어입니다.

## 2 PDA의 구성

간단히 말하면:  $PDA = [\epsilon\text{-NFA} + \text{"스택"}]$

무한한 크기의 스택을 가진 유한 상태 기계입니다.



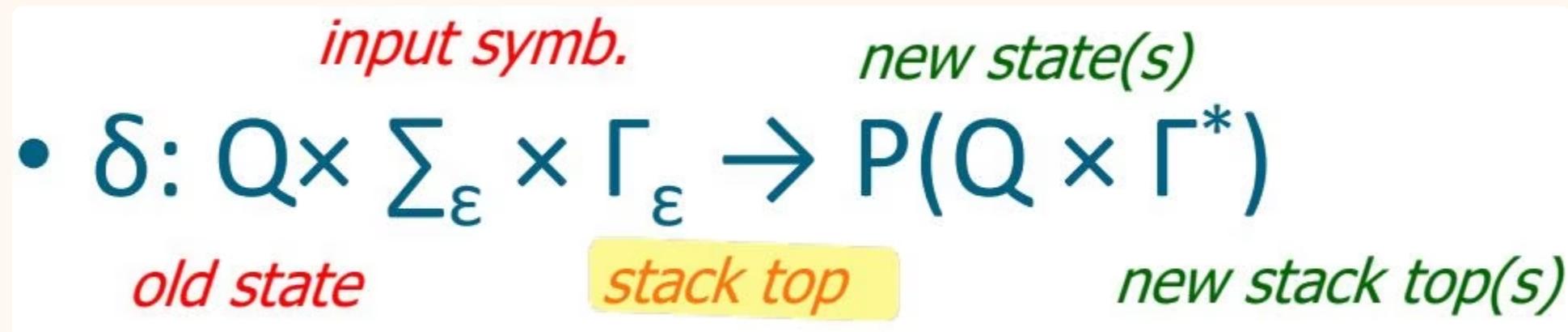
# 푸시다운 오토마타의 정형적 정의

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

- $Q$ : 유한한 상태 집합
- $\Sigma$ : 유한한 입력 기호 집합(알파벳)
- $\Gamma$ : 유한한 푸시다운 스택 기호 집합(알파벳)
- $q_0$ : 시작 상태(초기 상태)  $q_0 \in Q$
- $Z_0$ : 스택의 시작 기호  $Z_0 \in \Gamma$
- $F$ : 최종(수용) 상태 집합  $F \subseteq Q$
- $\delta$ : 전이 함수

전이 함수  $\delta$

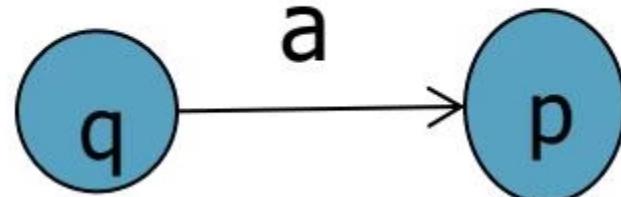
- $\delta(q, a, X) = \{(p, Y), \dots\}$
- $q$ 에서  $p$ 로 상태 전이
  - $a$ 는 다음 입력 기호
  - $X$ 는 현재 스택 최상위 기호
  - $Y$ 는  $X$ 의 대체물;  $\Gamma^*$ (스택 기호의 문자열)에 속함



# 스택 연산

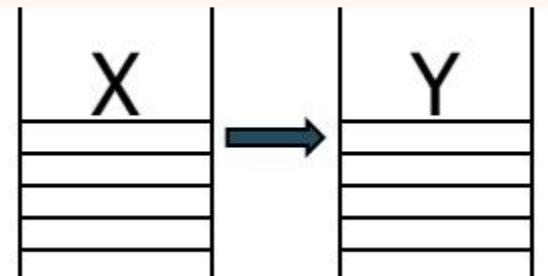
1  $Y = \epsilon$ 인 경우

$\text{Pop}(X)$  - 스택 최상위 요소를 제거합니다.



2  $Y = X$ 인 경우

스택 최상위가 변경되지 않습니다.



3  $Y = Z_1 Z_2 \dots Z_k$ 인 경우

X가 팝되고 Y로 대체됩니다(역순으로, 즉  $Z_1$ 이 새로운 스택 최상위가 됨).

# PDA 예제: 회문 언어

회문 언어 정의

$$L_{\text{wwr}} = \{wwR \mid w \in \{0,1\}^*\} = \{\epsilon, 00, 011, 1111, 0000, \dots\}$$

여기서  $wR$ 은  $w$ 의 역순을 의미합니다.

회문 언어의 CFG

$$S \rightarrow 0S0 \mid 1S1 \mid \epsilon$$

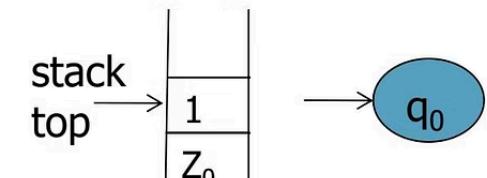
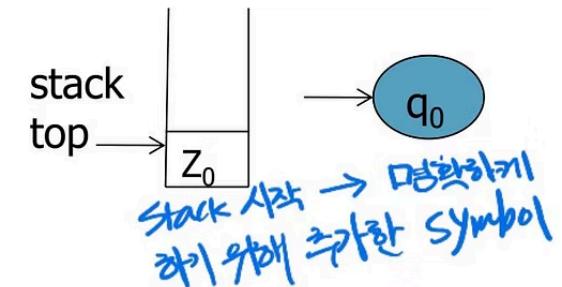
회문 언어의 PDA

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F) = (\{q_0, q_1, q_2\}, \{0,1\}, \{0,1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$$

## Transition function of P

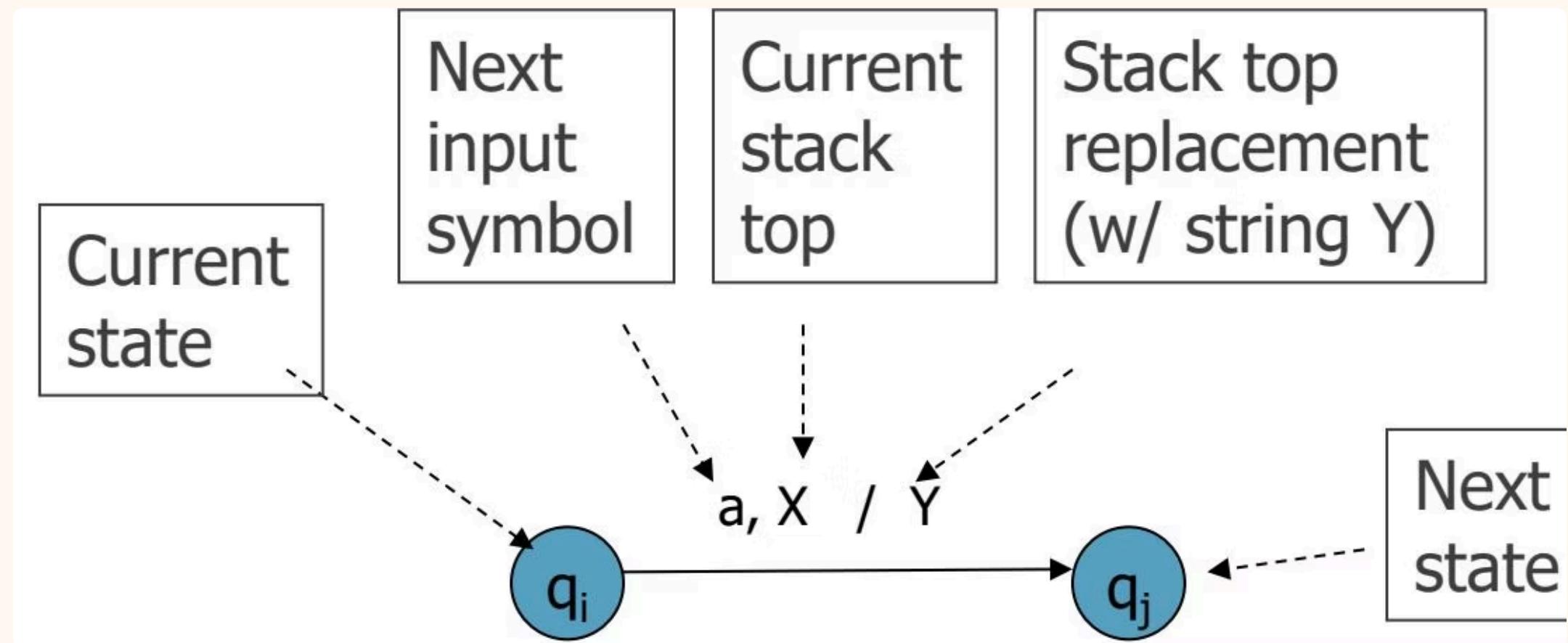
- $L_{\text{wwr}} = \{wwR \mid w \in \{0,1\}^*\}$
1.  $\delta(q_0, 0, Z_0) = \{(q_0, 0Z_0)\}$   
2.  $\delta(q_0, 1, Z_0) = \{(q_0, 1Z_0)\}$
3.  $\delta(q_0, 0, 0) = \{(q_0, 00)\}$   
4.  $\delta(q_0, 0, 1) = \{(q_0, 01)\}$   
5.  $\delta(q_0, 1, 0) = \{(q_0, 10)\}$   
6.  $\delta(q_0, 1, 1) = \{(q_0, 11)\}$
7.  $\delta(q_0, \epsilon, 0) = \{(q_1, 0)\}$   
8.  $\delta(q_0, \epsilon, 1) = \{(q_1, 1)\}$   
9.  $\delta(q_0, \epsilon, Z_0) = \{(q_1, Z_0)\}$
10.  $\delta(q_1, 0, 0) = \{(q_1, \epsilon)\}$   
11.  $\delta(q_1, 1, 1) = \{(q_1, \epsilon)\}$
12.  $\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$
- INPUT : 0  
Z\_0 stack element 0 → O Push O*
- First symbol push on stack**  
**→ stack의 최상단 element를 의미한다!**  
**→ 무언간 stack-top 단을 본다.**
- Grow the stack by pushing new symbols on top of old (w-part)**  
**State変わったよ!**  
**ε: 임의로 전여한다.**
- Switch to popping mode, nondeterministically (boundary between w and  $w^R$ )**
- POP을 의미함: State変わったよ!**  
**Shrink the stack by popping matching symbols ( $w^R$ -part)**
- Enter acceptance state**  
**Empty stack**

