Compiler (컴파일러)
Syntax Analysis-Top Down

2015년 2학기

충남대학교 컴퓨터공학과
조은선

Top-down 으로 구문 분석하기

- 직관적, 좌측 유도 과정과 유사
- 시작:
  - 시작 심벌의 첫번째 생성 규칙으로 유도하여 문자열 생성
- 진행
  - 유도 과정에서 생성된 문자열과 입력 문자열을 차례로 비교
    - 유도된 문자열과 입력문자열이 같으면 계속 비교해 나감
    - 유도 과정에서 생성된 문자열에 nonterminal이 나오면 이 nonterminal의 첫번째 생성규칙으로 또다시 유도하여 새 문자열 생성 후 비교해 나감
  - 다르면? … backtracking!

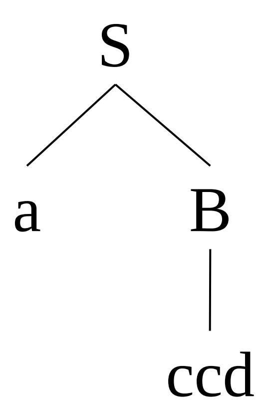
Backtracking

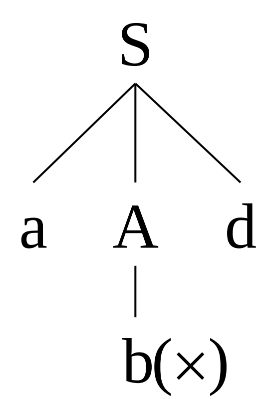
- 비교한 두 문자가 같지 않을 경우,
  - 직전 생성 규칙을 잘못 적용한 것이므로 원위치 시키고 다른 생성규칙을 적용 (→ 이게 backtracking)
    - 직전 규칙은 적용 안 했던 것으로 하고
    - 생성된 문자열도 직전 규칙 적용 전으로 환원시키고
    - 입력 문자열도, 직전 규칙 적용 후에 비교됐던 문자 개수만큼 원위치 시킴 (안 읽었던 척)
  - 다른 생성 규칙을 가지고 유도해도 마찬가지로 같지 않은 문자가 나타나면 또 반복
  - 이상과 같은 과정을 반복하다가 더 이상 적용할 생성 규칙이 없으면, 입력 문자열은 틀린 문장으로 인식

예제

스트링 accd가 정의된 문법의 올바른 문장임을 확인

1. $S \rightarrow aAd$ 2. $S \rightarrow aB$
3. $A \rightarrow b$ 4. $A \rightarrow c$
5. $B \rightarrow ccd$ 6. $B \rightarrow ddc$





LL 파싱

- 왼쪽에서 오른쪽으로 읽어가며(Left to right scanning)
- 좌파스(Left Parse)를 생성
- 결정적으로 (deterministic) 파싱
  - 입력 문자를 보고 적용될 생성 규칙을 결정적으로 선택하여 유도
  - 준비가 필요
    - 각 입력 문자당 적용될 생성 규칙을 미리 뽑아 두고 시작
    - 한 입력 문자에 두 개 이상의 규칙이 해당되면? 파싱 불가능! (단점)
- Backtracking 이 없다! (장점)
  - 현재의 입력 문자와 생성된 terminal 심벌이 같지 않으면 그냥 틀린 것으로 간주
  - Top down 방식의 시간 낭비를 줄임

First

$$\mathrm{FIRST}(A) = \{a \in V_T \cup \{\varepsilon\} \mid A \Rightarrow^* a\gamma,\ \gamma \in V^*\}$$

- FIRST 는 nonterminal A로 부터 유도되어 첫번째로 나타날 수 있는 terminal의 집합
- 앞의 예에서
  - FIRST(S) 는?
    - S→aAd 와 S→aB 므로 {a}
  - FIRST(A)는?
    - A→b 와 A→c 므로 {b,c}
  - FIRST(B)는?
    - B → ccd와 B→ddc 에서 {c,d}
- S → Abe 와 같은 규칙이 추가되면 FIRST(S)는 ?
  - {b,c} 가 추가됨
  - 일반적으로는 "계산"이 필요

FIRST(X)를 계산하는 방법

- 기본) X가 terminal이면 X의 FIRST는 자기 자신이 됨
  - $FIRST(a) = \{ a \mid \text{if } a \in V_T \}$
- $X \rightarrow a\alpha$의 생성규칙이 존재하면 a가 FIRST(X)에 포함됨
  - $FIRST(X) = FIRST(X) \cup \{a\}$ if $X \rightarrow a\alpha \in P$ and $a \in V_T$
- $X \rightarrow \varepsilon$ 즉, X가 ε-생성규칙을 가지면 X의 FIRST에 ε이 포함됨
  - $FIRST(X) = FIRST(X) \cup \{\varepsilon\}$ if $X \rightarrow \varepsilon \in P$
- $X \rightarrow Y_1 Y_2 \ldots Y_k$인 생성규칙이 있을 때,
  - $FIRST(X) = FIRST(X) \cup FIRST(Y_1 Y_2 \ldots Y_k)$이다.

