KNU 4471.043 컴파일러 설계

고상기

6주차

2022 Spring

강원대학교 컴퓨터공학과

5주차 요약

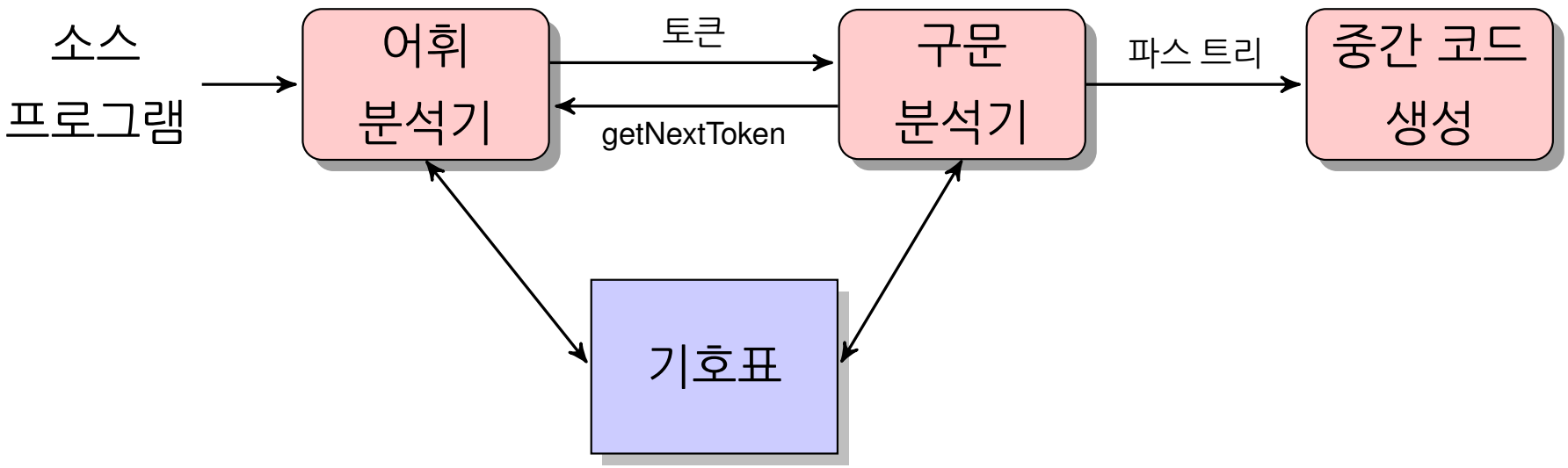
- 문맥-자유 문법context-free grammar
- 파스 트리
- 모호한 문법ambiguous grammar
- 문법 변환하기

6주차 개요

- 구문 분석 방법의 분류
- 재귀-하강 구문 분석recursive-descent parsing
- 예측 구문 분석predictive parsing
- 이동-감축 구문 분석shift-reduce parsing

구문 분석 과정

- 주어진 입력이 올바른 프로그램인가를 검사하고 다음 단계에서 필요한 정보를 구성하는 과정
- 스캐너에 의해 생성된 토큰을 입력으로 받아 주어진 문법에 맞는지를 검사하고, 문법에 맞으면 파스 트리를 생성하고 맞지 않으면 오류를 출력하는 단계
- 구문 분석을 담당하는 도구를 구문 분석기$_{\text{syntax analyzer}}$ 혹은 파서$_{\text{parser}}$ 라고 함



구문 분석 방법의 분류

하향식 구문 분석top-down parsing

- 입력 문자열을 한번에 한 기호씩 좌에서 우로 검사left-to-right scanning하면서 루트 노드에서 시작하여 터미널 노드로 만들어나가는 방법이다.
- 즉, 하향식 구문 분석은 시작 기호 S로부터 문법의 규칙을 적용하여 왼쪽 유도에 의해 주어진 문장을 찾아가는 방법
- 하향식 구문 분석의 경우 왼쪽 유도에 의한 분석 결과가 생성된다.