Initial state of the PDA:



# PDA의 상태 다이어그램

PDA를 상태 다이어그램으로 표현할 수 있습니다. 전이 함수  $\delta(q_i, a, X) = \{(q_j, Y)\}$ 는 다음과 같이 표현됩니다:



화살표 위의 레이블은 "a,X/Y" 형식으로, 입력 기호 a를 읽고 스택 최상위 X를 Y로 대체함을 의미합니다.

# 회문 언어 PDA의 전이 다이어그램

Grow stack

Stack 쌓는 과정.

0,  $Z_0/Z_0$   
1,  $Z_0/Z_0$   
0, 0/0  
0, 1/01  
1, 0/10  
1, 1/11

$\epsilon, Z_0/Z_0$



Pop stack for matching symbols

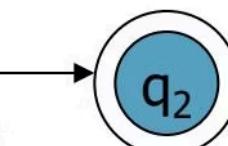
0, 0/  $\epsilon$   
1, 1/  $\epsilon$

$\epsilon, Z_0/Z_0$



$\Sigma = \{0, 1\}$   
 $\Gamma = \{Z_0, 0, 1\}$   
 $Q = \{q_0, q_1, q_2\}$

Go to acceptance



Switch to popping mode

$\epsilon, Z_0/Z_0$   
 $\epsilon, 0/0$   
 $\epsilon, 1/1$

는 포함하기 때문

This would be a **non-deterministic PDA**

첫 번째 단계

$q_0$  상태에서 입력을 읽으며 스택에 저장합니다.

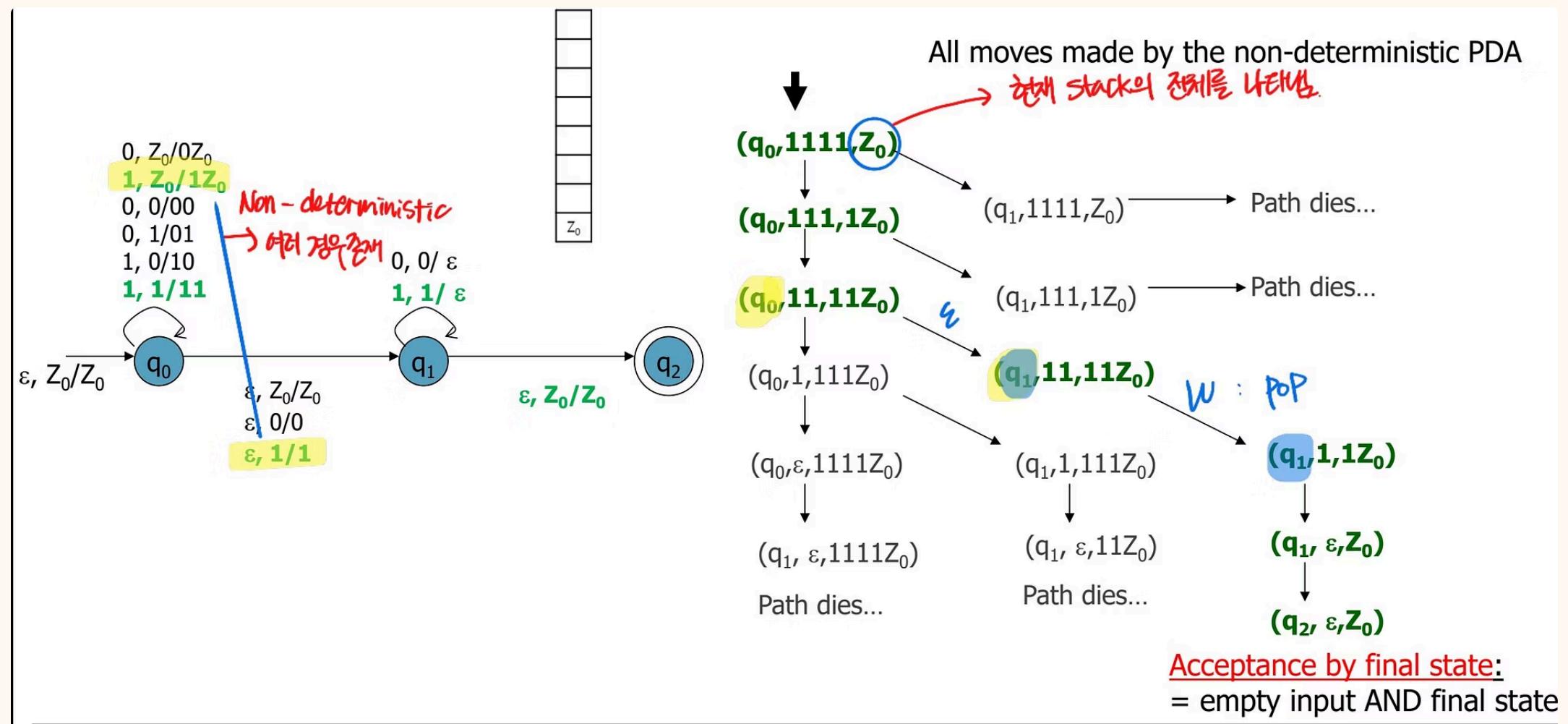
두 번째 단계

$q_1$  상태에서 입력과 스택 최상위를 비교하며 일치하면 스택에서 제거합니다.

마지막 단계

모든 입력을 처리하고 스택이 비어있으면  $q_2$  상태에 도달하여 입력을 수용합니다.

## PDA의 입력 "1111" 처리 과정



# PDA가 수용하는 언어

## 최종 상태로 수용하는 PDA

PDA P에 의해 수용되는 언어  $L(P)$ 는 다음과 같이 정의됩니다:

$$L(P) = \{w \mid (q_0, w, Z_0) \xrightarrow{*} (q, \varepsilon, A)\}, q \in F$$

즉, 시작 상태에서 시작하여 모든 입력을 소비하고 최종 상태에 도달하는 문자열의 집합입니다.

최종 상태로 수용하는 PDA와 빈 스택으로 수용하는 PDA는 동등한 표현력을 가집니다. 어떤 언어가 문맥 자유 언어인 것은 어떤 푸시다운 오토마타가 그 언어를 인식할 때와 같습니다.

## 빈 스택으로 수용하는 PDA

PDA P에 의해 수용되는 언어  $L(P)$ 는 다음과 같이 정의됩니다:

$$L(P) = \{w \mid (q_0, w, Z_0) \xrightarrow{*} (q, \varepsilon, \varepsilon)\}, \text{ 임의의 } q \in Q$$

즉, 시작 상태에서 시작하여 모든 입력을 소비하고 스택이 비어있는 상태에 도달하는 문자열의 집합입니다.

# 결정적 PDA

## 결정적 PDA의 조건

- 유일한 전이: 임의의  $a \in \Sigma$ 에 대해  $\delta(q,a,X)$ 는 최대 하나의 원소만 가짐
- 비- $\epsilon$  전이: 만약  $\delta(q,a,X)$ 가 어떤  $a \in \Sigma$ 에 대해 비어있지 않다면,  $\delta(q,\epsilon,X)$ 는 반드시 비어있어야 함

## 결정적 PDA의 특징

결정적 PDA는 비결정적 PDA보다 표현력이 약하지만, 효율적으로 시뮬레이션할 수 있습니다.

결정적 PDA가 인식하는 언어를 결정적 문맥 자유 언어(DCFL)라고 합니다.

## 결정적 PDA 예제: Lwcwr

Lwcwr = {wcwR | w ∈ ( $\Sigma - \{c\}$ ) $^*$ }는 중간에 c가 있는 회문 언어입니다.

