# 12. LR파서

충북대학교

이 재 성

# 학습내용

- 상향식 파서의 기본 개념
- LR파서의 종류 및 작동 예

# 상향식 파싱(Bottom-up Parsing)

## 상향식 파싱

- 이동 축소 파싱(shift-reduce parsing)과 동일
- 축소과정을 반복하여 파싱
  - 축소: 생성규칙의 오른쪽과 일치하는 부분 문자열을 왼쪽의 기호로 대체하는 작업
- 맨 오른쪽 유도(right most derivation)의 역순
- 예:

  $S \rightarrow aABe$

  $A \rightarrow Abc \mid b$

  $B \rightarrow d$

- 파싱예

  a**b**bcde => a**Abc**de => aA**d**e => **aABe** => S

  (역으로하면 맨오른쪽 유도와 동일)

# 핸들

## 핸들

- 생성규칙의 오른쪽과 일치하는 부분문자열
- S =>aAw => abw
- b는 A → b의 규칙에 해당되는 핸들
- 이때 w는 모두 단말기호들만으로 구성

## 핸들의 결정

- a**b**bcde => a**Abc**de => aA**d**e => **aABe** => S
- a**b**bcde => aA**b**cde => aAAcde => **?**

참고 : 문법
S → aABe
A → Abc | b
B→ d

## 핸들가지치기(handle pruning)

- 핸들을 발견하여 왼쪽의 문법기호로 계속 대체함으로써 파스 트리를 축소 (시작기호까지 축소)

# 스택을 이용한 이동 축소 파싱

## 동작

- 스택이 빈 상태에서 입력문자열을 하나씩 스택 이동
- 스택의 맨 위에 핸들 $\beta$가 오면 이 핸들을 생성규칙에 의해 축소
- 다시 입력 문자열을 받아서 축소 동작을 반복
- 입력문자열이 모두 처리되고, 스택에 시작기호만 있으면 분석 끝

## 동작의 종류

- 이동(shift): 다음의 입력기호를 스택 위로 옮김
- 축소(reduce): 핸들의 오른쪽 끝이 스택의 맨 위에 오면 이를 축소
- 수용(accept): 파싱의 성공적인 완료
- 오류(error): 구문 오류의 발생

# 스택을 이용한 이동축소 파싱의 동작예

## 동작 예

| 스택 | 입력 | 동작 |
|---|---|---|
| (1) $ | id + id * id$ | shift |
| (2) $id | + id * id$ | reduce by E → id |
| (3) $E | + id * id$ | shift |
| (4) $E + | id * id$ | shift |
| (5) $E + id | * id$ | reduce by E → id |
| (6) $E + E | * id$ | shift |
| (7) $E + E * | * id$ | shift |
| (8) $E + E * id | $ | reduce by E → id |
| (9) $E + E * E | $ | reduce by E → E*E |
| (10) $E + E | $ | reduce by E → E+E |
| (11) $E | $ | accept |

문법

1. E → E + E
2. E → E * E
3. E → (E)
4. E → id

- 바이어블 프리픽스(Viable Prefixes)
  - 스택에 나타낼 수 있는 문자열집합
  - 오른쪽-문장형태(right sentential form)의 앞 문자열(prefix)

# 이동/축소 충돌

## 발생 상황

- 다음입력 기호에 대해 이동인지 축소인지 결정 불가

## 이동/축소 충돌 상황의 예

문법

stmt → if expr then stmt
| if expr then stmt else stmt
| other

| STACK | INPUT |
|---|---|
| ... if expr then stmt | else ... $ |

이동: stmt → if expr then stmt else stmt로 이동한 후 축소
축소: stmt → if expr then stmt로 축소

# 축소/축소 충돌

## 발생 상황

- 어떤 생성규칙을 적용하여 축소할 것인지 결정 불가

## 축소/축소 충돌 상황의 예

| STACK | INPUT |
|---|---|
| ... id(id | ,id) ... $ |

문법

(1) stmt → id(para_list)
(2) stmt → expr := expr
(3) para_list → para_list, parameter
(4) para_list → parameter
(5) parameter → id
(6) expr → id(expr_list)
(7) expr → id
(8) expr_list → expr_list, expr
(9) expr_list → expr

A(i, j)의 문장에 대한 파싱 id(id, id)

- (5)번 규칙 적용시: A를 프로시저로 취급
- (7)번 규칙 적용시: A를 배열로 취급
- 해결책 예: A를 선언한 것에 따라 다르게 처리.
- 즉 프로시저로 선언되었다면, 어휘분석기에서 A를 procid로 전달 (문법수정필요)