    단, $FIRST(A_1 A_2 \ldots A_n) = FIRST(A_1) \oplus FIRST(A_2) \ldots \oplus FIRST(A_n)$
  - 모든 nonterminal의 FIRST가 변하지 않을 때 까지 반복

사용된 정의 1 : ε-생성규칙

- ε-생성규칙
  - X → ε 형태의 생성규칙을 의미
- Nullable nonterminal
  - Nonterminal A가 ε가 유도할 수 있으면 A를 nullable하다고 부른다.
  - 즉, $A \Rightarrow^* \varepsilon$가 가능하면 A가 nullable하다.

사용된 정의 2 : lhs, rhs

- 생성규칙 A→XXX 에서
  - lhs (left hand side): A
  - rhs (right hand side) : XXX

사용된 정의 3 : ⊕ (Ring Sum)

- 두개의 집합에 대한 연산자인 ring sum ⊕ 은 다음과 같이 정의 됨.
  - if $\varepsilon \notin A$ then $A \oplus B = A$
  - if $\varepsilon \in A$ then $A \oplus B = (A - \{\varepsilon\}) \cup B$
- 예 $\{a, b, c\} \oplus \{c, d\} = \{a, b, c\}$
  $\{a, b, c, \varepsilon\} \oplus \{c, d\} = \{a, b, c, d\}$
  $\{a, b, \varepsilon\} \oplus \{c, d, \varepsilon\} = \{a, b, c, d, \varepsilon\}$

- $FIRST(A_1 A_2 \ldots A_n) = FIRST(A_1) \oplus FIRST(A_2) \ldots \oplus FIRST(A_n)$ 의 용도
  - S→Ab S→c A→ ε 라면..
    'b'가 들어오면 S→Ab, 'c'가 오면 S→c 로 유도하는게 맞다.
    - FIRST(S) = {c}, FIRST(A) = {ε} 를 구한 후
    - FIRST(S) = {b, c} 로 보정.
      - FIRST(Ab)= FIRST(A) ⊕ FIRST(b) 에 의함

- 구하는 방법
  - 초기에 모든 nonterminal은 FIRST는 공집합이 된다.
  - rhs에 첫번째로 terminal이 나오는 생성 규칙(ε-생성 규칙 포함)에서 FIRST를 구한 후, 이 형태의 생성 규칙은 다시 고려할 필요가 없기 때문에 제거한다.
  - 남은 생성 규칙의 형태는 모두 nonterminal로 시작하므로 ring sum 연산을 이용하여 모든 nonterminal의 FIRST가 변하지 않을 때까지 반복한다.
  - 일반적으로 A-생성 규칙이 $A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n \in P$와 같은 형태일 때 다음과 같이 된다.
    - $FIRST(A) = FIRST(\alpha_1) \cup FIRST(\alpha_2) \cup \ldots \cup FIRST(\alpha_n)$

예제 $S \rightarrow ABe$

$A \rightarrow dB \mid aS \mid c$

$B \rightarrow AS \mid b$

FIRST(S) = { }
FIRST(A) = {a, c, d}
FIRST(B) = {b}

FIRST(S) = FIRST(S)∪(FIRST(A)⊕FIRST(B)⊕FIRST(e))
= { } ∪{a, c, d} ={a, c, d}
FIRST(B) = FIRST(B)∪(FIRST(A)⊕FIRST(S))
= {b}∪{a, c, d} = {a, b, c, d}

다시 반복을 해도 값이 변하지 않으므로
∴ FIRST(S) = {a, c, d}
FIRST(A) = {a, c, d}
FIRST(B) = {a, b, c, d}

예제

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

$FIRST(E) = FIRST(T) = FIRST(F) = \{(, id\}$

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

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

Class Problem

- 다음 문법에서 각 non-terminal 의 FIRST를 각각 구하시오.

1.

$E \rightarrow TE'$

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

$T \rightarrow a$

2.

$S \rightarrow aB \mid Bb$

$B \rightarrow aB \mid \varepsilon$

3.

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

4.

$E \rightarrow a \mid L$

$L \rightarrow (\ Q\ )$

$Q \rightarrow L\ Q'$

$Q' \rightarrow ,\ Q \mid \varepsilon$

FIRST 의 한계

- A→ XXX 과 같은 생성규칙이 있을 때 nonterminal A가 nullable한 경우에서 문제발생
  - FIRST를 가지고는 생성규칙을 결정적으로 선택할 수 없다.
  - 예: S→Ab   A→a A→ ε 라면..
    - a, b 가 들어오면 두 경우 모두 S→Ab를 하면 된다.
    - 그런데 입력이 b라서 그 다음에 A→ ε 를 선택하고 싶을때 ..
    - FIRST(ε)={ε} 말고는 힌트가 없다!
  - A 다음에 나오는 심벌에 따라 어느 생성 규칙으로 유도할 것인가를 결정하는 것이 맞음

그래서, FOLLOW!