Grow stack

↑ input symbol 하나에 대해 하나만 존재

0,  $Z_0/Z_0$

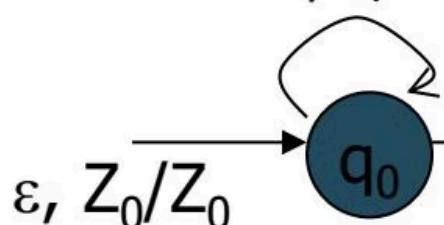
1,  $Z_0/Z_0$

0, 0/00

0, 1/01

1, 0/10

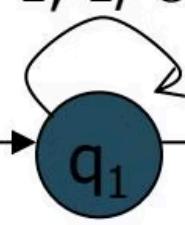
1, 1/11



Pop stack for matching symbols

0, 0/  $\epsilon$

1, 1/  $\epsilon$



$\epsilon, Z_0/Z_0$

c,  $Z_0/Z_0$   
c, 0/0  
c, 1/1

Switch to popping mode

첫 번째 단계

$q_0$  상태에서 c를 만날 때까지 입력을 스택에 저장합니다.

두 번째 단계

c를 읽으면  $q_1$  상태로 전이합니다.

세 번째 단계

$q_1$  상태에서 입력과 스택 최상위를 비교하면 스택에서 제거합니다.

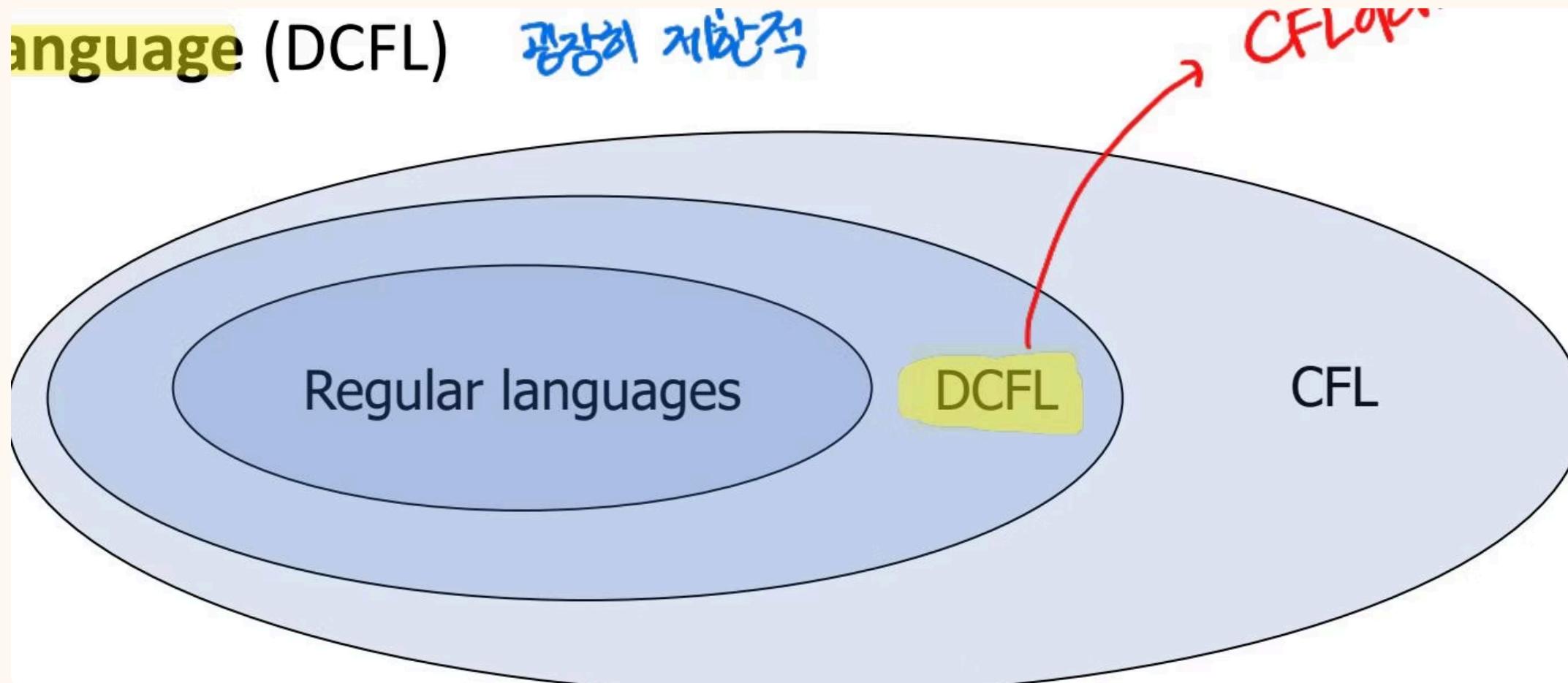
# 결정적/비결정적 PDA 비교

## 비결정적 PDA (NPDA)

- 결정적 PDA보다 더 강력함
- 모든 문맥 자유 언어(CFL)를 인식함

## 결정적 PDA (DPDA)

- CFL의 부분집합인 언어만 인식함
- 결정적 문맥 자유 언어(DCFL)를 인식함



# 결정적 PDA의 중요성



## 효율적인 시뮬레이션

결정적이기 때문에 효율적으로 시뮬레이션할 수 있습니다.



## 테이블 기반 구현

각 입력/스택 쌍에 대해 수행할 스택 연산과 다음 상태를 지정하는 액션/고토 테이블을 저장할 수 있습니다.



## 스택 관리

스택의 최상위만 추적하면 됩니다.



## 입력 처리

오토마타가 거부하거나 모든 입력을 소비하고 수용 상태에서 끝날 때까지 입력/스택 쌍을 처리하며 입력을 반복합니다.

# CFL/DCFL 예제

## CFL 예제

- $\{wwR \mid w \in \Sigma^*\}$  - 회문 언어
- $\{aibj ck \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k\}$  - 두 개의 카운터 중 하나가 일치하는 언어

## DCFL 예제

- $\{wzwR \mid w \in (\Sigma-\{z\})^*\}$  - 중간에 특수 문자가 있는 회문 언어

비결정적 PDA는 모든 문맥 자유 언어를 인식할 수 있지만, 결정적 PDA는 일부 문맥 자유 언어만 인식할 수 있습니다. 이는 파서 구현에 중요한 의미를 가집니다.

# 구문 분석 (Syntax Analysis)

## 파서의 역할

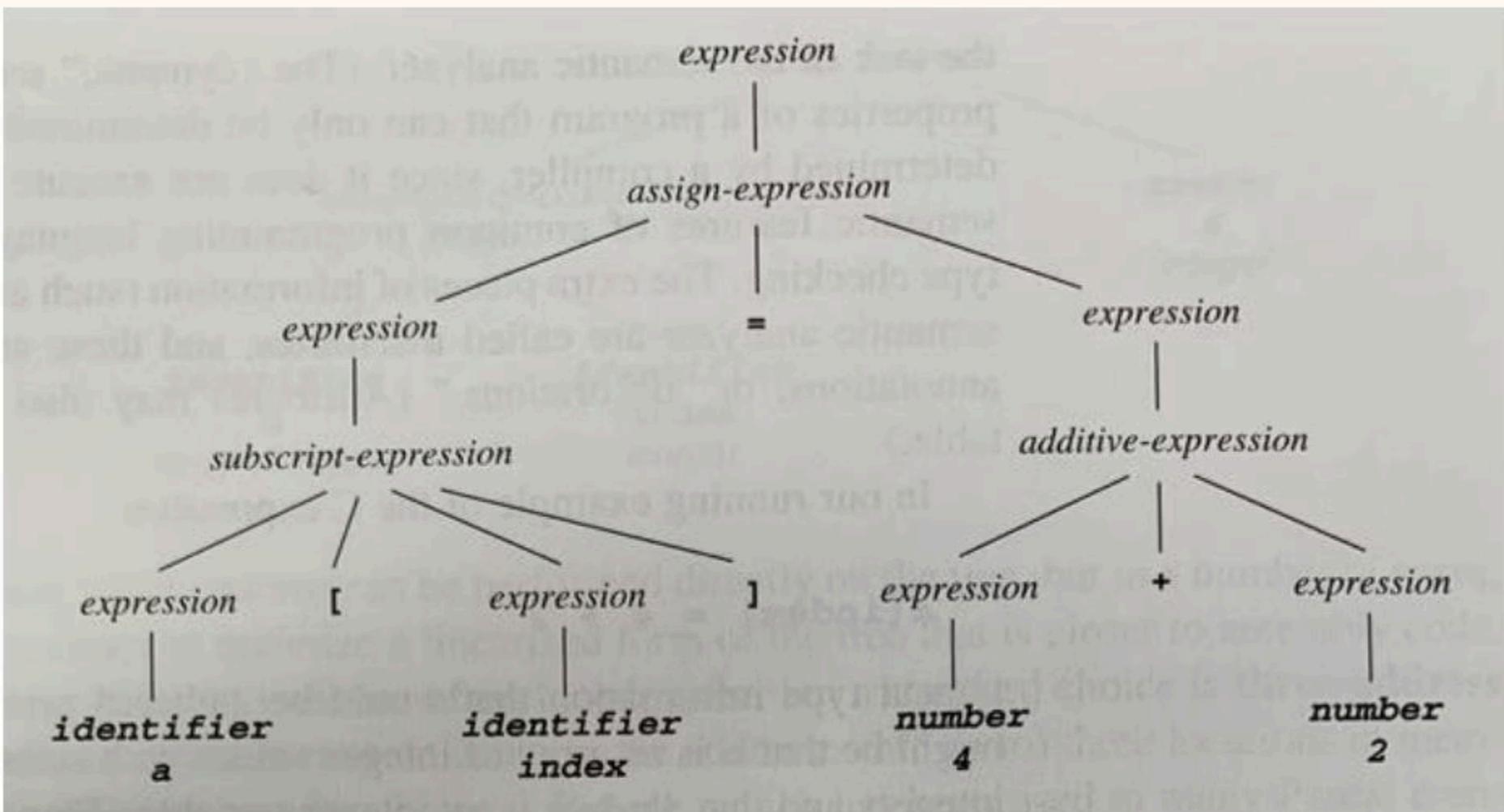
컴파일러의 2단계로, 프로그램의 구조를 결정합니다(문법을 통해 토큰을 그룹화).