상향식 구문 분석bottom-up parsing

- 입력 문자열을 한번에 한 기호씩 좌에서 우로 검사left-to-right scanning하면서 터미널 노드에서 시작하여 루트 노드로 만들어나가는 방법이다.
- 상향식 구문 분석은 입력된 문장에서 감축에 의해 시작 기호를 찾아가는 방법으로 오른쪽 유도의 역순이 된다.
- 상향식 구문 분석의 경우 오른쪽 유도의 역순에 해당하는 분석 결과가 생성된다.

하향식 구문 분석top-down parsing

- 시작 기호로부터 생성 규칙에 왼쪽 유도만을 적용하는 방법
- 구현이 간편하여 구문 분석을 공부할 때는 자주 사용하지만 역추적backtracking 문제로 인해 상업적인 용도로는 거의 사용되지 않음
- 다음과 같은 과정을 거친다.
  1. 시작 기호에 대해 생성 규칙을 적용. 이때 생성 규칙이 여러 개 존재하면 첫 번째 생성 규칙부터 적용
  2. 생성된 문장 형태의 문자열과 입력 기호를 차례로 비교
  3. 만약 유도된 문자열과 입력 기호가 같으면 계속 비교
  4. 비교한 두 기호가 같지 않으면 생성 규칙을 잘못 적용한 것이므로 현재 적용된 생성 규칙을 제거하고 다른 생성 규칙을 적용. 이때 입력 기호의 위치는 제거한 생성 규칙에서 보았던 기호의 개수만큼 뒤로 이동하는데 이를 역추적이라 한다.
  5. 이 과정을 반복하다가 더 이상 적용할 생성 규칙이 없으면 주어진 문장을 주어진 문법에 올바르지 않은 문장으로 인식하고 오류 메시지를 출력
  6. 생성된 문자열이 주어진 문장과 일치하면 올바른 문장으로 인식하고 파스 트리 출력

하향식 구문 분석 예

- 다음 문법에 대해 문자열 *cabdd*가 문법에 맞는 문장인지 하향식 구문 분석 방법을 통해 검사하고 이에 대한 파스 트리를 만들어보자.

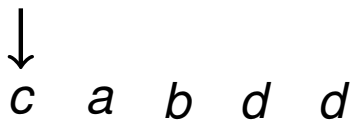
$S \rightarrow cDd$

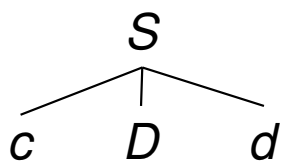
$D \rightarrow a$

$D \rightarrow aE$

$E \rightarrow bb$

$E \rightarrow bd$





하향식 구문 분석 예

- 다음 문법에 대해 문자열 *cabdd*가 문법에 맞는 문장인지 하향식 구문 분석 방법을 통해 검사하고 이에 대한 파스 트리를 만들어보자.

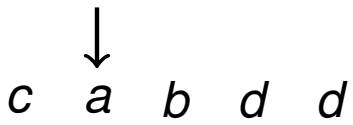
$S \rightarrow cDd$

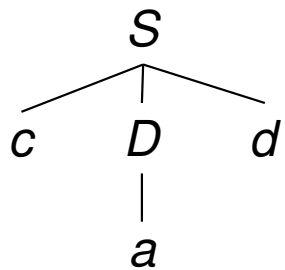
$D \rightarrow a$

$D \rightarrow aE$

$E \rightarrow bb$

$E \rightarrow bd$





하향식 구문 분석 예

- 다음 문법에 대해 문자열 *cabdd*가 문법에 맞는 문장인지 하향식 구문 분석 방법을 통해 검사하고 이에 대한 파스 트리를 만들어보자.

