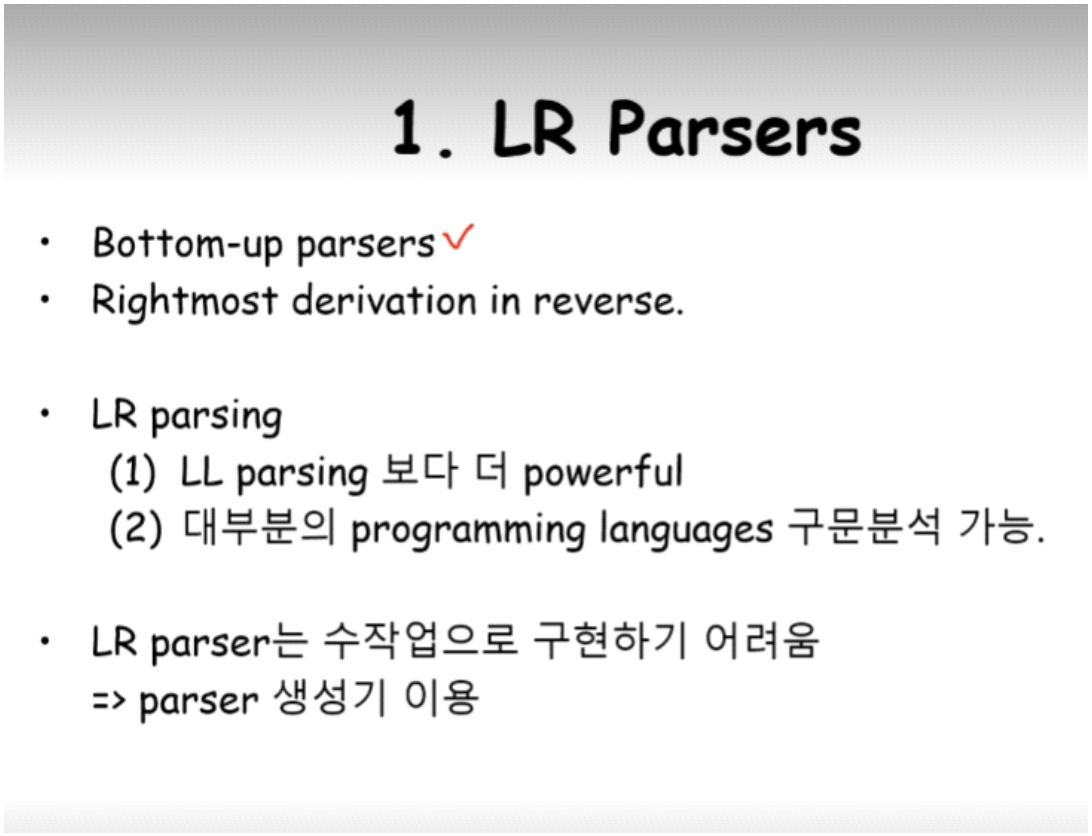
Bottom up Parsing

2022년 5월 25일 수요일 오후 12:16



화면 캡처: 2022-05-25 오후 12:16

Rightmost dervation의 역순으로 진행한다
LL parsing 보다 powerful 하다
대부분의 programming languages 구문 분석 가능

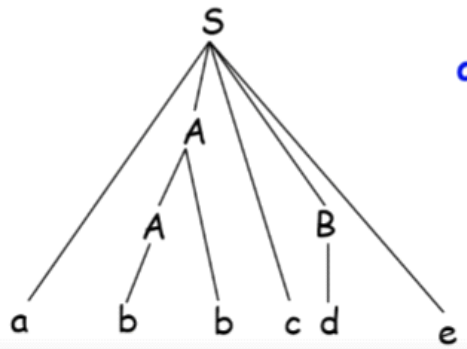
Bottom-up Parsing

ex) G: (1) S → aAcBe
(2) A → Ab
(3) A → b
(4) B → d

input : abbcde

rm derivation

S=>aAcBe
1
=> aAcde
4
=> aAbcde
2
=> abbcde
3



parsing

abbcde => aAbcde
3
=> aAcde
2
=> aAcBe
4
=> S
1

화면 캡처: 2022-05-25 오후 12:18

Handle을 non terminal로 치환 -> handle pruning

Bottom-up Parsing

Input string을 start symbol로 reduce 한다.