문맥 자유 문법  $G = (VN, VT, P, S)$ 가 주어졌을 때, 문자열  $w$ 를 파싱한다는 것은  $w \in L(G)$ 인지 확인하고, 그렇다면  $w$ 에 대한 유도( $S \Rightarrow^* w$ )를 생성하는 것을 의미합니다.

## 파스 트리

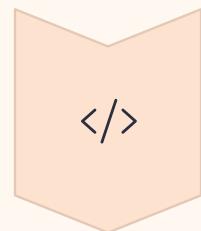
이 유도에 해당하는 파스 트리를 구성합니다.

- 내부 노드: 표현하는 구조의 이름으로 레이블됨
- 리프 노드: 스캐너에서 온 토큰



Parse tree / Syntax tree

# 하향식 파싱 (Top-down Parsing)



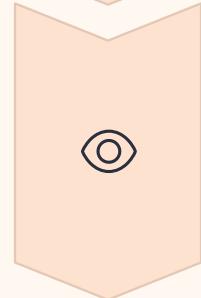
## 하향식 파싱의 정의

토큰의 입력 문자열을 좌측 유도의 단계를 추적하여 파싱합니다.



## 재귀 하강 파싱

백트래킹을 사용하는 방식과 예측 파서 방식이 있습니다.



## 예측 파서

하나 이상의 선행 토큰을 사용하여 입력 문자열의 다음 구조를 예측하려고 시도합니다.

LL(1), LL(k) 등의 방식이 있습니다.

# 재귀 하강 파싱 (Recursive Descent Parsing)

## 1 기본 개념

비단말 A에 대한 문법 규칙을 A를 인식할 프로시저로 보는 방식입니다.

## 2 파싱 과정

최상위 비단말(S)부터 시작합니다.  
순서대로 시작 기호에 대한 규칙을 시도합니다.  
각 단계에서 사용할 생성 규칙에 대한 많은 선택지가 있습니다.  
백트래킹: 잘못된 선택을 취소합니다.

## 3 예측적 파싱

단일 선행 토큰을 기반으로 결정을 내릴 수 있습니다.

# 재귀 하강 파싱 예제

## 문맥 자유 문법

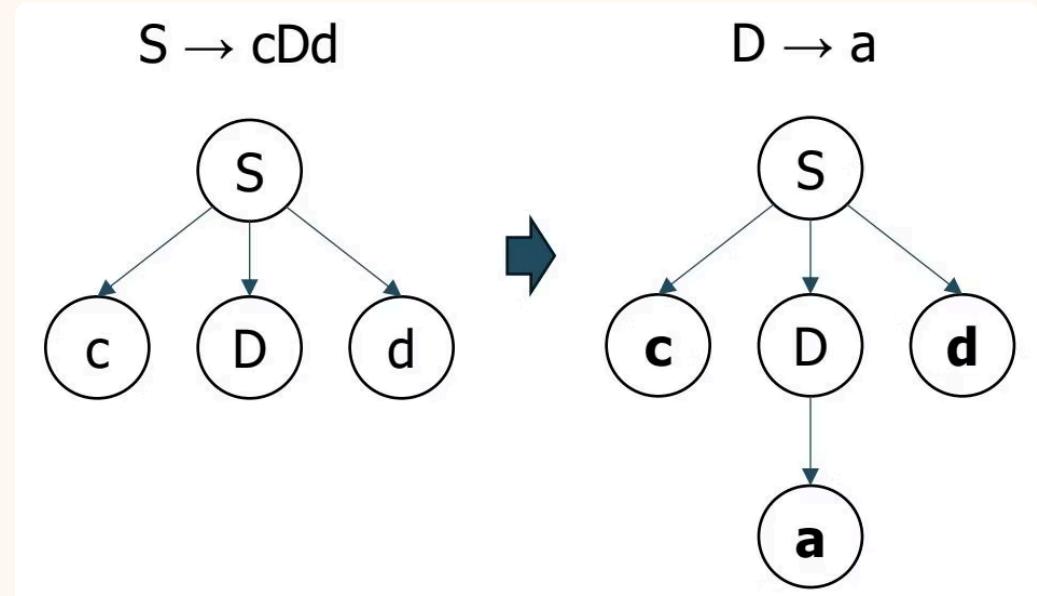
- $S \rightarrow cDd$
- $D \rightarrow a \mid aE$
- $E \rightarrow bb \mid bd$

입력 문자열: **cabdd**

## 첫 번째 시도

$S \rightarrow cDd$ 를 적용하고,  $D \rightarrow a$ 를 선택하면 결과는 **cad**가 됩니다.

이는 입력 문자열 **cabdd**와 일치하지 않으므로 실패합니다.



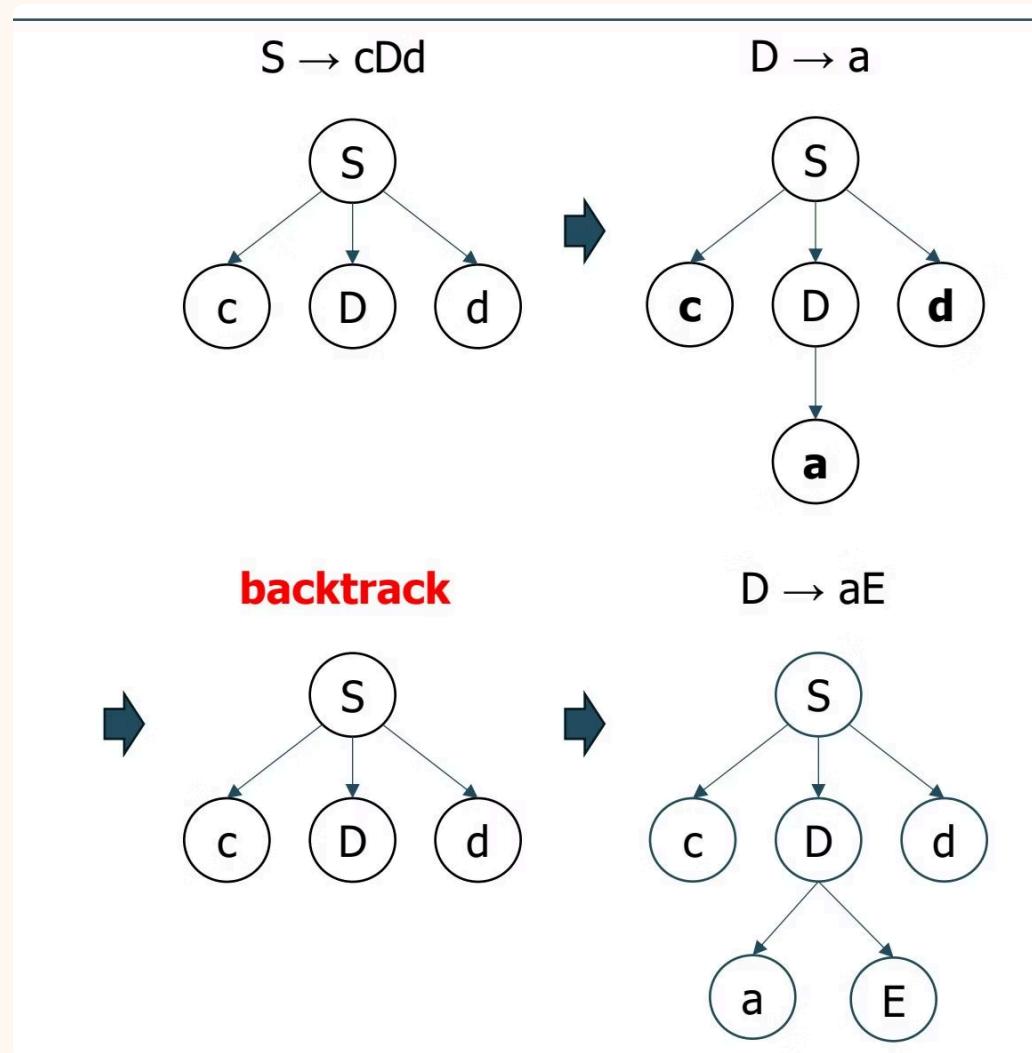
# 백트래킹을 통한 재귀 하강 파싱

## 두 번째 시도

$S \rightarrow cDd$ 를 적용하고,  $D \rightarrow aE$ 를 선택합니다.

그 다음  $E \rightarrow bb$ 를 선택하면 결과는 **cabbd**가 됩니다.

이는 입력 문자열 **cabdd**와 일치하지 않으므로 다시 백트래킹합니다.