# LR 파서 (LR parsers)

## LR(k) 파싱 용어

- " L" : left to right scan
- "R" : right most derivation
- k : 파싱을 결정하는데 사용되는 예측기호의 개수
  - k가 생략되면 k=1로 가정

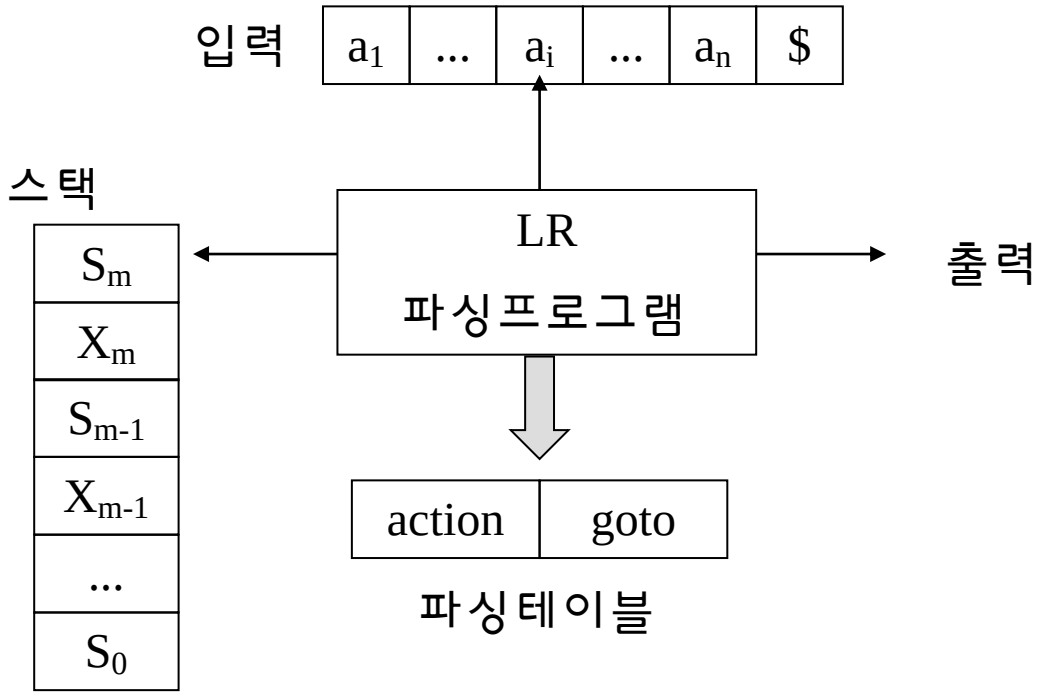
## 장단점

- 장점
  - CFG로 쓰여질 수 있는 모든 언어는 LR파싱으로 인식 가능
  - 백트래킹없이 이동축소 파싱이 가능한 가장 일반적인 방법
  - 예측파서로 파싱할 수 있는 문법은 모두 LR파서로 파싱 가능
  - left to right scan으로 구문오류가 발생하면 곧바로 발견
- 단점
  - 문법작성시 많은 작업이 필요: 파서생성기(예: Yacc등) 필요

## 종류

- SLR(simple LR)
  - 구현이 가장 쉬우나 인식능력이 가장 빈약
- CLR(canonical LR)
  - 구현이 복잡하나 인식능력이 가장 뛰어남
- LALR(lookahead LR)
  - 구현 및 인식능력이 중간 정도

# LR 파싱 알고리즘




Xi : 문법 기호
Si: 상태 기호

- 상태기호는 그 기호 아래쪽의 스택 정보를 종합
- action은 스택 위의 상태와 입력기호를 첨자로 파싱테이블에서 이동과 축소를 결정 action(Sm, ai)
- goto함수: 상태와 문법기호를 인수로 하여 새로운 상태 반환

# LR파서의 이동

## 나열형태

- 스택의 내용과 처리되지 않은 입력을 표시

  예: $(s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_m\ S_m,\ a_i\ a_{i+1}\ \ldots\ a_n\$)$