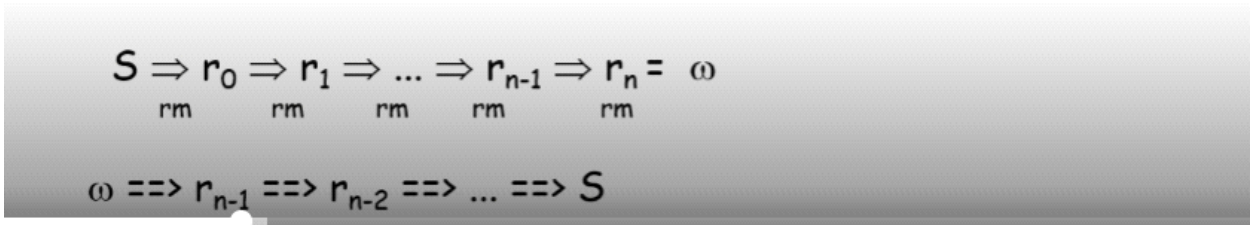
정의
reduce: A->β인 경우 β => A의 역변환을 reduce 라고 한다.

handle: $S \Rightarrow \alpha A \omega \Rightarrow \alpha\beta\omega$ 라면, β를 αβω의 **handle** 이라고 한다.

handle pruning
$\alpha A \omega \Rightarrow \alpha\beta\omega$ 의 역변환, 즉, handle β를 A로 reduce하는 것을 handle pruning 이라고 한다.

LR parsing은 handle pruning 을 반복하여 start symbol로

LR parsing은 handle pruning 을 반복하여 start symbol로 reduce 한다.



화면 캡처: 2022-05-25 오후 12:23

Reduce

A-> β 인 경우 β => A의 역변환을 reduce라고 한다

Handle

abbcde => aAbcde 역변환 할때 handle b를 non terminal A로 치환함

Handle을 non terminal로 치환하는것을 **Handle pruning이라고 한다**

LR parsing은 handle pruning을 계속 하는 과정이다

ex) G: E → E + T | T　　string : a + a * a
　　T → T * F | F
　　F → (E) | a

E => E + T
=> E + T * F => E + T * a => E + F * a => E + a * a => T + a * a
=> F + a * a => a + a * a

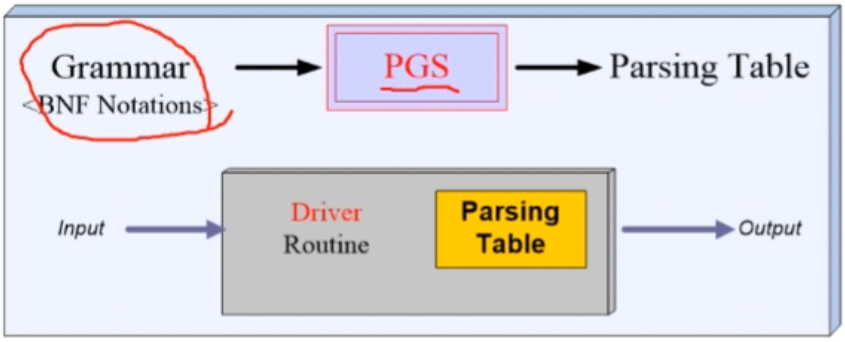
E => <u>E + T</u>
⇒ E + <u>T * F</u>
⇒ E + T * <u>a</u>
⇒ E + <u>F</u> * a
⇒ E + <u>a</u> * a
⇒ <u>T</u> + a * a
⇒ <u>F + a</u> * a
⇒ <u>a</u> + a * a