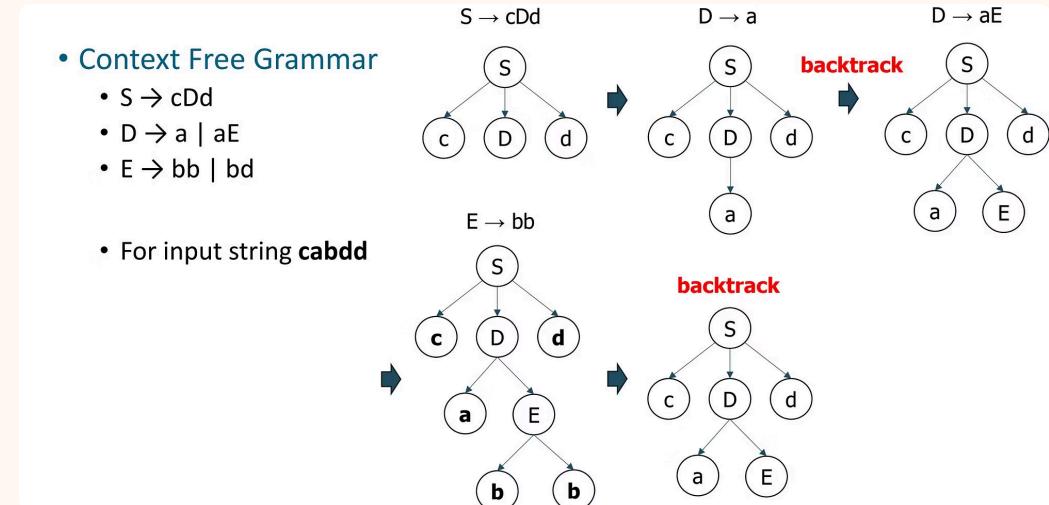


## 세 번째 시도

$S \rightarrow cDd$ 를 적용하고,  $D \rightarrow aE$ 를 선택합니다.

그 다음  $E \rightarrow bd$ 를 선택하면 결과는 **cabdd**가 됩니다.

이는 입력 문자열 **cabdd**와 일치하므로 성공합니다.



# 백트래킹 요약

## 장점

- 간단하고 일반적인 파싱 전략
- 자동으로 수행 가능

## 단점

- 백트래킹으로 인해 비효율적
- 문법을 제한하여 백트래킹을 제거할 수 있음

이 수업에서는 예측적 파서가 더 효율적이기 때문에 백트래킹을 제거하는 방법은 다루지 않습니다.

# 예측적 파서 (Predictive Parser)



## 예측 기능

파서는 다음 몇 개의 토큰을 보고 어떤 생성 규칙을 사용할지 "예측"할 수 있습니다.

백트래킹이 필요 없습니다.

파싱 테이블을 기반으로 합니다.



## LL(k) 문법

예측적 파서는 LL(k) 문법을 수용합니다.

- 첫 번째 L은 "왼쪽에서 오른쪽으로" 입력을 스캔함을 의미
- 두 번째 L은 "좌측 유도"를 의미
- k는 "k개의 선행 토큰을 기반으로 예측"함을 의미

실제로는 LL(1)이 주로 사용됩니다.

# LL(1) 예측적 파서

## 명시적 스택과 파싱 테이블

파싱을 수행하기 위해 명시적 스택과 파싱 테이블을 사용합니다.

문법  $G = (\{S\}, \{(, )\}, P, S)$ 를 고려해봅시다.

- $P: S \rightarrow (S)S \mid \epsilon$
- 균형 잡힌 괄호 문자열을 생성합니다.

## 문자열 "("에 대한 하향식 파서의 동작

파싱 스택	입력	동작
\$ S	( ) \$	$S \rightarrow (S)S$
\$ S ) S (	( ) \$	match
\$ S ) S	) \$	$S \rightarrow \epsilon$
\$ S )	) \$	match
\$ S	\$	$S \rightarrow \epsilon$
\$	\$	accept

Predictive Parser LL(1)

lookahead

# 하향식 파싱의 단계

초기화

시작 기호를 스택에 푸시합니다.

파싱 과정

일련의 동작 후, 스택과 입력이 모두 비어있으면 입력 문자열을 수용합니다.

스택 최상위에 따른 동작

비단말: 문법 선택  $A \rightarrow \alpha$ 를 사용하여 스택 최상위의 비단말  $A$ 를 문자열  $\alpha$ 로 대체합니다.

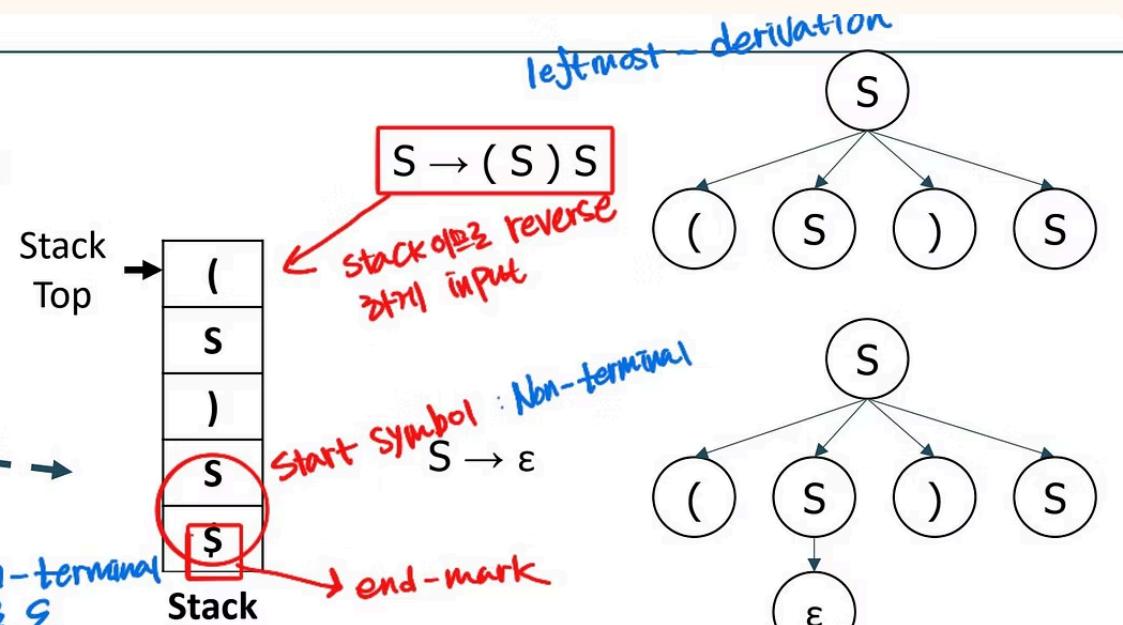
단말(토큰): 스택 최상위의 토큰과 다음 입력 토큰을 일치시킵니다.

# LL(1) 예측적 파서의 구조

- List of actions and left derivation



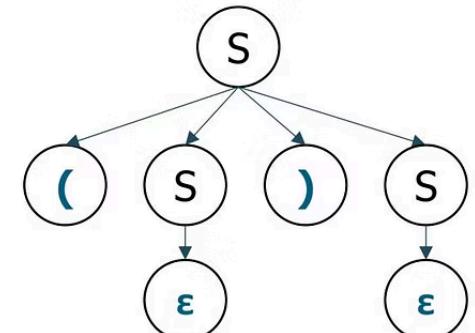
	Parsing stack	Input	Action
	location	terminal	
1	\$ S	( ) \$	$S \rightarrow (S)S$
2	\$ S ) S (	( ) \$	match
3	\$ S ) S	) \$	$S \rightarrow \epsilon$
4	\$ S )	) \$	match
5	\$ S	\$	$S \rightarrow \epsilon$
6	\$	\$	accept



Action of Non-terminal의 Production은 어떻게 정의?  
→ Parsing table이 사용된다.

- Generation actions: **leftmost derivation**

$$\begin{aligned}
 \bullet S \Rightarrow (S)S & [S \rightarrow (S)S] \\
 \Rightarrow ( )S & [S \rightarrow \epsilon] \\
 \Rightarrow ( ) & [S \rightarrow \epsilon]
 \end{aligned}$$



스택 최상위에 비단말 A가 있을 때, A에 대한 어떤 문법 규칙 선택을 사용할지 결정해야 합니다. 이는 파싱 테이블을 사용하여 수행됩니다.

## 파싱 테이블

문법 규칙 선택은 파싱 테이블을 구성하여 수행됩니다.

LL(1) 파싱 테이블  $M[N, T]$ 는 비단말과 단말로 인덱싱된 2차원 배열입니다.

- N: 문법의 비단말 집합(스택 최상위)
- T: 문법의 단말 집합(현재 입력)

# LL(1) 파싱 테이블 예제

M[N, T]	(	)	\$
S	$S \rightarrow (S)S$	$S \rightarrow \epsilon$	$S \rightarrow \epsilon$

이 파싱 테이블을 사용하여 문자열 "( )"를 파싱해보겠습니다.

