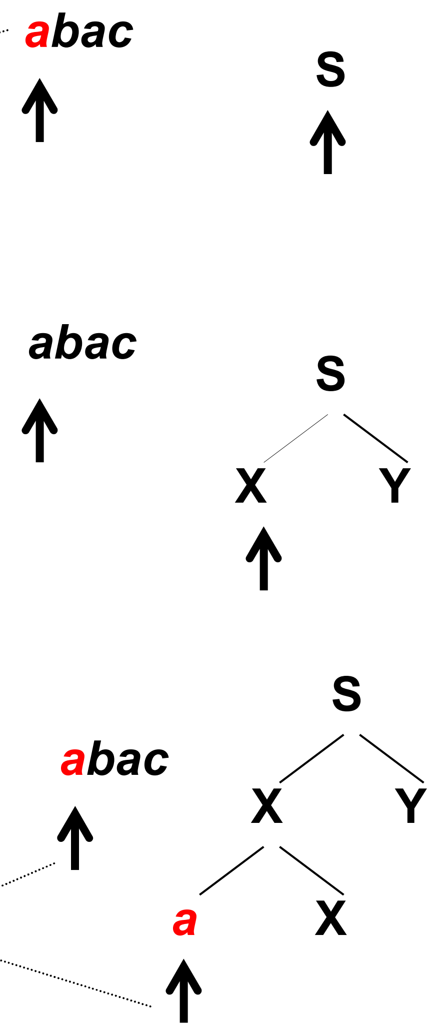
Chapter 6

하향식 파싱 알고리즘 - **Part II**

Top-Down Parsing (1/5)

lookahead Symbol을 읽어 옴

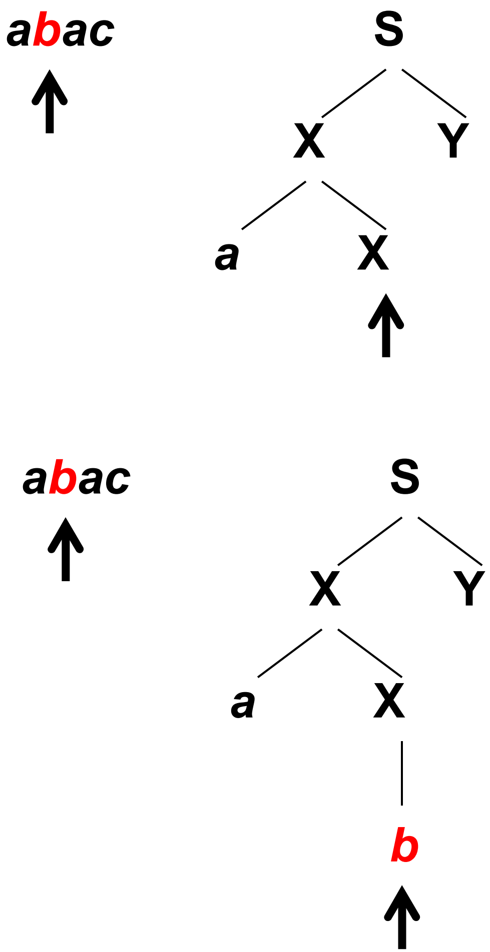


P :1. S → XY
2. X → a X
3. X → b
4. Y → aY
5. Y → c

Matching !!!
advance the pointers

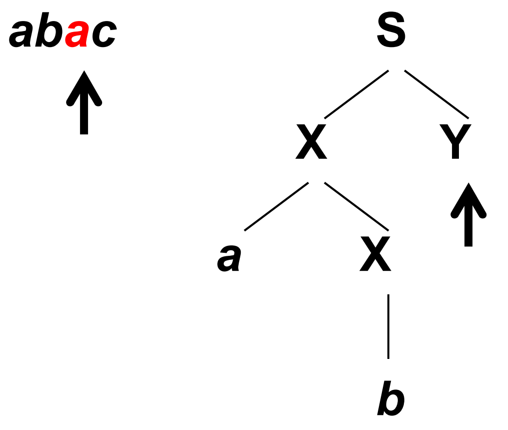
Top-Down Parsing (2/5)