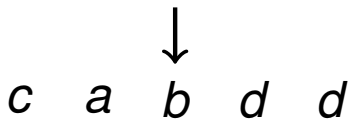
$S \rightarrow cDd$

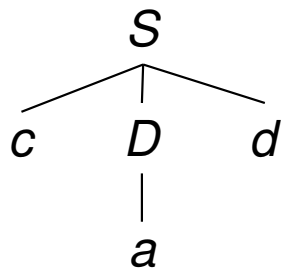
$D \rightarrow a$

$D \rightarrow aE$

$E \rightarrow bb$

$E \rightarrow bd$





하향식 구문 분석 예

- 다음 문법에 대해 문자열 *cabdd*가 문법에 맞는 문장인지 하향식 구문 분석 방법을 통해 검사하고 이에 대한 파스 트리를 만들어보자.

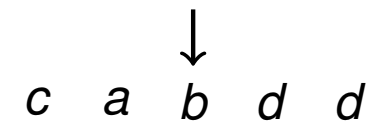
$S \rightarrow cDd$

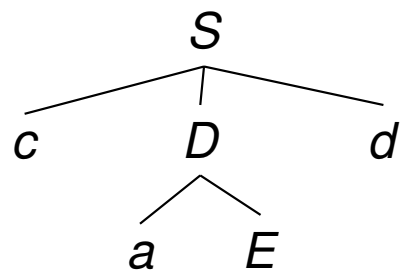
$D \rightarrow a$

$D \rightarrow aE$

$E \rightarrow bb$

$E \rightarrow bd$





하향식 구문 분석 예

- 다음 문법에 대해 문자열 *cabdd*가 문법에 맞는 문장인지 하향식 구문 분석 방법을 통해 검사하고 이에 대한 파스 트리를 만들어보자.

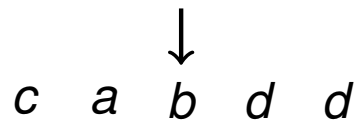
$S \rightarrow cDd$

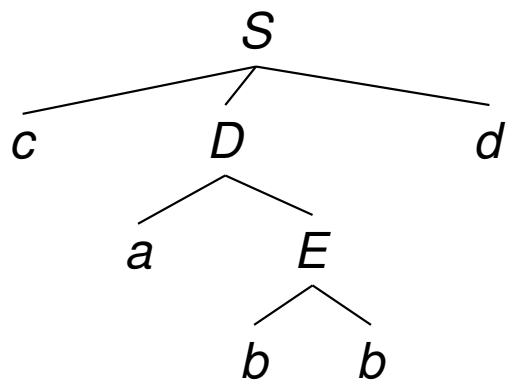
$D \rightarrow a$

$D \rightarrow aE$

$E \rightarrow bb$

$E \rightarrow bd$





하향식 구문 분석 예

- 다음 문법에 대해 문자열 *cabdd*가 문법에 맞는 문장인지 하향식 구문 분석 방법을 통해 검사하고 이에 대한 파스 트리를 만들어보자.

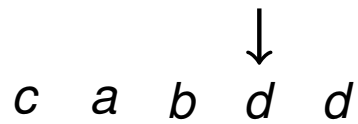
$S \rightarrow cDd$

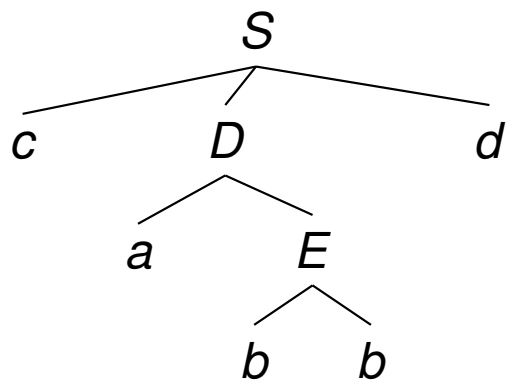
$D \rightarrow a$

$D \rightarrow aE$

$E \rightarrow bb$

$E \rightarrow bd$





하향식 구문 분석 예

- 다음 문법에 대해 문자열 *cabdd*가 문법에 맞는 문장인지 하향식 구문 분석 방법을 통해 검사하고 이에 대한 파스 트리를 만들어보자.