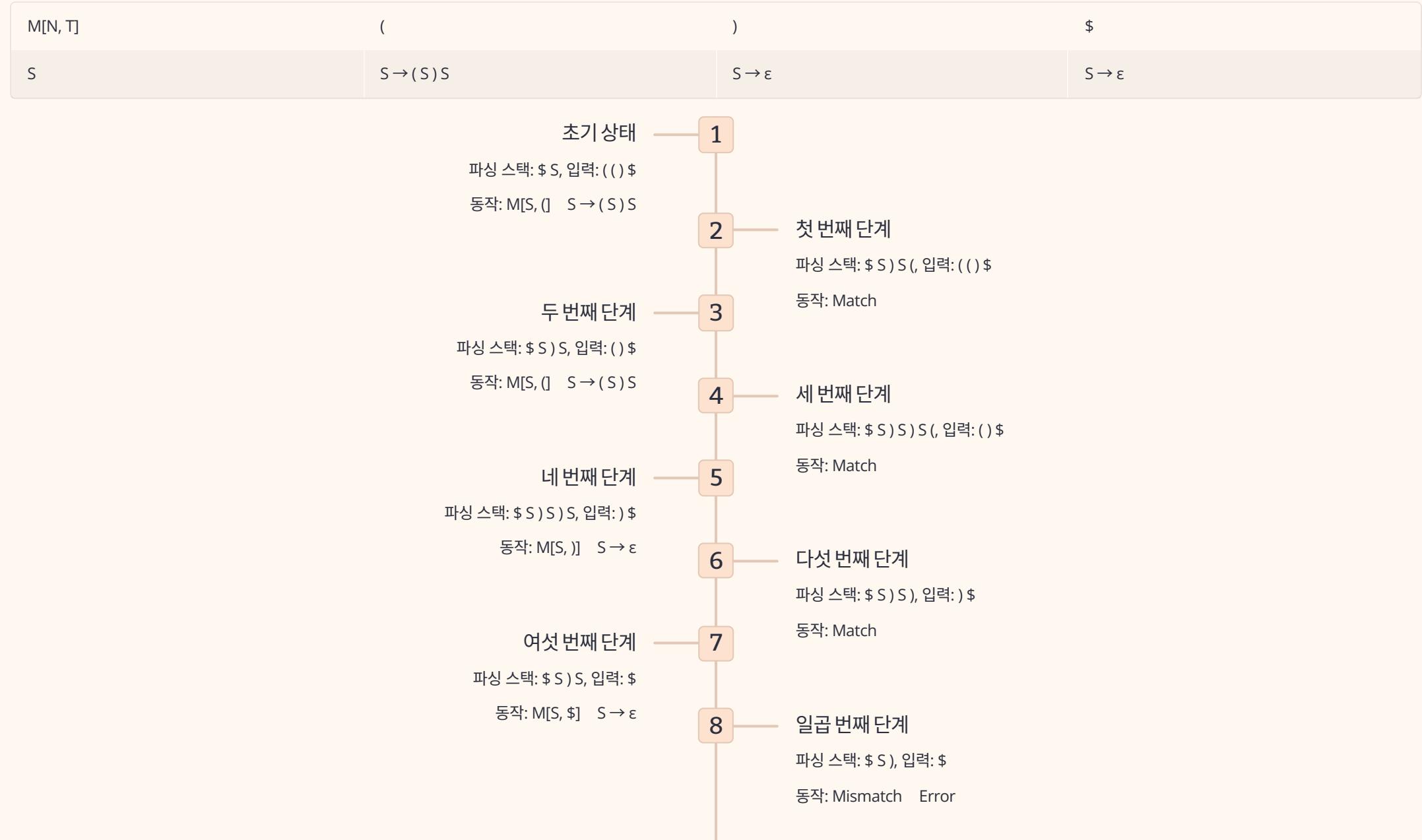
## 스택 최상위에 따른 동작

- 비단말: 문법 선택  $A \rightarrow a$ 를 사용하여 스택 최상위의 비단말 A를 문자열 a로 대체합니다.
- 단말: 스택 최상위의 토큰과 다음 입력 토큰을 일치시킵니다.

## 파싱 과정

파싱 스택	입력	동작
\$ S	( ) \$	M[S, ()] $S \rightarrow (S)S$
\$ S ) S (	( ) \$	Match
\$ S ) S	) \$	M[S, ]) $S \rightarrow \epsilon$
\$ S )	) \$	Match
\$ S	\$	M[S, \$] $S \rightarrow \epsilon$
\$	\$	Accept (empty)

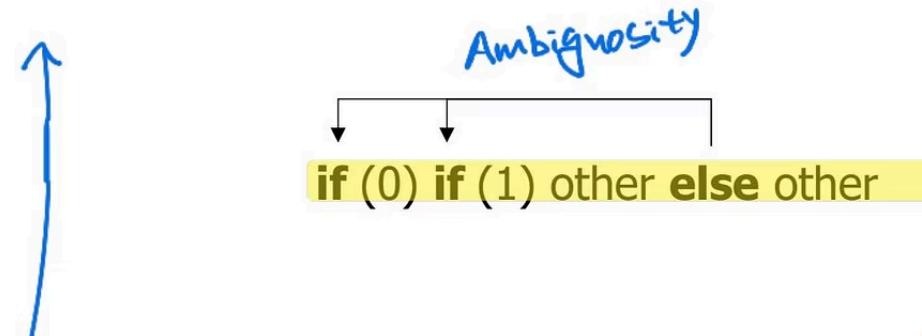
## 문자열 "()"에 대한 LL(1) 파싱



# if 문에 대한 LL(1) 파싱 테이블

- Grammar for if-statements  $\Rightarrow$  Ambiguous Grammar

- statement  $\rightarrow$  if-stmt | other
- if-stmt  $\rightarrow$  if ( exp ) statement else-part
- else-part  $\rightarrow$  else statement |  $\epsilon$
- exp  $\rightarrow$  0 | 1



예측적 파서는 결정적이므로 모호한 문법을 처리할 수 없습니다.

M[N, T]	if	other	else	0	1	\$
statement	statement $\rightarrow$ if-stmt	statement $\rightarrow$ other				
if-stmt	if-stmt $\rightarrow$ if ( exp ) statement else-part					
else-part			else-part $\rightarrow$ else statement			else-part $\rightarrow$ $\epsilon$
exp				exp $\rightarrow$ 0	exp $\rightarrow$ 1	

# 예측적 파서의 전처리

## 1 문법 변환의 필요성

예측적 파서를 구성하기 위해 문법을 변환해야 할 수 있습니다.

## 2 좌재귀 제거

좌재귀를 제거하여 무한 재귀를 방지합니다.

## 3 좌인수분해 적용

공통 접두사를 가진 규칙을 분해합니다.

## 4 파싱 테이블 구성

모호하지 않고 결정적인 파싱 테이블을 만들기 위함입니다.

문법이 모호하지 않더라도, 파싱 테이블을 구성하기에 완벽하지 않을 수 있습니다.

특정 형태의 문법이 필요합니다.

# 좌재귀 (Left Recursion)

## 좌재귀의 정의

비단말 A가  $A \Rightarrow^+ A\alpha$  ( $\alpha \in V^*$ )를 만족하면 A는 좌재귀입니다.

좌재귀 비단말은 하향식 결정적 파서가 무한 루프에 빠지게 합니다(무한 재귀).

## 좌재귀의 종류

- 직접 좌재귀(Immediate left recursive):  $A \rightarrow A\alpha$
- 간접 좌재귀(Indirect left recursive):  $A \Rightarrow^+ A\alpha$

# 좌재귀 제거

직접 좌재귀를 제거하는 방법:

- $A \rightarrow A\alpha \mid \beta$ , (the corresponding language is  $\beta\alpha^*$ )

- Rewrite rules using right-recursion  $A \rightarrow aA$

- $A \rightarrow \beta A'$

- $A' \rightarrow \alpha A' \mid \epsilon$  *matching 목적*

*추가적인 Non-terminal Rule 추가*

## Indirect left recursive

- $S \rightarrow Aa \mid b$

- $A \rightarrow [Ac \mid Sd \mid e]$

## Indirect left recursion

- $S \xrightarrow{+} Sda$   $[S \rightarrow Aa, A \rightarrow Sd]$

*Cycle을*

*끊는다.*

*Sd*

*$\rightarrow Aad \mid bd$*

*Topological order of nonterminals:  $S \rightarrow A$*

- $A \rightarrow [Ac \mid Aad \mid bd] \mid e$

## Rewrite rules using right-recursion

- $A \rightarrow bdA' \mid eA'$

- $A' \rightarrow cA' \mid adA' \mid \epsilon$

## 결과 확인

이제 A는 더 이상 좌재귀가 아니며, 원래 문법과 동일한 언어를 생성합니다.

## 원래 문법

$$A \rightarrow Aa \mid \beta$$

여기서 A는 비단말, a와 β는 문자열입니다.

## 좌재귀 제거 후

$$A \rightarrow \beta A'$$

$$A' \rightarrow aA' \mid \epsilon$$

새로운 비단말 A'를 도입하여 좌재귀를 제거합니다.

# 좌인수분해 (Left Factoring)

## 좌인수분해의 필요성

두 개 이상의 문법 규칙이 공통 접두사 문자열을 공유할 때 필요합니다.

예:  $A \rightarrow ab \mid ac$

## 해결책

왼쪽에 있는  $\alpha$ (공통 접두사 문자열)를 "인수분해"하고 규칙을 두 개의 규칙으로 다시 작성합니다.