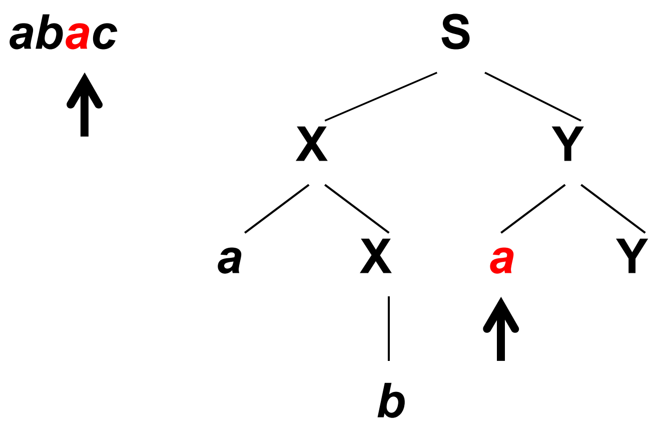
P :1. S → XY
2. X → a X
3. X → b
4. Y → aY
5. Y → c



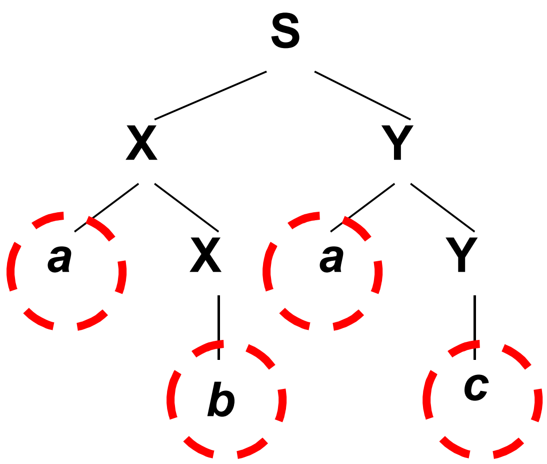
Top-Down Parsing (3/5)

P :1. S → XY
2. X → a X
3. X → b
4. Y → aY
5. Y → c





Top-Down Parsing (4/5)



Top-Down Parsing (5/5)

■ 좌 파스(*Left Parse*)

- 좌측 유도(*leftmost derivation*)를 적용
- Parsing 과정을 이해하기 쉬움

S ⇒ XY ⇒ *a*XY ⇒ *ab*Y ⇒ *aba*Y ⇒ *abac*, 즉 $S \overset{*}{\underset{lm}{\Longrightarrow}} abac$

1 ***2*** ***3*** ***4*** ***5***

Table of contents

Table of contents

- 좋은 구문 분석이란?
  - **Without backtracking**
  - **Deterministic parsing**
- Predict Set과 LL(1) 조건
- Recursive descent LL(1) Parsers
- Table-driven LL(1) Parsers

좋은 구문 분석이란

- 구문 분석은 입력의 일부(***lookahead***)만을 보고 문법의 생성 규칙 중 하나를 선택
  - 만약 생성 규칙을 잘못 선택했다면
    - 지금까지 진행했던 유도 과정을 취소하고, 직전 상태로 되돌아가야 (*backtracking*) 함

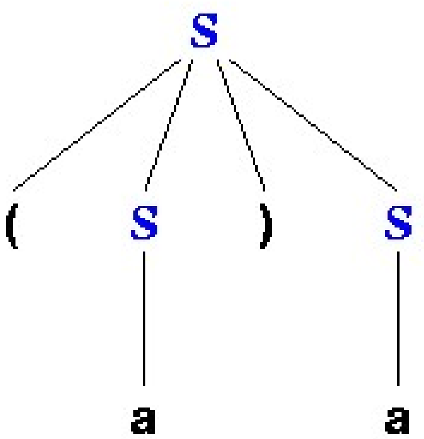
- 결정적(***deterministic***) 구문 분석
  - 입력 내용(*lookahead symbol*)과 현재 구문 분석 진행 상태 (*sentential form*)만을 갖고
    - **backtracking**이 발생하지 않도록
    - 한 번에 올바른 생성 규칙을 선택 !!!