화면 캡처: 2022-05-25 오후 12:25

E를 aaa로 바꿀때 Rightmost derivation 사용하고 있음을 알 수 있다

a가 handle이 되어서 a가 F로 바뀌고 F+A가 T로 바뀌면서 계속 handle pruning을 수행한다
결국 E가 된다

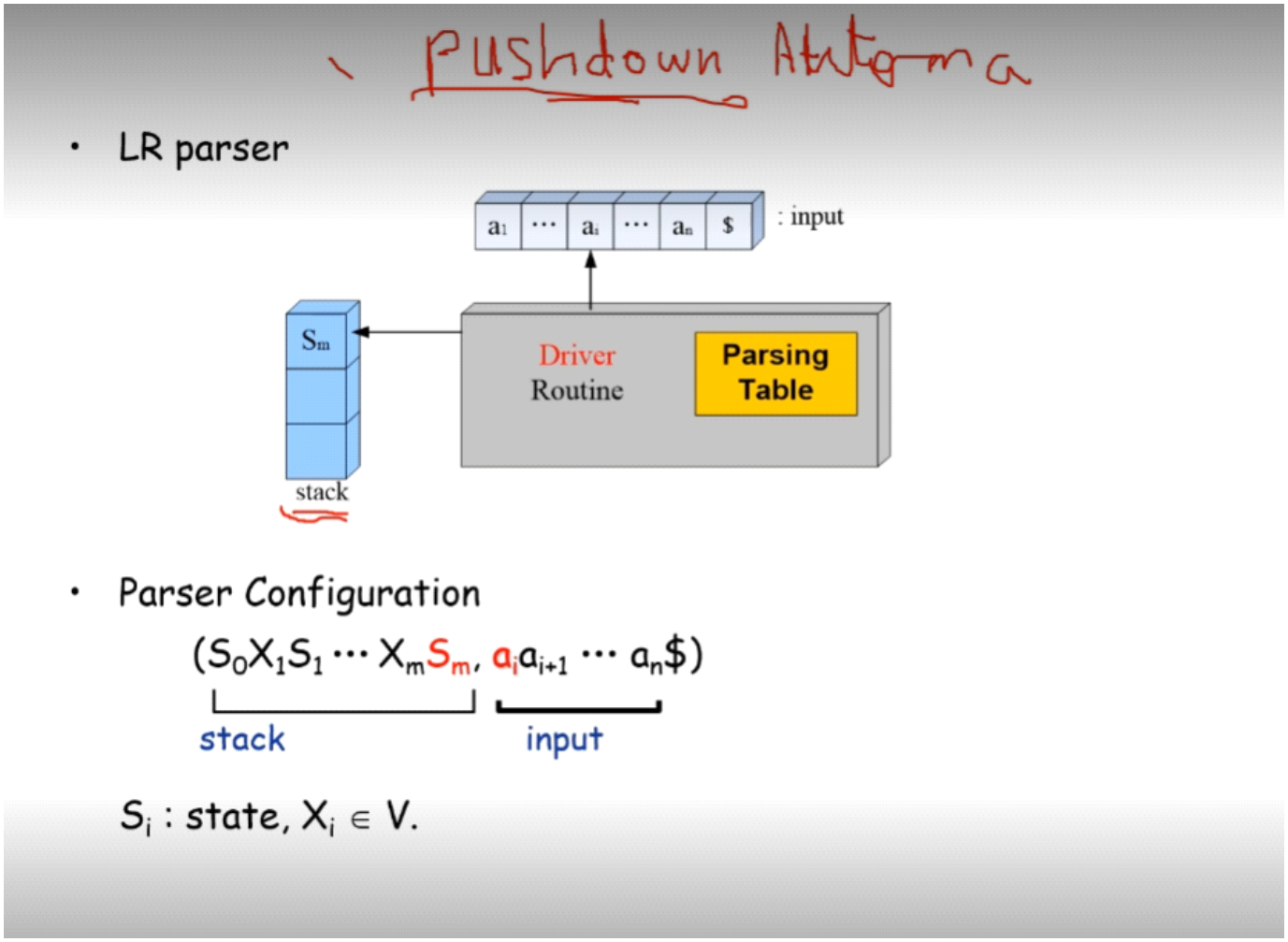
- Parser Generating System 모형



- Driver routine 은 생성되는 모든 parser에서 동일