- FOLLOW(A)
  - 시작 심벌로 부터 유도될 수 있는 모든 문장 형태에서 A 다음에 나오는 terminal심벌의 집합

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

    $는 입력 스트링의 끝을 표기하는 마커 심벌
  - 모든 문장형태를 고려하기 위해

    => 생성 규칙의 rhs를 이용하여 FOLLOW를 구할 수 있다.

- FOLLOW를 계산하는 방법
  - 시작심벌은 $를 초기값을 갖는다.

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

  - 생성 규칙의 형태가 $A \rightarrow \alpha B \beta$, $\beta \neq \varepsilon$일 때, FIRST(β)에서 ε을 제외한 terminal 심벌을 B의 FOLLOW에 추가

    if $A \rightarrow \alpha B \beta,\ \beta \neq \varepsilon$ then

    $FOLLOW(B) = FOLLOW(B) \cup (FIRST(\beta) - \{\varepsilon\})$

  - $A \rightarrow \alpha B$이거나 $A \rightarrow \alpha B \beta$에서 FIRST($\beta$)에 ε이 속하는 경우 (즉, $\beta \Rightarrow \varepsilon$), A의 FOLLOW 전체를 B의 FOLLOW에 추가

    if $A \rightarrow \alpha B \in P$ or $(A \rightarrow \alpha B \beta$ and $\beta \Rightarrow \varepsilon)$ then

    $FOLLOW(B) = FOLLOW(B) \cup FOLLOW(A)$

- Note)
  - 생성규칙 형태가 $A \rightarrow \alpha B \beta$인 경우, β가 ε이거나 또는 ε을 유도할 수 있으면, A의 FOLLOW전체를 B의 FOLLOW에 넣는다.
    - 임의의 문장 형태에서 A 다음에 나올 수 있는 심벌은 모두 B 다음에 나올 수 있기 때문

$$S \Rightarrow \alpha_1 A \alpha_2 \Rightarrow \alpha_1 \alpha B \beta \alpha_2 \Rightarrow \alpha_1 \alpha B \alpha_2$$

  - FOLLOW속성으로 인하여 $A \rightarrow \alpha B$, $B \rightarrow \alpha A$와 같은 형태의 생성 규칙을 갖고 있으면, FOLLOW(A) = FOLLOW(B)이다.
    - 첫번째 형태의 생성 규칙으로부터 FOLLOW(A) $\subseteq$ FOLLOW(B) 이고 두 번째 형태의 생성 규칙으로부터 FOLLOW(A)$\supseteq$FOLLOW(B)가 되기 때문

예제

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

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

$F \rightarrow (E) \mid id$

FOLLOW(E) = {\$}

FOLLOW(E) $\supset$ FIRST()) = {)}

FOLLOW(F) $\supset$ FIRST(T') = {*} (ε은 제외)

FOLLOW(T) $\supset$ FIRST(E') = {+ } (ε은 제외)

FOLLOW(E) $\subset$ FOLLOW(E')

FOLLOW(T) $\subset$ FOLLOW(T')

FOLLOW(E) $\subset$ FOLLOW(T)

FOLLOW(T) $\subset$ FOLLOW(F)

FOLLOW(E) $\subset$ FOLLOW(T) $\subset$ FOLLOW(F)

FOLLOW(E) = {\$,)}

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

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

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

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

Class Problem

- 다음 문법에서 각 non-terminal 의 FOLLOW를 각각 구하시오.

1.

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow a$

2.

$S \rightarrow aB \mid Bb$
$B \rightarrow aB \mid \varepsilon$

3.

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

4.

$E \rightarrow a \mid L$
$L \rightarrow (\ Q\ )$
$Q \rightarrow L\ Q'$
$Q' \rightarrow\ ,\ Q \mid \varepsilon$

LL 조건

- CFG 가 임의의 생성 규칙 $A \rightarrow \alpha | \beta \in P$에 대하여 만족해야할 다음의 조건
  - $FIRST(\alpha) \cap FIRST(\beta) = \phi$;
  - if $\varepsilon \in FIRST(\alpha)$ then $FOLLOW(A) \cap FIRST(\beta) = \phi$;