결정적 구문 분석 예(1/2)

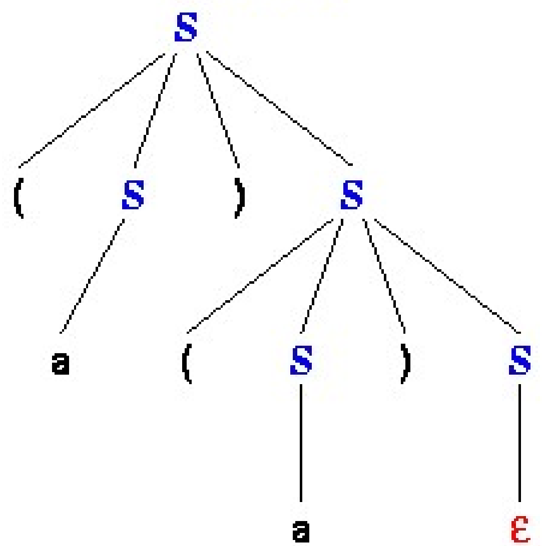
S 에 대해 3가지 생성 규칙 중 하나를 골라야 함

$$
\begin{aligned}
S &\longrightarrow (S)S \\
S &\longrightarrow a \\
S &\longrightarrow \epsilon
\end{aligned}
$$

(a)a



(a)(a)



결정적 구문 분석 예(2/2)

$$S \longrightarrow (S)S$$
$$S \longrightarrow a$$
$$S \longrightarrow \epsilon$$

First (S) = { (, a }
Follow (S) = {) , $ }

If *lookahead* = '('	**S derives (S) S**
If *lookahead* = 'a'	**S derives a**

If *lookahead* = ')'	**S derives λ**
If *lookahead* = '$'	**S derives λ**

Table of contents

- 좋은 구문 분석이란?
  - Without backtracking
  - Deterministic parsing
- **Predict Set과 LL(1) 조건**
- **Recursive descent LL(1) Parsers**
- **Table-driven LL(1) Parsers**

The Predict Set (1/2)

- **LL(k) parser :** n 개의 생성 규칙 중 하나를 선택하기 위해 한번에 k개의 **token**을 읽어 옴

- **The strategy for choosing productions**
  - $\mathrm{Pr}\,edict_k(p)$: the set of length-k token strings that predicts the application of rule p.
  - $\mathrm{Pr}\,edict(p)$: the set of length-1 strings

The Predict Set, P (2/2)

- **The predict set P**

$$P = \{ p \in \mathrm{Pr}\,oductionsFor(A) \mid a \in \mathrm{Pr}\,edict(p) \}$$

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$$

- **For the next input token *a***
  - if P is the ***empty*** set,
    - Syntax error is issued.
  - if P contains ***more than one*** production,
    - Nondeterminism would be required.
  - if P contains ***exactly one*** production,
    - The Leftmost parse can proceed deterministically.

Predict set : example

1	S	→	A	C	$	
2	C	→	c			
3		|	λ			
4	A	→	a	B	C	d
5		|	B	Q		
6	B	→	b	B		
7		|	λ			
8	Q	→	q			
9		|	λ			

First (Q) = { q }
Follow (Q) = { c, $ }

Predict(Q → q) = { q }
Predict(Q → λ) = { c, $ }

If ***lookahead*** **= 'q'**
Q derives q
If ***lookahead*** **== 'c' or '$'**
Q derives λ

Predict Calculation

1	$S \rightarrow$	$A\ C\ \$$
2	$C \rightarrow$	c
3	$\mid$	λ
4	$A \rightarrow$	$a\ B\ C\ d$
5	$\mid$	$B\ Q$
6	$B \rightarrow$	$b\ B$
7	$\mid$	λ
8	$Q \rightarrow$	q
9	$\mid$	λ

FIRST (S) = ?　FOLLOW(S) = ?
FIRST (C) = ?　FOLLOW(C) = ?
FIRST (A) = ?　FOLLOW(A) = ?
FIRST (B) = ?　FOLLOW(B) = ?
FIRST (Q) = ?　FOLLOW(Q) = ?

각 생성 규칙의 **Predict Set은?**
Predict(S → A C $) = ?

. . . .
Predict(Q → q) = ?
Predict(Q →λ) = ?

Predict Calculation : ANSWER

1	S	→	A C \$
2	C	→	c
3		|	λ
4	A	→	a B C d
5		|	B Q
6	B	→	b B
7		|	λ
8	Q	→	q
9		|	λ

FIRST (S) = { a, b, c, q, \$ }   FOLLOW(S) = { \$ }

S는 single rule이므로, predict set을 구할 필요가 없음.