$A \rightarrow \alpha A'$

$A' \rightarrow b \mid c$

# 좌인수분해 예제

## 예제 1

$$G = (\{S, A\}, \{a, b, c, d\}, P, S)$$

$$P: S \rightarrow cAd$$

$$A \rightarrow a \mid ab$$

공통 접두사 찾기:  $A \rightarrow a \mid ab \rightarrow$  인수분해

$$S \rightarrow cAd$$

$$A \rightarrow aA'$$

$$A' \rightarrow \varepsilon \mid b$$

## 예제 2

$$G = (\{S, E\}, \{i, t, e, a, b\}, P, S)$$

$$P: S \rightarrow iEtS \mid iEtSeS \mid a$$

$$E \rightarrow b$$

공통 접두사 찾기:  $S \rightarrow iEtS \mid iEtSeS \rightarrow$  인수분해

$$S \rightarrow iEtSS' \mid a$$

$$S' \rightarrow \varepsilon \mid eS$$

$$E \rightarrow b$$

# LL(1) 파싱 테이블

## 1 파싱 테이블 구성 알고리즘

LL(1) 파싱 테이블을 구성하기 위한 알고리즘이 필요합니다.

### FIRST 집합

문맥 자유 문법  $G = (VN, VT, P, S)$ 에서,

모든 비단말  $A \in VN$ 에 대해,

$$FIRST(A) = \{b \in VT \cup \{\epsilon\} \mid A \xrightarrow{*} b\beta, \beta \in V^*\}$$

비단말  $A$ 에서 유도될 수 있는 문자열의 시작 부분에 나타날 수 있는 단말 기호의 집합입니다.

## 2 FIRST와 FOLLOW 집합

두 집합을 정의하여 파싱 테이블을 구성합니다.

### FOLLOW 집합

문맥 자유 문법  $G = (VN, VT, P, S)$ 에서,

모든 비단말  $A \in VN$ 에 대해,

$$FOLLOW(A) = \{a \in VT \cup \{\$\} \mid S \xrightarrow{*} aAa\beta, a, \beta \in V^*\}$$

비단말  $A$  바로 뒤에 나타날 수 있는 단말 기호의 집합입니다.

\$: 입력 문자열의 끝을 나타내는 표시

# FIRST 집합 예제

## 예제 1

$$G = (\{S, C, D\}, \{a, b, c, d\}, P, S)$$

$$P: S \rightarrow C \mid D$$

$$C \rightarrow aC \mid b$$

$$D \rightarrow cD \mid d$$

$$\text{FIRST}(C) = \{a, b\}$$

$$\text{FIRST}(D) = \{c, d\}$$

$$\text{FIRST}(S) = \{a, b, c, d\}$$

## 예제 2

$$G = (\{T, T', E, E', F\}, \{+, *, (, ), a\}, P, E)$$

$$P: E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \varepsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \varepsilon$$

$$F \rightarrow (E) \mid a$$

$$\text{FIRST}(E) = \{(, a\}$$

$$\text{FIRST}(E') = \{+, \varepsilon\}$$

$$\text{FIRST}(T) = \{(, a\}$$

$$\text{FIRST}(T') = \{*, \varepsilon\}$$

$$\text{FIRST}(F) = \{(, a\}$$

# FIRST 집합 계산

## 1 단말에 대한 FIRST

단말  $a \in VT$ 에 대해,  $\text{FIRST}(a) = \{a\}$

## 2 비단말에 대한 FIRST 계산

$a$ 가  $\text{FIRST}(A)$ 에 속하는 경우는 다음과 같습니다:

- $a$ 가  $\text{INITFIRST}(A) = \{a \in VT \mid A \rightarrow a\alpha \in P, \alpha \in V^*\}$ 에 속하거나
- $a$ 가  $\{a \mid a \in \text{FIRST}(B), A \rightarrow Ba \in P, \alpha \in V^*, A \neq B\}$ 에 속하는 경우

## 3 재귀 방정식

각 비단말  $A$ 에 대해,  $\text{FIRST}(A) = \text{INITFIRST}(A) \cup \{\text{FIRST}(B) \mid A \rightarrow Ba \in P, A \neq B\}$

# FIRST 집합 계산 예제

문법

$$G = (\{S, C, D\}, \{a, b, c, d\}, P, S)$$

$$P: S \rightarrow C \mid D$$

$$C \rightarrow aC \mid b$$

$$D \rightarrow cD \mid d$$

FIRST 계산

$$\text{FIRST}(S) = \text{INITFIRST}(S) \cup \{\text{FIRST}(C) \mid S \rightarrow C \in P, S \neq C\} \cup \{\text{FIRST}(D) \mid S \rightarrow D \in P, S \neq D\}$$

$$\text{INITFIRST}(S) = \{\}$$

$$\text{FIRST}(C) = \text{INITFIRST}(C) = \{a, b\}$$

$$\text{FIRST}(D) = \text{INITFIRST}(D) = \{c, d\}$$

$$\text{FIRST}(S) = \{a, b, c, d\}$$

*• Update' 양을 따라'*

Grammar rule	Pass 1	Pass 2
$S \rightarrow C$		$\text{First}(S) = \{a, b\}$
$S \rightarrow D$		$\text{First}(S) = \{a, b, c, d\}$
$C \rightarrow aC$	$\text{First}(C) = \{a\}$	$\text{First}(C) = \{a, b\}$
$C \rightarrow b$		
$D \rightarrow cD$	$\text{First}(D) = \{c\}$	
$D \rightarrow d$	$\text{First}(D) = \{c, d\}$	

# FIRST 집합 계산 예제 2

문법 규칙	Pass 1	Pass 2	Pass 3
$E \rightarrow TE'$	$\text{FIRST}(E) = \{(), a\}$		
$E' \rightarrow +TE'$	$\text{FIRST}(E') = \{+\}$		
$E' \rightarrow \epsilon$		$\text{FIRST}(E') = \{+, \epsilon\}$	
$T \rightarrow FT'$	$\text{FIRST}(T) = \{(), a\}$		
$T' \rightarrow *FT'$	$\text{FIRST}(T') = \{*\}$		
$T' \rightarrow \epsilon$		$\text{FIRST}(T') = \{*, \epsilon\}$	
$F \rightarrow (E)$	$\text{FIRST}(F) = \{\()\}$		
$F \rightarrow a$		$\text{FIRST}(F) = \{(), a\}$	

# FOLLOW 집합

## FOLLOW 집합 정의

문맥 자유 문법  $G = (VN, VT, P, S)$ 에서,

모든 비단말  $A \in VN$ 에 대해,

$$FOLLOW(A) = \{a \in VT \cup \{\$\} \mid S \xrightarrow{*} \alpha A a \beta, \alpha, \beta \in V^*\}$$

비단말  $A$  바로 뒤에 나타날 수 있는 단말 기호의 집합입니다.

\$: 입력 문자열의 끝을 나타내는 표시

$A$ 가 시작 기호인 경우, \$는 FOLLOW( $A$ )에 포함됩니다.

## FOLLOW 집합 예제

$$G = (\{S, C, D\}, \{a, b, c, d\}, P, S)$$

$$P: S \rightarrow C \mid D$$

$$C \rightarrow aC \mid b$$

$$D \rightarrow cD \mid d$$

$$FOLLOW(S) = \{\$\}$$

$$FOLLOW(C) = \{\$\}$$

$$S \rightarrow C \rightarrow FOLLOW(S) = FOLLOW(C)$$

$$FOLLOW(D) = \{\$\}$$

$$S \rightarrow D \rightarrow FOLLOW(S) = FOLLOW(D)$$

# FOLLOW 집합 계산

## 1 FOLLOW 집합 계산

a가 FOLLOW(A)에 속하는 경우는 다음과 같습니다:

- a가  $\text{INITFOLLOW}(A) = \{a \in VT \mid B \rightarrow \alpha AX\beta, a \in \text{FIRST}(X), \alpha, \beta \in V^*\}$ 에 속하거나
- a가  $\{a \mid a \in \text{FOLLOW}(B), B \rightarrow \alpha A \in P, \alpha \in V^*, A \neq B\}$ 에 속하는 경우

## 2 재귀 방정식

각 비단말 A에 대해,

$$\text{FOLLOW}(A) = \text{INITFOLLOW}(A) \cup \{\text{FOLLOW}(B) \mid B \rightarrow \alpha A \in P, \alpha \in V^*, A \neq B\}$$

# FOLLOW 집합 계산 예제 1

## 문법

$$G = (\{S, C, D\}, \{a, b, c, d\}, P, S)$$

$$P: S \rightarrow C \mid D$$

$$C \rightarrow aC \mid b$$

$$D \rightarrow cD \mid d$$

$$\text{FIRST}(C) = \{a, b\}$$

$$\text{FIRST}(D) = \{c, d\}$$

$$\text{FIRST}(S) = \{a, b, c, d\}$$

## FOLLOW 계산

문법 규칙	Pass 1	
$S \rightarrow C$	$\text{FOLLOW}(S) = \{\$\} \text{ (시작 기호)}$	$\text{FOLLOW}(C) = \{\$\}$ $(\text{FOLLOW}(S) \subseteq \text{FOLLOW}(C))$
$S \rightarrow D$	$\text{FOLLOW}(D) = \{\$\}$ $(\text{FOLLOW}(S) \subseteq \text{FOLLOW}(D))$	
$C \rightarrow aC$	pass	
$C \rightarrow b$		
$D \rightarrow cD$	pass	
$D \rightarrow d$		