- Parsing table만 Grammar에 따라 달라짐

화면 캡처: 2022-05-25 오후 12:28

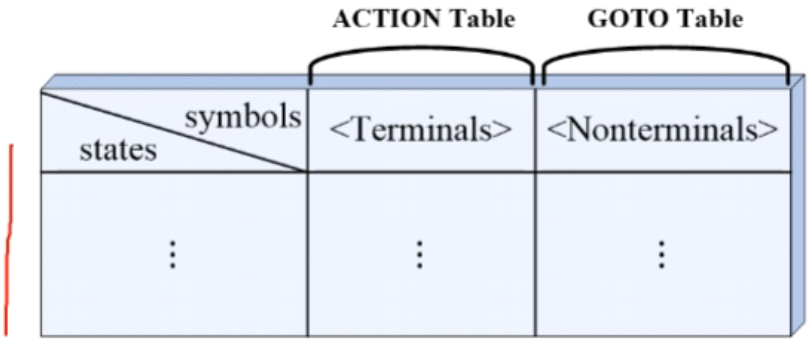
PushDown(stack) automata



화면 캡처: 2022-06-06 오후 6:50

Parsing Table 구조

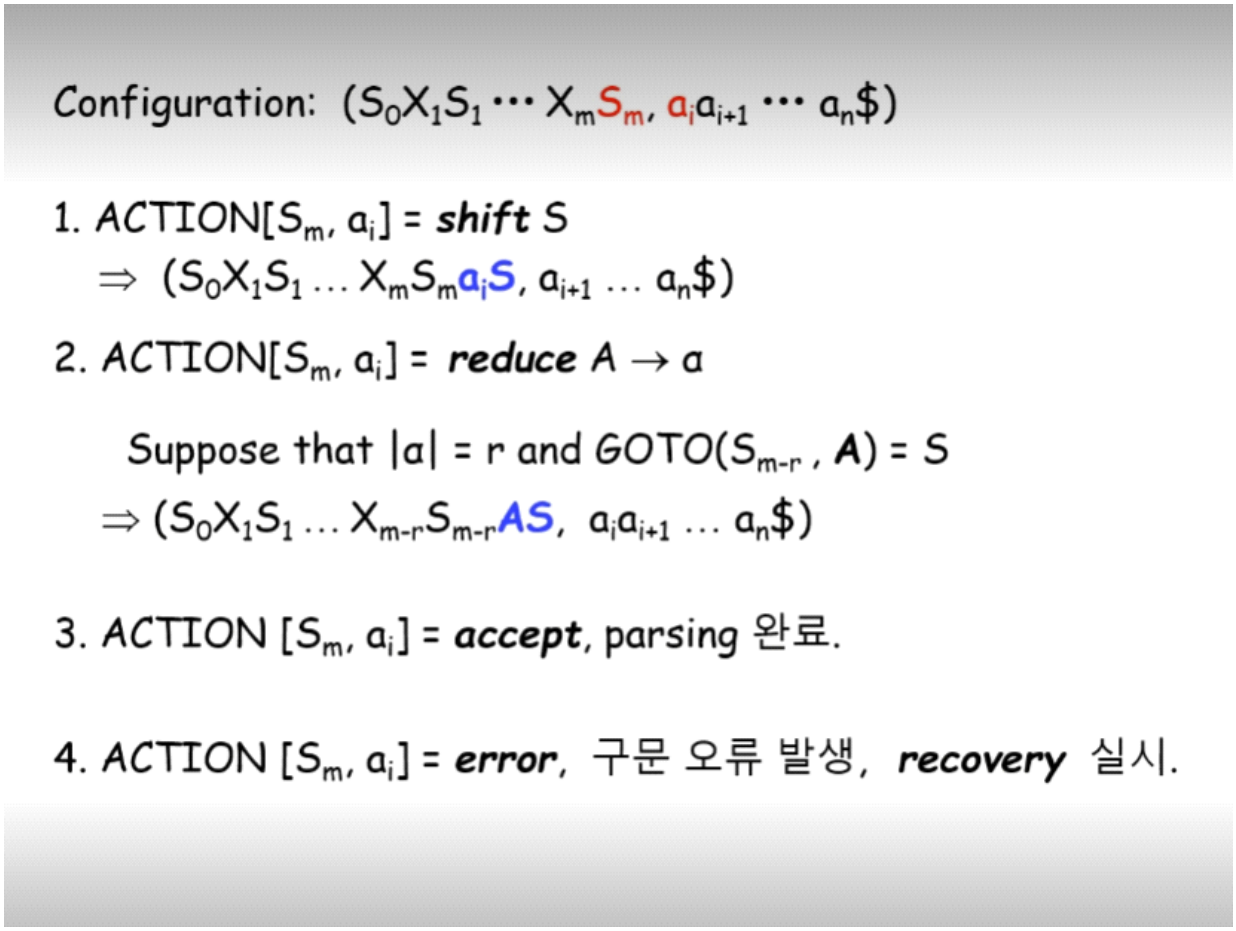
- Parsing Table 구조 (ACTION table + GOTO table)