FIRST (C) = { c }   FOLLOW(C) = { d, \$ }

FIRST (A) = { a, b, q }   FOLLOW(A) = { c, \$ }

FIRST (B) = { b }   FOLLOW(B) = { c, d, q, \$ }

A → a B C d 에서 FIRST (Cd) = { c, d}

A → B Q 에서 FIRST (Q) = {q}

A → B Q 에서 FOLLOW (A) = { c, \$ }

FIRST (Q) = { q }   FOLLOW(Q) = { c, \$ }

A → B Q 에서 FOLLOW (A) = { c, \$}

LL(1) 조건

결정적 구문 분석을 위해서는
생성 규칙 ***A*** $\rightarrow \alpha$ **|** β 에 대해
아래 조건들을 모두 만족해야 한다.

1. FIRST(α **)** $\cap$ **FIRST(** β **) =** ϕ

2. FOLLOW(A) $\cap$ **FIRST(** β **) =** ϕ**, *if*** $\varepsilon \in$ **FIRST(** α **)**

LL(1) 조건 : 예(1)

$$S \rightarrow aAb$$
$$A \rightarrow aS \mid b$$

FIRST(aS**)** $\cap$ **FIRST(**b**) = {**a**}** $\cap$**{**b**}=**ϕ

$$S \rightarrow ABc$$
$$A \rightarrow bA \mid \varepsilon$$
$$B \rightarrow c$$

FIRST(bA**)** $\cap$ **FOLLOW(**A**) = {**b**}** $\cap$**{**c**}=**ϕ

LL(1) 조건 : 예(2)

$$\begin{aligned} &A \rightarrow iB \leftarrow e \\ &B \rightarrow SB \mid \varepsilon \\ &S \rightarrow [\ eC\ ] \mid .\ i \\ &C \rightarrow eC \mid \varepsilon \end{aligned}$$

FIRST(SB **)** $\cap$ **FOLLOW(** B **) = {**$[\ ,\ .$**}** $\cap$**{** $\leftarrow$ **}=**ϕ

FIRST($[\ eC\ ]$ **)** $\cap$ **FIRST(** $.\ i$ **) = {** $[$ **}** $\cap$**{** $.$ **}=**ϕ

FIRST(eC **)** $\cap$ **FOLLOW(** C **) = {** e **}** $\cap$**{** $]$ **}=**ϕ

***i*[e]** $\leftarrow$ **e**

A => *iB* $\leftarrow$ **e => *iSB*** $\leftarrow$ **e**

=> *i*[e*C*]*B* $\leftarrow$ **e**

=> *i*[e]*B* $\leftarrow$ **e => *i*[e]** $\leftarrow$ **e**

Quiz

- 다음은 **LL(1) grammar**인가**?**

$$E \rightarrow E + id \mid id$$

Table of contents

- 좋은 구문 분석이란?
  - Without backtracking
  - Deterministic parsing
- Predict Set과 LL(1) 조건
- **Recursive descent LL(1) Parsers**
- Table-driven LL(1) Parsers

Recursive Descent Parser (RDP)

To construct a RDP for an LL(1) grammar

- Write a separate procedure for each nonterminal.
- For productions of the form A → λ
  - The parsing procedures simply returns immediately.

- For productions of the form $A \rightarrow X_1 \ldots X_m$
  - X_i is a ***terminal*** symbol, a call to **Match** (*ts*, X_i)
    - *ts* is the token stream
  - X_i is a ***nonterminal*** symbol, a call to the procedure

RDP 구성 예 (1/3) : *match* procedure

```
procedure match (ts, token)
    if ts.PEEK() = token
    then call ts.ADVANCE()
    else call ERROR(Expected token)
end
```

입력에서 읽어 온 **token(*ts*)**이 문법에서 정의한 ***token***과 일치하는가**?**
PEEK : examines the next input token ***without advancing*** **the input**
ADVANCE : advances the input by one token

token

$S \Rightarrow XY \Rightarrow aXY$

abac

ts

P :1. S → XY
2. X → a X
3. X → b
4. Y → aY
5. Y → c

RDP 구성 예 (2/3)

1 S → A C $
2 C → c
3 | λ
4 A → a B C d
5 | B Q
6 B → b B
7 | λ
8 Q → q
9 | λ

FIRST(Q) = {q}
FOLLOW(Q) = {c, $}

```
procedure Q( )
    switch (. . .)
        case  ts.peek() ∈ {q}
            call  match (q)
        case  ts.peek() ∈ {c, $}
            return ()
end
```

RDP 구성 예 (3/3)

```
procedure B( )
    switch (.  )
        case ts.peek() ∈ {b}
            call match (b)
            call B( )
        case ts.peek() ∈ {q, c, d, $}
            return ()
end
```