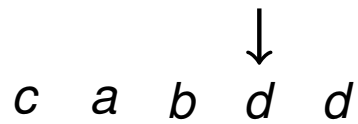
$S \rightarrow cDd$

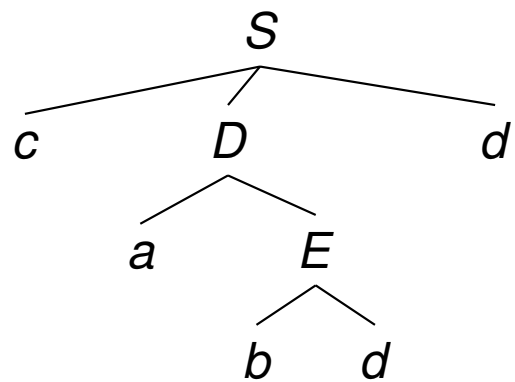
$D \rightarrow a$

$D \rightarrow aE$

$E \rightarrow bb$

$E \rightarrow bd$





하향식 구문 분석 예

- 다음 문법에 대해 문자열 *cabdd*가 문법에 맞는 문장인지 하향식 구문 분석 방법을 통해 검사하고 이에 대한 파스 트리를 만들어보자.

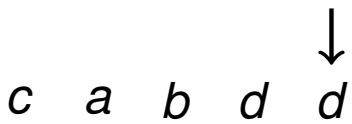
$S \rightarrow cDd$

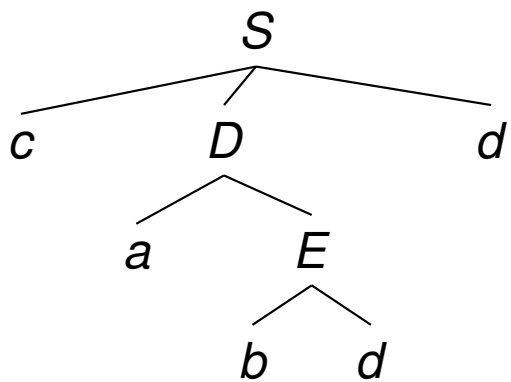
$D \rightarrow a$

$D \rightarrow aE$

$E \rightarrow bb$

$E \rightarrow bd$





역추적의 문제점

- 이 과정을 역추적이라 하는데, 한 입력 기호를 여러 번 반복해서 검조하기 때문에 시간이 아주 많이 소요되어 비효율적
- 결정적deterministic으로 구문 분석을 할 수 있는 방법이 필요
- 주어진 문맥-자유 문법이 특정 조건을 만족하면 결정적인 구문 분석이 가능
  - 이러한 조건을 LL(1)Left-to-right, Leftmost derivation 조건이라 함
  - 왼쪽에서 오른쪽으로 읽어가며left-to-right scanning 왼쪽 유도를 진행한다는 의미
  - LL 조건을 만족하는 문법을 LL 문법이라 함
  - FIRST와 FOLLOW를 이용
- 주어진 문자열과 같은 문장을 생성하기 위해 현재의 입력 기호를 보고 적용될 생성 규칙을 결정적으로 선택하여 유도하는 방식
- 현재 위치에서 터미널 기호를 미리 k개를 보고 결정적 구문 분석을 할 수 있는 문법을 LL(k) 문법이라 한다.

$\mathrm{LL}(k)$ 문법

- 문자열의 현재 위치에서 k개의 터미널 기호를 보고 결정적으로 생성 규칙을 선택할 수 있는 문법을 $\mathrm{LL}(k)$ 문법이라 한다.
- 다음 문법은 $\mathrm{LL}(1)$ 문법인가?

$$S \to aAb \mid aBb \mid acc$$

$$A \to a \mid Acc$$

$$B \to b$$

- 위 문법은 $\mathrm{LL}(2)$ 문법인가?
- $k \geq 1$일 때, $\mathrm{LL}(k)$ 문법은 $\mathrm{LL}(k+1)$ 문법이다.