# FOLLOW 집합 계산 예제 2

- $G = (\{T, T', E, E', F\}, \{+, *, (, ), a\}, P, E)$
- $P: E \rightarrow TE'$
- $E' \rightarrow +TE' \mid \epsilon$
- $T \rightarrow FT'$
- $T' \rightarrow *FT' \mid \epsilon$
- $F \rightarrow (E) \mid a$
- $FIRST(E) = \{(, a\}$
- $FIRST(E') = \{+, \epsilon\}$
- $FIRST(T) = \{(, a\}$
- $FIRST(T') = \{*, \epsilon\}$
- $FIRST(F) = \{(, a\}$

↑  
FIRST(A) contains  $\{\epsilon\}$  means that A could be derived into  $\epsilon$   
 $(A \xrightarrow{*} \epsilon)$

Grammar rule	Pass 1
$E \rightarrow TE'$	$FOLLOW(E) = \{\$\}$ (start symbol) $FOLLOW(T) = \{+\}$ ( $FIRST(E') - \{\epsilon\} = \{+\}$ ) $FOLLOW(E') = \{\$\}$ ( $FOLLOW(F) \subseteq FOLLOW(E')$ ) $FOLLOW(T) = \{\$, +\}$ ( $FOLLOW(E) \subseteq FOLLOW(T)$ because $FIRST(E) \text{ contains } \{\epsilon\}$ )
$E' \rightarrow +TE'$	$FOLLOW(T) = \{\$, +\}$
$E' \rightarrow \epsilon$	
$T \rightarrow FT'$	$FOLLOW(F) = \{*\}$ ( $FIRST(T') - \{\epsilon\} = \{*\}$ ) INIT FOLLOW $FOLLOW(T') = \{\$, +\}$ ( $FOLLOW(T) \subseteq FOLLOW(T')$ ) $FOLLOW(F) = \{\$, *, +\}$ ( $FOLLOW(T) \subseteq FOLLOW(F)$ because $FIRST(T) \text{ contains } \{\epsilon\}$ )
$T' \rightarrow *FT'$	$FOLLOW(F) = \{\$, *, +\}$
$T' \rightarrow \epsilon$	
$F \rightarrow (E)$	$FOLLOW(E) = \{\$, )\}$ ( $FIRST(()) = \{\)\}$ )
$F \rightarrow a$	

문법  $G = (\{T, T', E, E', F\}, \{+, *, (, ), a\}, P, E)$ 에 대한 FOLLOW 집합 계산 과정입니다. 이 과정은 여러 패스를 통해 각 비단말의 FOLLOW 집합을 점진적으로 구축합니다.

## FOLLOW 집합 계산 예제 2(계속)

### 문법

$$G = (\{T, T', E, E', F\}, \{+, *, (, ), a\}, P, E)$$

$$P: E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \epsilon$$

$$F \rightarrow (E) \mid a$$

$$\text{FIRST}(E) = \{(, a\}$$

$$\text{FIRST}(E') = \{+, \epsilon\}$$

$$\text{FIRST}(T) = \{(, a\}$$

$$\text{FIRST}(T') = \{*, \epsilon\}$$

$$\text{FIRST}(F) = \{(, a\}$$

### FOLLOW 계산 (Pass 1)

문법 규칙	Pass 1		
$E \rightarrow TE'$	$\text{FOLLOW}(E) = \{\$\}$	$\text{FOLLOW}(T) = \{+\}$	$\text{FOLLOW}(E') = \{\$\}$
$E' \rightarrow +TE'$		$\text{FOLLOW}(T) = \{\$, +\}$	
$E' \rightarrow \epsilon$			
$T \rightarrow FT'$		$\text{FOLLOW}(F) = \{*\}$	$\text{FOLLOW}(T') = \{\$, +\}$
$T' \rightarrow *FT'$		$\text{FOLLOW}(F) = \{\$, *, +\}$	
$T' \rightarrow \epsilon$			
$F \rightarrow (E)$	$\text{FOLLOW}(E) = \{\$, )\}$		
$F \rightarrow a$			

# FOLLOW 집합 계산 예제 2(최종)

## Pass 2

문법 규칙	Pass 2	
$E \rightarrow TE'$	$\text{FOLLOW}(E') = \{\$, )\}$	$\text{FOLLOW}(T) = \{\$, , +\}$
$E' \rightarrow +TE'$	$\text{FOLLOW}(T) = \{\$, , +\}$	
$E' \rightarrow \epsilon$		
$T \rightarrow FT'$	$\text{FOLLOW}(T') = \{\$, , +\}$	$\text{FOLLOW}(F) = \{\$, , *, +\}$
$T' \rightarrow *FT'$	$\text{FOLLOW}(F) = \{\$, , *, +\}$	
$T' \rightarrow \epsilon$		
$F \rightarrow (E)$	$\text{FOLLOW}(E) = \{\$, )\}$	
$F \rightarrow a$		

## 최종 결과

$$\text{FOLLOW}(E) = \{\$, )\}$$

$$\text{FOLLOW}(E') = \{\$, )\}$$

$$\text{FOLLOW}(T) = \{\$, , +\}$$

$$\text{FOLLOW}(T') = \{\$, , +\}$$

$$\text{FOLLOW}(F) = \{\$, , +, *\}$$

# LL(1) 파싱 테이블 구성

## 1 파싱 테이블 구성 알고리즘

각 비단말 A와 생성 규칙  $A \rightarrow a$ 에 대해 다음 단계를 반복합니다:

## 2 FIRST( $\alpha$ )의 단말 기호 처리

FIRST( $\alpha$ )의 각 단말 기호  $a$ 에 대해,  $A \rightarrow a$ 를  $M[A, a]$ 에 추가합니다.

## 3 $\epsilon$ 가 FIRST( $\alpha$ )에 있는 경우

FOLLOW(A)의 각 요소  $a$ (단말 기호 또는  $\$$ )에 대해,  $A \rightarrow a$ 를  $M[A, a]$ 에 추가합니다.

# LL(1) 파싱 테이블 구성 예제

문법

$$G = (\{S\}, \{(, )\}, P, S)$$

$$P: S \rightarrow (S)S \mid \epsilon$$

$$\text{FIRST}(S) = \{ (, \epsilon \}$$

$$\text{FOLLOW}(S) = \{ \$, ) \}$$

파싱 테이블 구성

$$S \rightarrow (S)S$$

- FIRST((S)S) = FIRST(()) = {}
- M[S, ()] =  $S \rightarrow (S)S$

$$S \rightarrow \epsilon$$

- FIRST( $\epsilon$ ) = {  $\epsilon$  }
- FOLLOW(S) = { \$, ) }
- M[S, \$] =  $S \rightarrow \epsilon$
- M[S, ]) =  $S \rightarrow \epsilon$

M[N, T]	(	)	\$
S	$S \rightarrow (S)S$	$S \rightarrow \epsilon$	$S \rightarrow \epsilon$

# LL(1) 파싱 테이블 구성 예제 2

$$G = (\{T, T', E, E', F\}, \{+, *, (, ), a\}, P, E)$$

$$P: E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \epsilon$$

$$F \rightarrow (E) \mid a$$

M[N, T]	*	(	)	a	\$
E		$E \rightarrow TE'$		$E \rightarrow TE'$	
E'			$E' \rightarrow \epsilon$		$E' \rightarrow \epsilon$
T		$T \rightarrow FT'$		$T \rightarrow FT'$	
T'	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$		$T' \rightarrow \epsilon$
F		$F \rightarrow (E)$		$F \rightarrow a$	

# 선행 토큰 확장: LL(k) 파서

## LL(k) 파서의 정의

$\text{FIRST}_k(A) = \{w_k \mid A \Rightarrow^+ w\}$ , 여기서  $w_k$ 는  $w$ 의 첫  $k$ 개 토큰입니다.

$\text{FOLLOW}_k(A) = \{w_k \mid S \Rightarrow^+ aAw\}$

## LL(k) 파서의 특징

- 파싱 테이블이 훨씬 커집니다.
- 열의 수가  $k$ 에 따라 지수적으로 증가합니다.
- 더 긴 선행 토큰과 더 큰 테이블로 인해 속도가 느려집니다.
- 변환 없이 더 많은 문법을 표현할 수 있습니다.
- 덜 일반적으로 지원됩니다(LL(1)은 간단하고 모호하지 않은 문법에 사용됨).

기호 군집이 약상 가정에 따른 문제점

$M[N, T]$

( )

(( ))

(( ))

\$

# 요약

## 푸시다운 오토마타

- 스택을 사용하여 문맥 자유 문법을 인식
- 결정적 푸시다운 오토마타(DPDA)
- 비결정적 푸시다운 오토마타(NPDA)

## 하향식 파싱

- 백트래킹: 비효율적
- 예측적 파서
- 파싱 테이블을 기반으로 생성 규칙 예측
- FIRST / FOLLOW 계산

- LL(1) / LL(k) parser