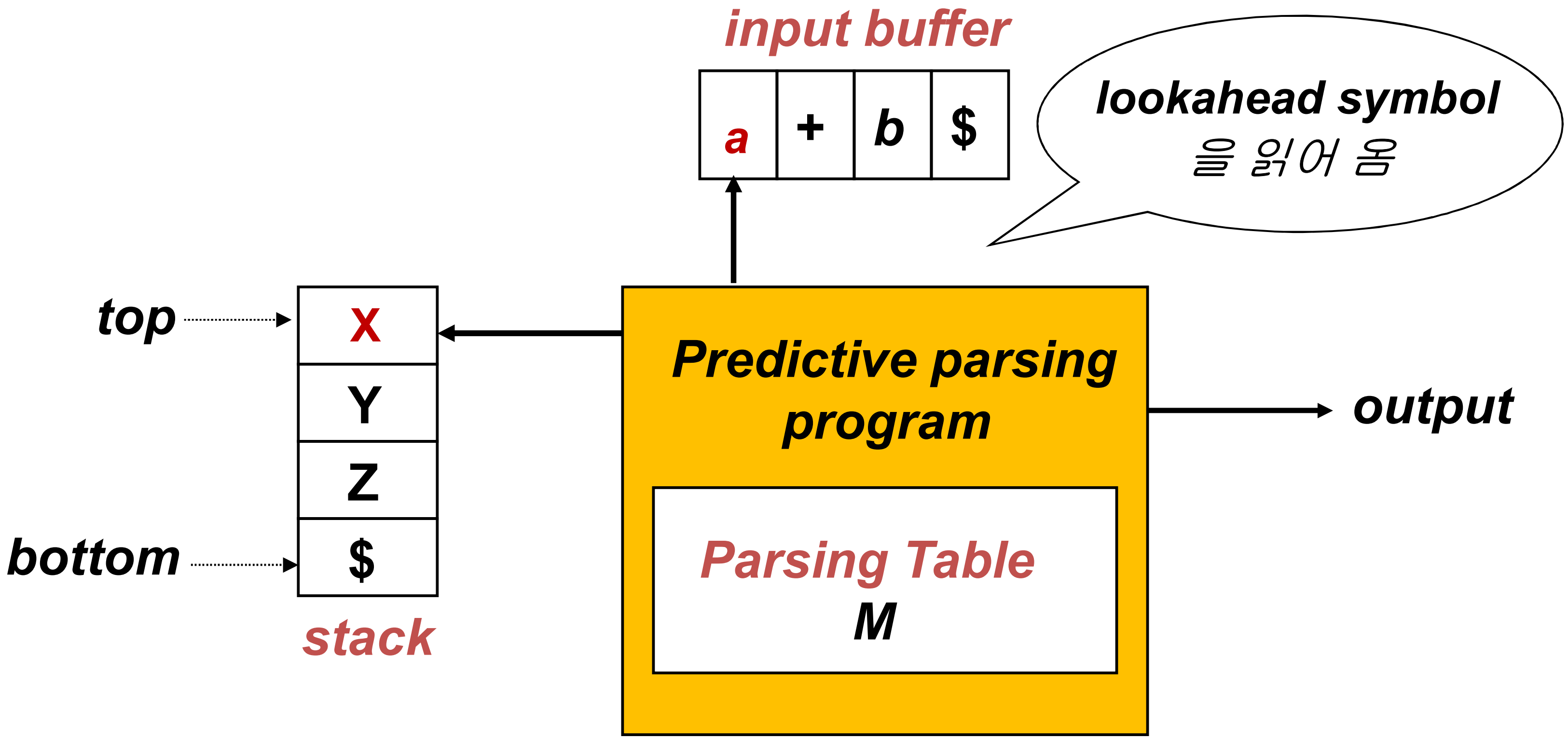
FIRST(B) = {b}
FOLLOW(B) = {c, d, q, $}

1	S	→	A C $
2	C	→	c
3		|	λ
4	A	→	a B C d
5		|	B Q
6	B	→	b B
7		|	λ
8	Q	→	q
9		|	λ

Table of contents

- 좋은 구문 분석이란?
  - Without backtracking
  - Deterministic parsing
- Predict Set과 LL(1) 조건
- Recursive descent LL(1) Parsers
- **Table-driven LL(1) Parsers**
  - **How to build a parsing table**