- The LR parser actions :

1. shift
2. reduce
3. accept
4. error

화면 캡처: 2022-06-06 오후 6:54



화면 캡처: 2022-06-06 오후 6:56

예를 보면서 파싱하면 위의 설명 이해 가능하다

Example

ex) G: 1. LIST → LIST , ELEMENT
2. LIST → ELEMENT
3. ELEMENT → a

Parsing Table :

states \ symbols	a	,	$	LIST	ELEMENT
0	s3			1	2
1		s4	acc		
2		r2	r2		
3		r3	r3		
4	s3				5
5		r1	r1		

s3 means **shift input** and push state 3,
r2 means **reduce** by production 2,
acc means **accept**,
blank means **error**.

화면 캡처: 2022-06-06 오후 6:57

a
a,a,
a,a,a
생성할수있는 언어이다

Termainl

a , $ ==> Action Table

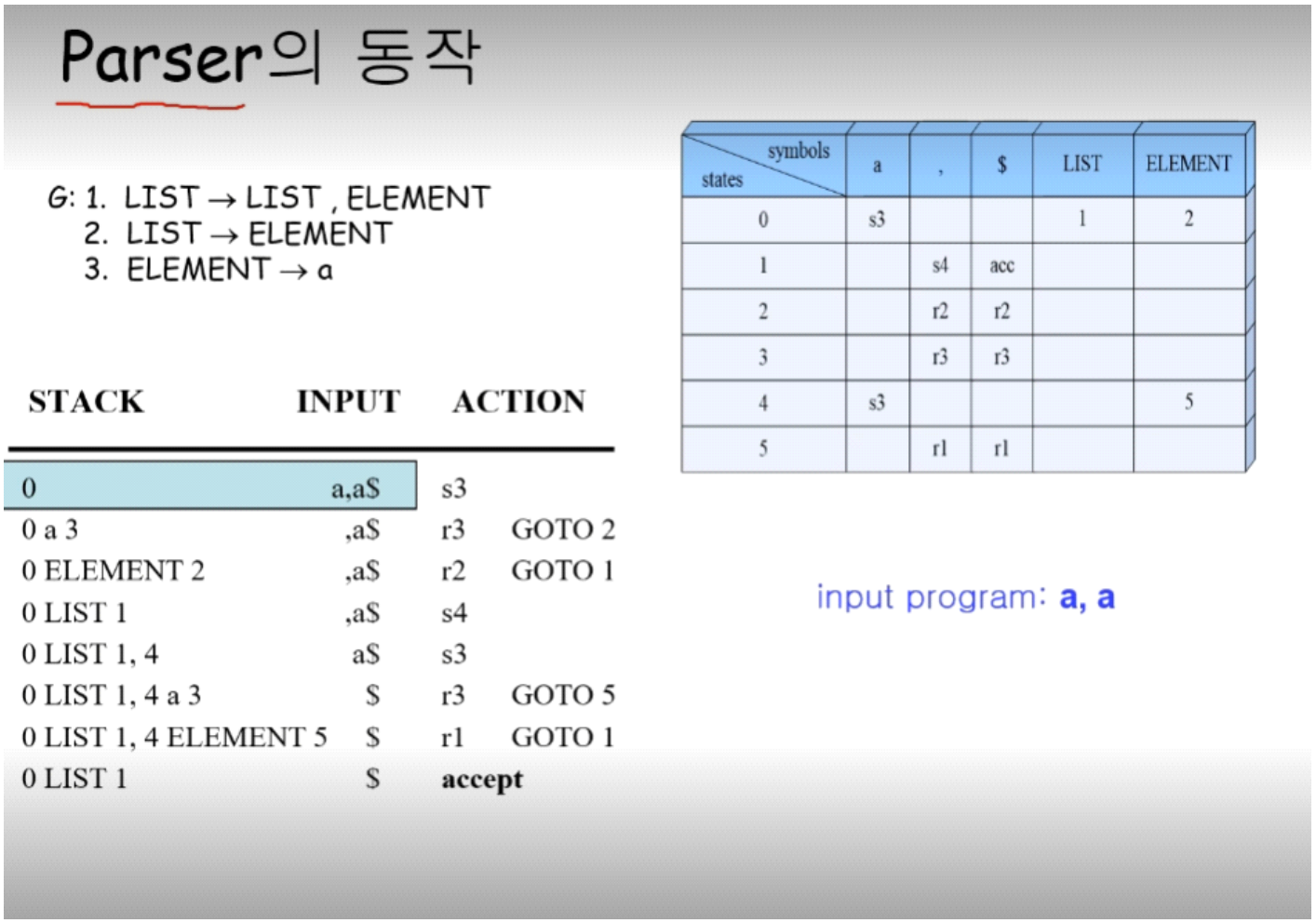
Non terminal

List, Element => GOTO table

S3의 의미 shift input and push state 3
그러면 S2는 shift input and push state 2

R2의 의미 reduce by production 2 => 규칙 2번에 의해 reduce 되었다는 의미

acc의 의미는 accept
Blank(빈칸)의 의미는 error => error Recovery해야한다



화면 캡처: 2022-06-06 오후 7:07

Input program : a,a

0이 시작 상태

Step 1
Stack top의 상태와 입력 기호를 가지고 action table을 찾아본다

0과 a를 가지로 action table 찾아본다 -> S3 [input을 shift 하고 3을 push 한다]

Input을 shift한다는 것은 지금 입력 a를 shift하면 stack에 a가 들어가게 된다 그리고 3을 push하게 되면

Stack의 상태가 0 a 3이 된다 input은 ,a$가 된다