➢ 생성 규칙의 FIRST의 공통 원소가 없으므로, 각 순간 적용할 생성 규칙이 유일하게 (결정적으로) 정해짐

➢ 또한 생성 규칙이 ε을 유도 할 수 있으면 FOLLOW에 대해서도 고려하는게 맞으므로 FOLLOW와도 서로 분리된 집합이어야 함

➢ 즉, `**LL 조건을 만족하는 문법 ≅ LL 파싱 되는 문법**' 이란 얘기

LL(1) 문법

- 임의의 생성 규칙 $A \rightarrow \alpha | \beta \in P$에 대하여 LL 조건을 만족하는 CFG
  - ('1') → lookahead 는 하나임 ... 즉, 입력 토큰 하나만 보고 결정
- 위 조건을 따른 다면, 다음과 같은 CFG는 절대 LL(1) 문법이 될 수 있는 여지가 없음.
  1. 모호하거나 (ambiguous)
  2. left-factoring 될 부분을 갖고 있거나
  3. left-recursive 함

1. 모호성 해결 (복습)

- 같은 언어를 생성하는 다른 문법을 사용함
- 예) 연산자 우선순위 주기!

  E→ E+ E| E*E | id 의 모호성 ...

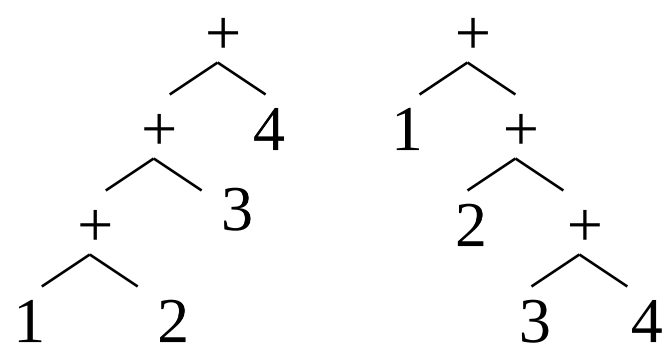
E→ E +T | T
T→T*F | F
F→id

연산자 별로 nonterminal 을 만들고 가장 나중에 수행되야 할 것을 시작심벌과 가깝게 배치

- 예) 결합법칙 반영!

E → E + T
E → T
T → num

E → T + E
E → T
T → num



2. Left-Recursion

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow num$

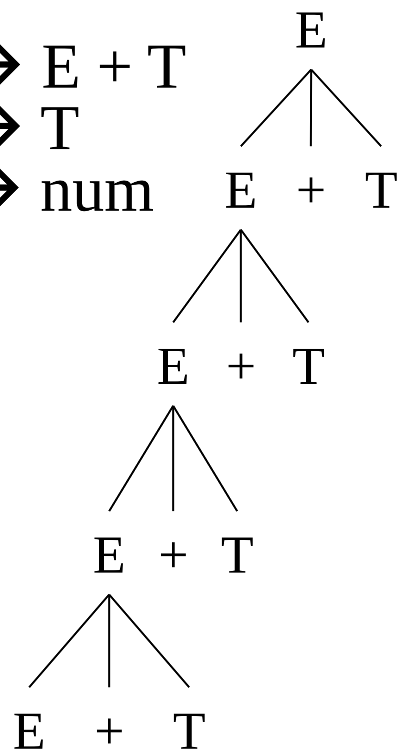
"1 + 2 + 3 + 4"

derived string	lookahead	read/unread
E	1	1+2+3+4
E+T	1	1+2+3+4
E+T+T	1	1+2+3+4
E+T+T+T	1	1+2+3+4
T+T+T+T	1	1+2+3+4
1+T+T+T	2	1+2+3+4
1+2+T+T	3	1+2+3+4
1+2+3+T	4	1+2+3+4
1+2+3+4	$	1+2+3+4

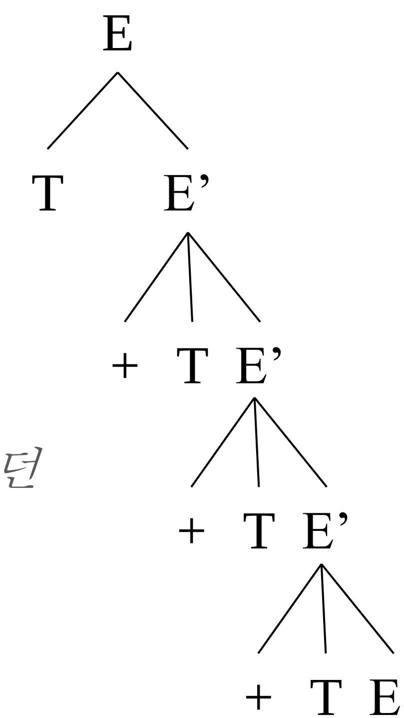
별 문제가 없을까?

- left-recursion
  - 무한 loop의 가능성 => right-recursion으로 해결

$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow num$



$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow num$



그럼 3장에서 모호성 제거에서 나왔던 이것과의 차이는?