LL(1) 파싱 : Overview (1/2)



lookahead symbol 과 **stack** 의 **top** 에 있는 심볼을 보고
어떤 **action** 을 취할 것인지를 결정

LL(1) 파싱 : Overview (2/2)

- ***Non-recursive*** **parsing**이라고도 함
- **stack, input buffer** 및 **parsing table**로 구성됨
  - **Parsing Table** : M (A, a)
    - A → Nonterminal, a → lookahead 심벌
    - M(A, a) → *semantic action*이 정의되어 있다.
  - **Stack**
    - 문법 기호들을 저장하고 있다.
      - 지금까지의 파싱 과정에서 만들어진 *sentential form*
  - **Input Buffer**
    - 입력 문자열을 저장하고 있다.
      - 한 번에 한 개의 lookahead 심벌을 읽어 온다.

The Basic Method of LL(1) Parsing

Grammar

$$S \rightarrow (S) S \mid \varepsilon$$

Input

( )

Left-most derivation of a string ( )

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

The Basic Method of LL(1) Parsing

Grammar

$S \rightarrow (S) S \mid \varepsilon$

Input

( )

Parsing actions of a top-down parser

	Parsing Stack	Input	Action
1	$S	( ) $	$S \rightarrow (S) S$

generate

스택이 비었음을 가리키는 기호
(stack is *empty*)

더 이상 읽어 올 입력이 없음
(an *end-marker* of the input file)

The Basic Method of LL(1) Parsing

Grammar

$S \rightarrow (S) S \mid \varepsilon$

Input

( )

Parsing actions of a top-down parser

	Parsing Stack	Input	Action
1	\$S	( ) \$	$S \rightarrow (S) S$
2	\$S) S (	( ) \$	

generate

Left-most derivation of a string ( )

$$S \Rightarrow (S) S$$

The Basic Method of LL(1) Parsing

Grammar

$S \rightarrow (S) S \mid \varepsilon$

Input

( )

Parsing actions of a top-down parser

	Parsing Stack	Input	Action
1	$S	( ) $	$S \rightarrow (S) S$
2	$S) S (	( ) $	match

Left-most derivation of a string ( )

$$S \Rightarrow (S) S$$

The Basic Method of LL(1) Parsing

Grammar

$S \rightarrow (S) S \mid \varepsilon$

Input

( )

Parsing actions of a top-down parser

	Parsing Stack	Input	Action
1	$S	( ) $	S → (S) S
2	$S) S (	( ) $	match
3	$S) S	) $	

The Basic Method of LL(1) Parsing

Grammar

$S \rightarrow (S) S \mid \varepsilon$

Input

( )

Parsing actions of a top-down parser

	Parsing Stack	Input	Action
1	\$S	( ) \$	$S \rightarrow (S) S$
2	\$S) S (	( ) \$	match
3	\$S) S	) \$	$S \rightarrow \varepsilon$

generate

Left-most derivation of a string ( )

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

The Basic Method of LL(1) Parsing

Grammar

$S \rightarrow (S) S \mid \varepsilon$

Input

( )

Parsing actions of a top-down parser

	Parsing Stack	Input	Action
1	$S	( ) $	$S \rightarrow (S) S$
2	$S) S (	( ) $	match
3	$S) S	) $	$S \rightarrow \varepsilon$
4	$S)	) $	

The Basic Method of LL(1) Parsing

Grammar

$S \rightarrow (S) S \mid \varepsilon$

Input

( )

Parsing actions of a top-down parser

	Parsing Stack	Input	Action	
1	$S	( ) $	$S \rightarrow (S) S$	*generate*
2	$S) S (	( ) $	match	
3	$S) S	) $	$S \rightarrow \varepsilon$	*generate*
4	$S)	) $	match	
5	$S	$	$S \rightarrow \varepsilon$	*generate*
6	$	$	accept	

Left-most derivation of a string ( )

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

LL(1) Grammar

- A grammar is an ***LL(1) grammar*** if the associated LL(1) parsing table has ***at most one production*** in each table entry.