Step2

3과 , 가지고 action table을 찾으면 r3가 나온다 3번 규칙으로 reduce하라

3번 규칙은 Element -> a니까 reduce하면 a가 Element가 된다

Reduce된 대상과 상태를 stack에서 pop한다 stack에 a와 3이 pop이 되고 a는 Element로 바뀐다

그러면 0 Element가 stack상태가 되는데 non terminal 나왔으니까 GOTO table을 살펴 본다
0과 Element를 참고 해서 GOTO table을 찾으면 2가 나온다 이걸 stack에 push하면
최종 stack 상태는 0 Element 2가 된다

Setp 3

이제 2와 , 를 가지고 action table검색하면 r2가 나옴

2번 규칙 List -> Element 가지고 reduce한다 이때 상태도 같이 pop하고 바뀐다

Stack에 Element는 List로 바뀐다

stack상태는 0 List가 된다 0과 list가지고 GOTO table검색하면 1이 나온다 나온 결과 1을 다시 stack에 push한다

최종 stack 상태는 0 List 1

Step 4

1과 ,을 action table에서 찾으면 S4가 나온다 이러면 input ,을 shift한다 그리고 상태 4를 저장한다

Step 5

4와 a를 action table에서 찾는다 S3가 나옴 a를 shift 하고 3을 추가한다

최종 stack은 0 LIST 1,4 a 3이 된다

Step 6

3과 $를 action table에서 찾으면 r3가 된다 r3는 3번 규칙으로 reduce하니까 a를 ELEMENT로 reduce한다
Stack이 0 LIST 1,4 ,ELEMENT 가 된다

4와 element 통해서 GOTO가면 5가 되니까 5를 추가 한다

최종 stack은 0 LIST 1,4 ,ELEMENT 5

Step 7

5와 $를 보면 r1이 나온다

List -< list,element로 reduce한다

List , Element와 상태 1 4 5를 pop하고 list로 바꾼다

List의 상태 ,의 상태 element의 상태 번호도 같이 pop하는 작업이다

최종 stack은 0 list 이된다.