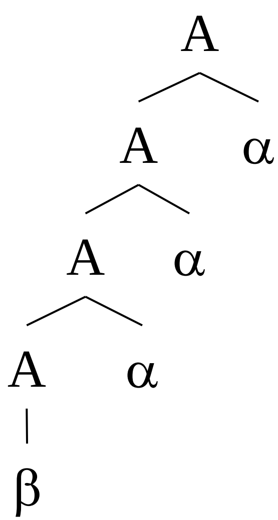
$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow num$

- 직접 left-recursion 제거 방법
  - $A \rightarrow A\alpha$의 형태
  - 즉, lhs의 nonterminal이 rhs의 첫 심벌에 나타날 때 ...

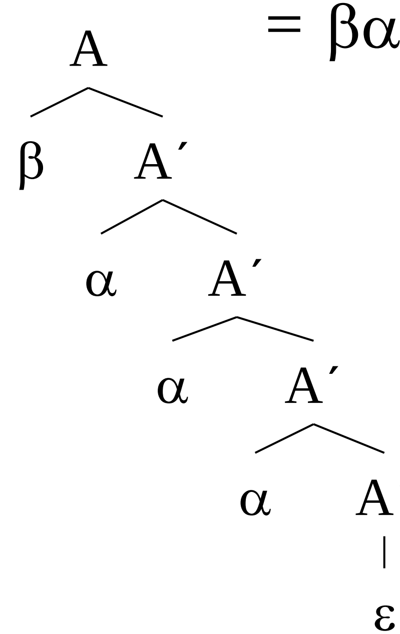
$A \rightarrow A\alpha \mid \beta$
$\Rightarrow A = A\alpha + \beta$
$= \beta\alpha^*$

$A \rightarrow \beta A'$
$A' \rightarrow \alpha A' \mid \varepsilon$

$\Rightarrow A = (\beta\alpha^+ + \beta)$
$= \beta\alpha^+ + \beta$
$= \beta(\alpha^+ + \varepsilon)$
$= \beta\alpha^*$



=>
right-recursion



Class Problem : Left Recursion 제거

- 다음 문법을 left recursion 을 제거한 형태로 변환하시오.

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

- 간접 left-recursion을 직접 left-recursion으로 변환하는 방법
  - 일정한 순서로 nonterminal을 순서화
  - lhs보다 순서적으로 앞선 nonterminal이 rhs의 첫번째로 나오는 생성 규칙을 찾아내면
  - 대입을 해서 간접 recursion을 직접 recursion 으로 바꾸어나감
  - 예

$S \to Aa \mid b$
$A \to Ac \mid Sd \mid e$

$S \to Aa \mid b$
$A \to Ac \mid Aad \mid bd \mid e$

$S \to Aa \mid b$
$A \to (bd \mid e)\, A'$
$A' \to (c \mid ad)\, A' \mid \varepsilon$

$S \to Aa \mid b$
$A \to bdA' \mid eA'$
$A' \to cA' \mid adA' \mid \varepsilon$

3. Left Factoring

- 공통 앞부분(common prefix)을 새로운 nonterminal을 도입하여 인수분해 하는 것.

예제

$A \rightarrow \alpha\beta \mid \alpha\gamma$

$\Longrightarrow \quad A \rightarrow \alpha A'$

$\qquad A' \rightarrow \beta \mid \gamma$

예제

$S \rightarrow iCtS \mid iCtSeS \mid a$

$C \rightarrow b$

$\Longrightarrow \quad S \rightarrow iCtSS' \mid a$

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

$\qquad C \rightarrow b$

예) Left factoring (more)

$E \rightarrow T + E$
$E \rightarrow T$
$T \rightarrow num$
$T \rightarrow (E)$

$E \rightarrow TE'$
$E' \rightarrow \varepsilon$
$E' \rightarrow +E$
$T \rightarrow num$
$T \rightarrow (E)$

LL 중간 정리

- LL 문법이란?
  - LL 조건을 만족하는 CFG
- 주어진 문법에서 LL 문법을 구하기
  - 모호성제거, left-factoring 하기, left-recursion 제거 등을 함
  - 그리고, LL 조건을 만족하는지 확인.
    - $FIRST(\alpha) \cap FIRST(\beta) = \phi$;
    - if $\varepsilon \in FIRST(\alpha)$ then $FOLLOW(A) \cap FIRST(\beta) = \phi$;
- 그 다음에, LL parsing 하기

LOOKAHEAD(...)