Grammar

$$S \rightarrow (S) S \mid \varepsilon$$

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

Construction of an LL(1) parse table

procedure FILLTABLE $(LLtable)$
    foreach $A \in N$ do
        foreach $a \in \Sigma$ do $LLtable[A][a] \leftarrow 0$
    foreach $A \in N$ do
        foreach $p \in ProductionsFor(A)$ do
            foreach $a \in Predict(p)$ do $LLtable[A][a] \leftarrow p$
end

모든 **Nonterminal *A***에 대해
모든 **terminal *a***에 대해

하향식 구문분석 예(1/5)

FIRST(S) = {a, b, c, q, $}
FIRST(C) = { c }
FIRST(A) = { a, b, q}
FIRST(B) = {b}
FIRST(Q) = {q}

FOLLOW(C) = { d, $}
FOLLOW(A) = {c, $}
FOLLOW(B) = {c, d, q, $}
FOLLOW(Q) = {c, $}

1 S → A C $
2 C → c
3 | λ
4 A → a B C d
5 | B Q
6 B → b B
7 | λ
8 Q → q
9 | λ

Predict(S → A C $) = {a, b, c, q, $}

Predict(C → c) = {c}
Predict(C → λ) = {d, $}

Nonterminal	a	b	c	d	q	$
	Lookahead					
S	1	1	1		1	1
C			2	3		3
A	4	5	5		5	5
B		6	7	7	7	7
Q			9		8	9

하향식 구문분석 예(2/5)

TOS: top of stack

TOS가 terminal 이면 입력 기호(***ts***)와 같은지 확인

TOS가 Nonterminal 이면 **Table** 참조

```
procedure LLPARSER(ts)
    call PUSH(S)
    accepted ← false
    while not accepted do
        if TOS( ) ∈ Σ
        then
            call MATCH( ts, TOS( ))
            if TOS( ) = $
            then  accepted ← true
            call POP( )
        else
            p ← LLtable[TOS( ), ts.PEEK ( ) ]
            if p = 0
            then
                call  ERROR(Syntax Error – no production applicable)
            else  call  APPLY(p)
end
```

하향식 구문분석 예(3/5)

procedure APPLY($p : A \rightarrow X_1 \ldots X_m$)
    call *POP( )*
    for $i = m$ downto 1 do
        call PUSH(X_i)
end

생성 규칙의 가장 왼쪽 기호가
Stack의 **top**에 위치해야 하므로

Grammar

$S \rightarrow (S) S \mid \varepsilon$

Input

( )

	Parsing Stack	Input	Action
1	\$S	( ) \$	$S \rightarrow (S) S$
2	\$S) S (	( ) \$	

하향식 구문분석 예(4/5)

Nonterminal	a	b	c	d	q	$
	Lookahead					
S	1	1	1		1	1
C			2	3		3
A	4	5	5		5	5
B		6	7	7	7	7
Q			9		8	9

1 S → A C $
2 C → c
3 | λ
4 A → a B C d
5 | B Q
6 B → b B
7 | λ
8 Q → q
9 | λ

Parse Stack	Action	Remaining Input
S		abbdc$
	Apply 1: S→AC$	
$CA		abbdc$
	Apply 4: A→aBCd	
$CdCBa		abbdc$
	Match	

하향식 구문분석 예(5/5)

Nonterminal	a	b	c	d	q	$
	Lookahead					
S	1	1	1		1	1
C			2	3		3
A	4	5	5		5	5
B		6	7	7	7	7
Q			9		8	9

1	S → A C $	
2	C → c	
3	| λ	
4	A → a B C d	
5	| B Q	
6	B → b B	
7	| λ	
8	Q → q	
9	| λ	

Parse Stack	Action	Remaining Input
S		abbdc$
	Apply 1: S→AC$	
$CA		abbdc$
	Apply 4: A→aBCd	
$CdCBa		abbdc$
	Match	
$CdCB		bbdc$
	Apply 6: B→bB	
$CdCBb		bbdc$
	Match	
$CdCB		bdc$
	Apply 6: B→bB	
$CdCBb		bdc$
	Match	
$CdCB		dc$
	Apply 7: B→λ	
$CdC		dc$
	Apply 3: C→λ	
$Cd		dc$
	Match	
$C		c$
	Apply 2: C→c	
$c		c$
	Match	
$		$
	Accept	

Practice #3

- **Given the grammar**

$$S \rightarrow \boldsymbol{a}\, S \mid \boldsymbol{b}\, A$$

$$A \rightarrow \boldsymbol{a}\, A\, \boldsymbol{b} \mid \varepsilon$$

  - Construct First and Follow sets for each of the Nonterminals
  - Show this grammar is LL(1)