Step 8
0과 list 검색해서 들어가면 1이 추가 된다

Stack : 0 list 1

Step 9

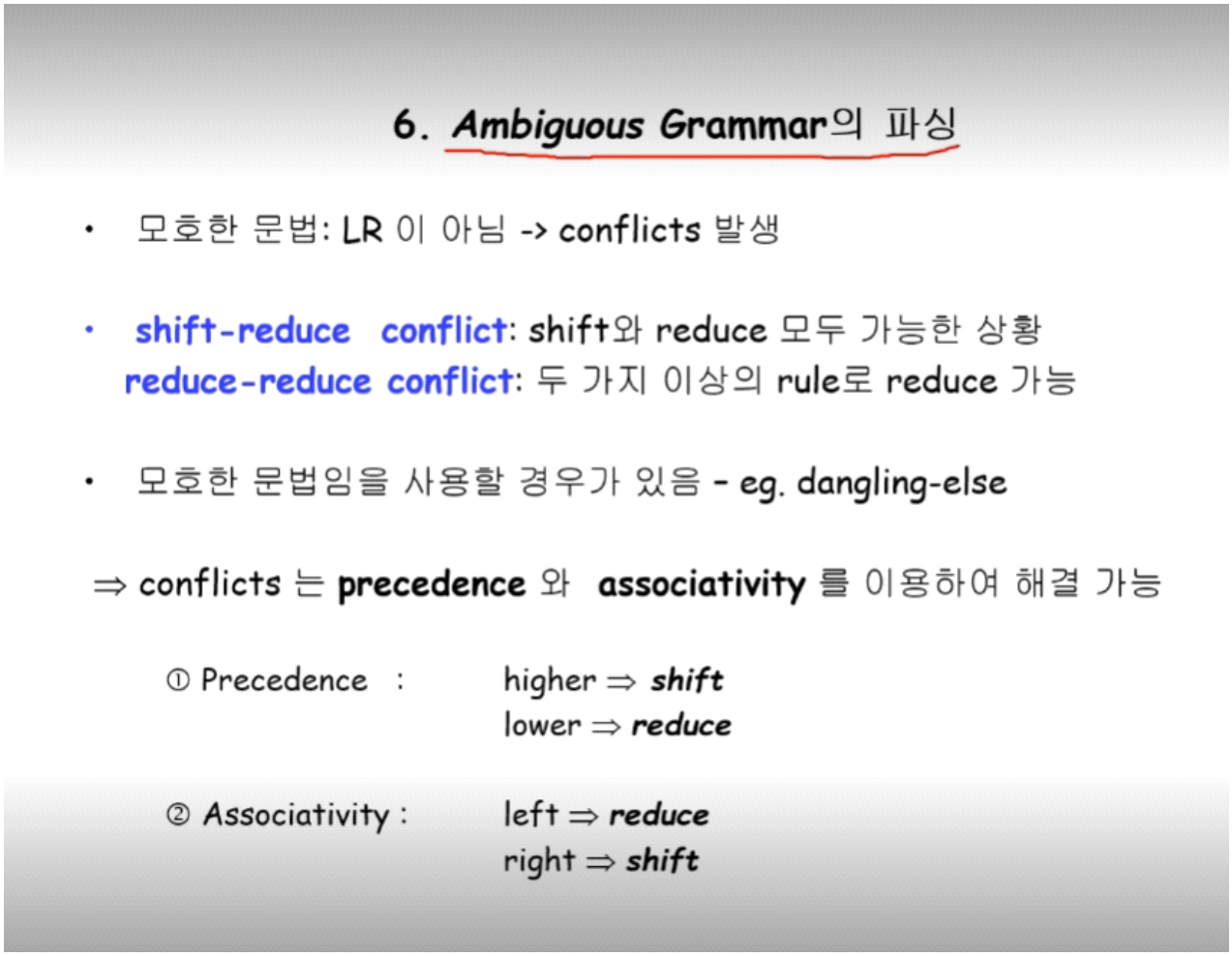
1과 $가 되면 accept된다

5. LR Parser 종류

- Simple LR(SLR), Canonical LR(CLR), Look-Ahead LR(LALR)
- Parsing table의 크기는 상태 수에 비례
  - parsing table의 크기 : |states| × |V|
  - 일반적인 프로그래밍 언어 문법 :
    - |V| = 100, |states| = 300, parsing table 크기 = 30,000 entries
- parsing 능력
  CLR > LALR > SLR
- parsing table의 크기:
- CLR >> LALR ≈ SLR

종류 3가지
Simple LR
Canonical LR
Look ahead LR

Parser generator -> yacc [LALR]



화면 캡처: 2022-06-06 오후 8:37

LALR은 모호한 문법 파싱 못한다

모한 문법에다가 LR적용하면 conflicts가 발생하기 때문이다.

모호한 문법임에도 사용하는 경우가 있다 -> dangling else

Conflict는 precedence와 associativity를 이용하여 해결 가능

Conflict 종류

- **Shift reduce conflict**
  - Shift와 reduce 모두 가능한 상황
- **Reduce- reduce conflict**
  - 두가지 이상의 rule로 reduce 가능