- 어떤 규칙이 적용되었을 때 맨 처음 나올 수 있는 terminal symbols
  - cf. FIRST(A), FIRST(a), FIRST($A_1A_2\ldots A_n$) vs. LOOKAHEAD($A\rightarrow\alpha$)

LOOKAHEAD($A\rightarrow\alpha$)

$$= \text{FIRST}(\{\omega \mid S \Rightarrow \mu A\beta \Rightarrow \mu\alpha\beta \Rightarrow \mu\omega \in V_T^*\}$$

LOOKAHEAD($A\rightarrow X_1X_2\ldots X_n$)

$$= \text{FIRST}(X_1)\oplus\text{FIRST}(X_2)\oplus\cdots\oplus\text{FIRST}(X_n)\oplus\text{FOLLOW}(A)$$

- Strong LL(1) 조건

  임의의 택일 규칙 $A\rightarrow\alpha|\beta$ 에 대하여 다음을 strong LL 조건이라 함
  - LOOKAHEAD($A\rightarrow\alpha$) $\cap$ LOOKAHEAD($A\rightarrow\beta$) = ϕ
  - 한마디로, 주어진 상황에서 'LOOKAHEAD' 가 unique 하게 결정
  - LL(1) 과 동일

Class Problem

다음 문법이 LL(1) 인가?

(1)

S→ Abc | aAcb

A→ b|c| ε

(2)

S → aAS | b

A → a | bSA

LL(1) 파서 구현 방법

1. Recursive descent parser
   - recursion 이용
   - 각 non-terminal 마다 한개의 procedure를 둠
   - 장점 : 직관적, 쉽다.
   - 단점: 생성규칙이 바뀌면 구문 분석기를 고쳐야함

2. Predictive parser
   - 이론적으로 PDA (push down automata) 에 기반
   - 생성 규칙이 바뀌더라도 구문 분석기는 고치지 않음
     - 단지 구문 분석기의 행동을 나타내는 파싱 테이블만 수정

1. Recursive Descent Parser

- 모든 terminal 심볼 a 에 대한 파서 코드

```
procedure pa;
begin
   if nextSymbol = ta then get_nextSymbol
                         else error
end; /*pa*/
```

- 모든 non-terminal 심볼 A 에 대한 파서 코드

```
procedure pA;
begin
   case nextSymbol of
     LOOKAHEAD(A→X1X2...Xm):  for i:=1 to m do pXi;
     LOOKAHEAD(A→Y1Y2...Yn):  for i:=1 to n do pYi;
     LOOKAHEAD(A→Z1Z2...Zr):  for i:=1 to r do pZi;
     LOOKAHEAD(A→ε ): ;
     otherwise : error
   end
end; /*pA*/
```

S→ aAb
A→ aS | b

예) Recursive Descent Parser

(1) Terminal symbols

```
void pa(){
    if (nextSymbol == ta )
            nextSymbol = get_nextSymbol();
      else error();
}
void pb() {
    if (nextSymbol == tb )
            nextSymbol = get_nextSymbol();
      else error
}
```

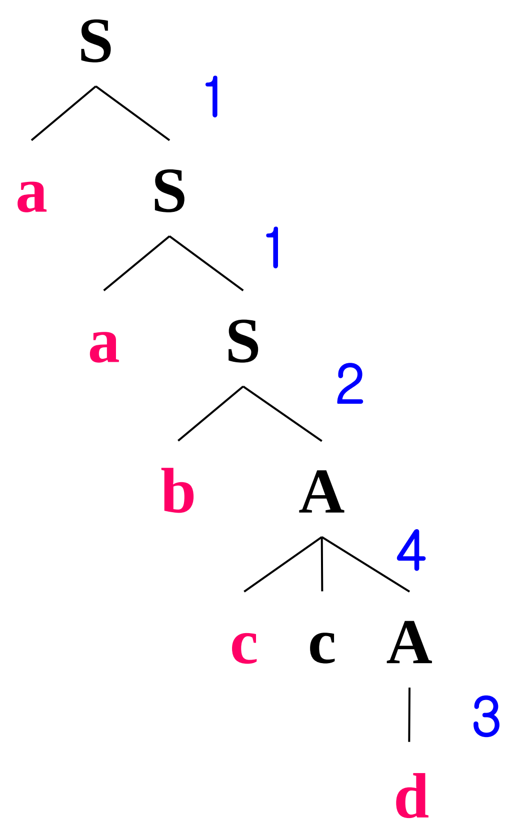
S→ aAb

A→ aS | b

(2) Non-terminal Symbols

```
void pS() {
    if (nextSymbol == ta) {
            pa(); pA(); pb();
    }
}
void pA() {
    switch (nextSymbol) {
      case ta : pa();pS(); break;
      case tb : pb(); break;
      default: error();
    }
}
```

2. Predictive Parser



- 테이블로 파싱하기 !
  - LL(1)은 lookahead 에 따라 적용할 생성규칙이 unique 하게 결정되므로
  - 주어진 문자열에 대해 기계적으로 파싱할 수 있다.
  - 예) 1. $S \rightarrow aS$ 2. $S \rightarrow bA$

    3.$A \rightarrow d$ 4. $A \rightarrow ccA$

	a	b	c	d
S	1	2		
A			4	3

파싱 Table 만들기

- 파싱테이블
  - Nonterminal X Terminal = 적용될 규칙

- 파싱테이블 구성 원칙

  "$a \in$ Lookahead($A \rightarrow \alpha$) 이면

  $M[A, a] := A \rightarrow \alpha$"
- 이런 절차로 구성
  - 모든 $a \in FIRST(\alpha)$에 대하여, $M[A, a] := A \rightarrow \alpha$로 채운다.
  - 만일 $\varepsilon \in FIRST(\alpha)$이면, 모든 $b \in FOLLOW(A)$에 대하여, $M[A,b] := A \rightarrow \alpha$로 채운다.