- 우문장형태 $(X_1X_2\ \ldots\ X_ma_ia_{i+1}\ \ldots\ a_n\ )$를 나열 형태로 표현

## 파서의 이동동작

- 이동축소 파서의 동작과 동일하되 상태만 추가
- 이동: 현재 입력 기호 $a_i$와 스택 맨 위의 상태 $S_m$을 읽고 결정

# action의 네가지 유형

- 나열 형태
  - $(s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_m\ S_m,\ a_i\ a_{i+1}\ \ldots\ a_n\$)$
- action[$s_m$, $a_i$] = shift s
  - 나열형태 $(s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_m\ S_m\ a_i\ s,\ a_{i+1}\ \ldots\ a_n\$)$
  - 현재 입력 기호 $a_i$와 다음상태 s(goto[$s_m$, $a_i$]의 값)를 푸시
- action[$s_m$, $a_i$] = reduce A->b
  - 나열형태 $(s_0\ X_1\ s_1\ X_2\ s_2\ \ldots\ X_{m-r}\ S_{m-r}\ A\ s,\ a_i\ a_{i+1}\ \ldots\ a_n\$)$
  - 스택에서 2r개의 기호 팝 (r = |b| 이고, 팝되는 문법기호는 b와 일치)
  - 생성규칙의 왼쪽 기호 A와 다음상태 s(goto[$s_{m-r}$, A]의 값을 푸시
  - 입력기호는 변화없음
- action[$s_m$, $a_i$] = accept
  - 파싱 종료
- action[$s_m$, $a_i$] = error
  - 오류 발견, 오류 회복 루틴 호출

# LR 파싱 알고리즘

- 입력
  - 입력문자열 w
  - 문법 G에 대한 LR 파싱테이블
- 출력
  - w가 L(G)안에 있다면 상향 파스,
  - 아니면 오류

```
스택 맨 위에 s0푸시, 입력버퍼에 w$넣음
ip를 w$의 처음 기호 위치로 설정
repeat forever begin
        s=스택 맨위에 있는 상태변수
        a=ip가 포인트하는 입력기호
        if action[s, a] = shift s' then begin
                스택에 a를 푸시하고 다음상태를 s'로
                ip를 다음 입력기호로  end
        else if action[s, a] = reduce A -> b then begin
                스택에 2*|b|개수의 기호를 제거
                이때 스택의 맨위의 상태를 s'라함
                스택에 A를 푸시, 다음 상태를 goto[s', A]로
                생성규칙 A-> b를 출력     end
        else if action[s, a] = accept then
                return
        else error()
end
```

# 파싱 테이블 예

| | action | | | | | | goto | | |
|---|---|---|---|---|---|---|---|---|---|
| 상태 | id | + | * | ( | ) | $ | E | T | F |
| 0 | s5 | | | s4 | | | 1 | 2 | 3 |
| 1 | | s6 | | | | acc | | | |
| 2 | | r2 | s7 | | r2 | r2 | | | |
| 3 | | r4 | r4 | | r4 | r4 | | | |
| 4 | s5 | | | s4 | | | 8 | 2 | 3 |
| 5 | | r6 | r6 | | r6 | r6 | | | |
| 6 | s5 | | | s4 | | | | 9 | 3 |
| 7 | s5 | | | s4 | | | | | 10 |
| 8 | | s6 | | | s11 | | | | |
| 9 | | r1 | s7 | | r1 | r1 | | | |
| 10 | | r3 | r3 | | r3 | r3 | | | |
| 11 | | r5 | r5 | | r5 | r5 | | | |

## 문법

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow id$

# LR파서의 동작 예

| 스택 | 입력 | 동작 |
|---|---|---|
| (1) 0 | id * id + id$ | shift |
| (2) 0 id 5 | * id + id$ | reduce by F → id |
| (3) 0 F 3 | * id + id$ | reduce by T → F |
| (4) 0 T 2 | * id + id$ | shift |
| (5) 0 T 2 * 7 | id + id$ | shift |
| (6) 0 T 2 * 7 id 5 | + id$ | reduce by F → id |
| (7) 0 T 2 * 7 F 10 | + id$ | reduce by T → T*F |
| (8) 0 T 2 | + id$ | reduce by E → T |
| (9) 0 E 1 | + id$ | shift |
| (10) 0 E 1 + 6 | id$ | shift |
| (11) 0 E 1 + 6 id 5 | $ | reduce by F → id |
| (12) 0 E 1 + 6 F 3 | $ | reduce by T → F |
| (13) 0 E 1 + 6 T 9 | $ | E→ E+T |
| (14) 0 E 1 | $ | accept |

문법

E → E + T
E → T
T → T * F
T → F
F → (E)
F → id

# 참고 문헌


- [1] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, "Compilers – Principles, Techniques, and Tools," Bell Telephone Laboratories, Incorporated, 1986.
- [2] 오세만, "컴파일러 입문", 정익사, 2004.