예제

1. $E \rightarrow TE'$
2. $E' \rightarrow +TE'$
3. $E' \rightarrow \varepsilon$
4. $T \rightarrow FT'$
5. $T' \rightarrow *FT'$
6. $T' \rightarrow \varepsilon$
7. $F \rightarrow (E)$
8. $F \rightarrow id$

(1) FIRST(E) = FIRST(T) = FIRST(F) = {(,id}
FIRST(E') = {+ , ε}
FIRST(T') = {*, ε}

(2) FOLLOW(E) = FOLLOW(E') = {), $}
FOLLOW(T) = FOLLOW(T') = {+ ,), $}
FOLLOW(F) = {+ , *,), $}

(3) 파싱테이블

	Id	+	*	(	)	$
E	1			1		
E'		2			3	3
T	4			4		
T'		6	5		6	6
F	8			7		

모호성과 테이블

- 파싱 테이블 한칸에 두개 이상의 생성규칙이 들어갈 수 있을 때
  - NOT LL(1) ... 결정적선택 불가
  - 예

    1. $S \rightarrow iCtSS'$ 2. $S \rightarrow a$ 3. $S' \rightarrow eS$

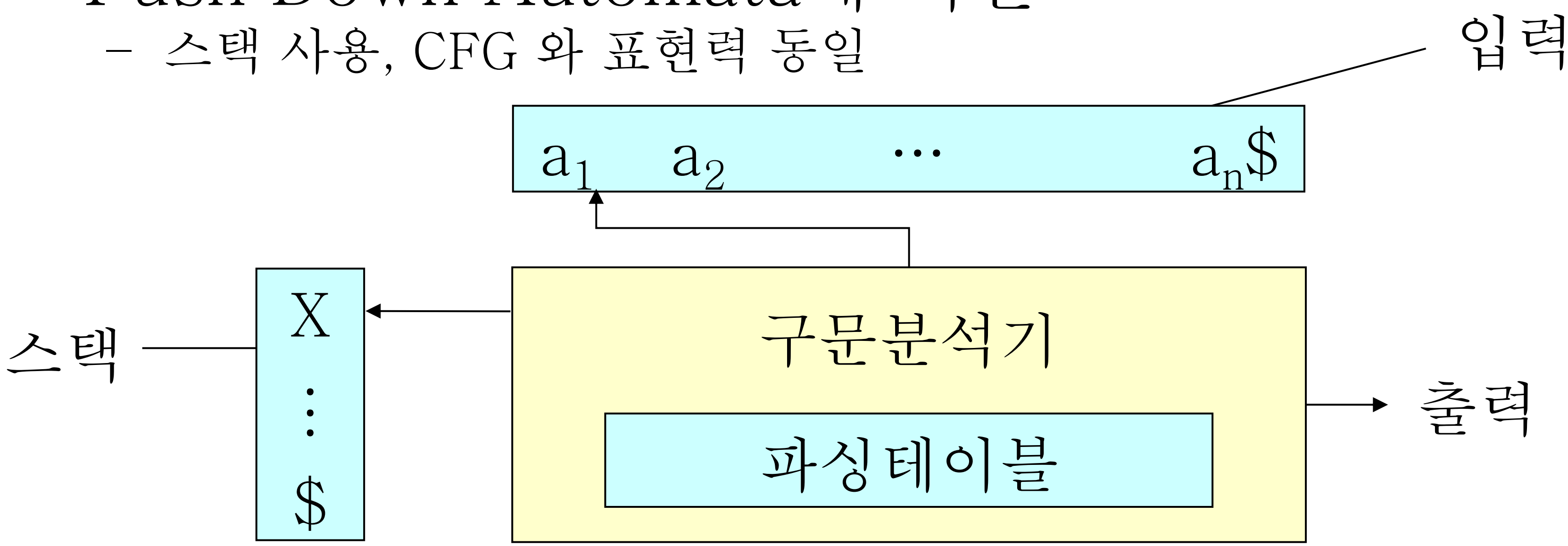
    4. $S' \rightarrow \varepsilon$ 5. $C \rightarrow b$

    - FIRST(S) = {i,a}
    - FIRST(S') = {e, ε}
    - FIRST(C)= {b}
    - FOLLOW(S)= {e, $}
    - FOLLOW(S')= {e, $}
    - FOLLOW(C)= {t}

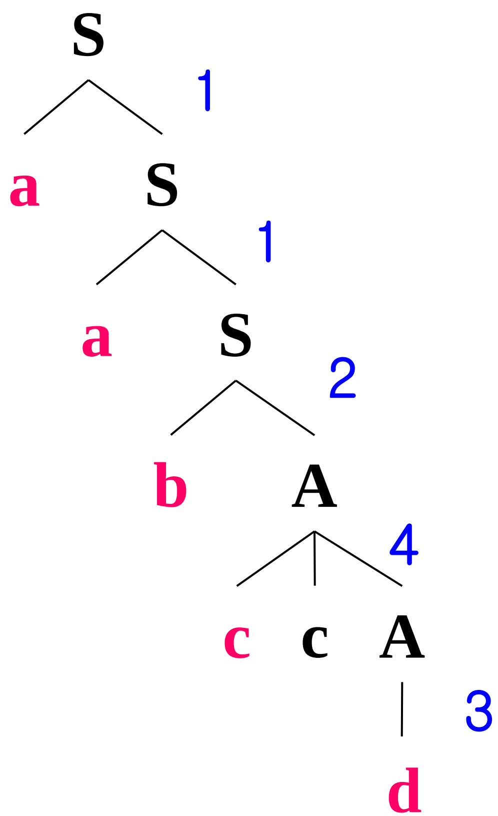
	a	b	e	i	t	$
S	2			1		
S'			3,4			4
C		5				

기계적 파서 구조

- 생성 규칙이 바뀌더라도 구문 분석기는 고치지 않게
  - 파싱테이블만 바꿈
- 스택을 '연습장'으로 사용
  - '(()(()))' 이런 것 인식할 때 무슨 자료구조를 사용하나?
- Push Down Automata에 기반
  - 스택 사용, CFG 와 표현력 동일



예) 구문 분석 (좌측유도) 복습



1. $S \rightarrow aS$
2. $S \rightarrow bA$
3. $A \rightarrow d$
4. $A \rightarrow ccA$

예) aabccd 에 대한 좌측 유도

$$
\begin{aligned}
S &\Rightarrow a\underline{S} && \text{aabccd } (1) \\
&\Rightarrow aa\underline{S} && \text{aabccd } (11) \\
&\Rightarrow aab\underline{A} && \text{aabccd } (112) \\
&\Rightarrow aabcc\underline{A} && \text{aabccd } (1124) \\
&\Rightarrow aabccd && \text{aabccd } (11243)
\end{aligned}
$$

구문 분석기 Action

- pop(제거)
  - stack의 top = 입력 symbol
  - stack의 top심벌은 stack에서 제거하고 현재 입력 심벌은 입력 스트링에서 제거
- expand(확장)
  - stack의 top이 nonterminal인 경우로 생성 규칙을 적용하여 확장
  - 예를 들어, M[A, a] = {A→XYZ}일때, A를 스택에서 제거하고 ZYX를 차례로 스택에 넣는다.

구문 분석기 Action (계속)

- accept(인식)
  - stack의 top심벌과 현재 입력 심벌 모두가 $인 경우
  - 주어진 입력 스트링이 올바른 스트링임을 알림
- error(오류)
  - stack의 top이 nonterminal 심벌인 경우
  - 그 심벌로부터 현재 보고 있는 입력 심벌을 결코 유도할 수 없음

스택의 내용	입력 스트링	파싱 행동	파스
$S	aabccd$	expand 1	1
$Sa	aabccd$	pop & advance	1
$S	abccd$	expand 1	11
$Sa	abccd$	pop & advance	11
$S	bccd$	expand 2	112
$Ab	bccd$	pop & advance	112
$A	ccd$	expand 4	1124
$Acc	ccd$	pop & advance	1124
$Ac	cd$	pop & advance	1124
$A	d$	expand 3	11243
$d	d$	pop & advance	11243
$	$	accept	11243

Class Problem

- 다음과 같은 LL(1) 문법의 파싱테이블을 작성하시오.

  $S \rightarrow (L) \mid a$

  $L \rightarrow SL'$

  $L' \rightarrow ,SL' \mid \varepsilon$

  단, FIRST(S) = FIRST(L) = {(, a}, FIRST(L') = {, , ε}

  FOLLOW(S) = {$, , ,) } FOLLOW(L) = FOLLOW(L') = {) }

- (a,a)가 문법을 만족하는지 알아보는 파싱과정을 스택과 입력, 파싱행동, 파스로 표현하시오.

스택의 내용	입력 스트링	파싱 행동	파스
$S …..	(a,a)$ …	…	…