FIRST 집합 정의

Definition

문법 $G = (V_N, V_T, P, S)$가 문맥-자유 문법일 때 논터미널 기호 A에 대한 FIRST 집합은 다음과 같이 정의된다.

$$\text{FIRST}(A) = \{a \in V_T \cup \{\varepsilon\} \mid A \overset{*}{\Rightarrow} a\beta, \beta \in V^*\}.$$

즉, 논터미널 기호 A가 문장으로 유도되었을 때 첫 번째로 나타날 수 있는 터미널 기호의 집합을 의미한다.

- 다음 문법에서 각 논터미널 기호에 대한 FIRST 집합을 구해보자.

$$S \to A \mid B$$

$$A \to aA \mid b$$

$$B \to cB \mid d$$

FIRST 집합 구하기 예

- 다음 문법에서 각 논터미널 기호에 대한 FIRST 집합을 구해보자.

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

nullable한 기호

Definition

논터미널 기호 A가 ε을 유도할 수 있으면 A를 nullable하다고 한다. 즉, $A \overset{*}{\Rightarrow} \varepsilon$이면 A는 nullable하다.

- 다음 문법에서 논터미널 기호 E는 nullable한지 확인해보자.

$$E \to TE'$$

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

$$T \to FT'$$

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

$$F \to (E) \mid id$$

nullable 검사 예

- 다음 문법에서 논터미널 기호 S는 nullable한지 확인해보자.

$$S \rightarrow AB$$

$$A \rightarrow dB \mid \varepsilon$$

$$B \rightarrow A \mid \varepsilon$$

FIRST 집합 계산 규칙

Definition (ring sum 연산자 $\oplus$)

만약 $\varepsilon \notin A$ 이면, $A \oplus B = A$이고, $\varepsilon \in A$이면 $A \oplus B = (A - \{\varepsilon\}) \cup B$이다.

- ring sum 연산자 '$\oplus$'는 첫 번째 피연산자 집합이 ε을 가지고 있지 않으면 그 자신이 되고, ε을 가지고 있다면 ε을 제외한 첫 번째 피연산자와 두 번째 피연산자의 합집합이 된다.
- 이 때, 다음과 같이 FIRST 집합을 정의한다.

Definition

$$\mathrm{FIRST}(\mathrm{A_1 A_2 \cdots A_n}) = \mathrm{FIRST}(\mathrm{A_1}) \oplus \mathrm{FIRST}(\mathrm{A_2}) \oplus \cdots \oplus \mathrm{FIRST}(\mathrm{A_n})$$

FIRST 집합 계산 규칙

- 이전 슬라이드의 형식적 정의를 문장으로 옮기면 다음과 같다.
- 생성 규칙의 오른쪽에 있는 첫 번째 기호가 nullable하면 그 문자열 다음 기호의 FIRST 집합도 해당 문자열의 FIRST에 속한다.
- 이렇게 nullable이 아닐 때까지 오른쪽으로 이동하면서 기호들의 FIRST 집합의 합집합을 구한 것이 논터미널 기호의 FIRST이 된다.
- $X \in V$에 대해, $\mathrm{FIRST}(X)$는 다음과 같이 계산할 수 있다.
  1. 만약 X가 터미널 기호라면 $\mathrm{FIRST}(X) = \{X\}$이다.
  2. 만약 $X \to a\beta$와 같은 생성 규칙이 존재한다면, $a \in \mathrm{FIRST}(X)$이다.
  3. 만약 $X \to \varepsilon$과 같은 생성 규칙이 존재한다면, $\varepsilon \in \mathrm{FIRST}(X)$이다.
  4. 만약 $X \to Y_1 Y_2 \cdots Y_K$와 같은 생성 규칙이 존재한다면, $\mathrm{FIRST}(X) = \mathrm{FIRST}(X) \cup \mathrm{FIRST}(Y_1 Y_2 \cdots Y_k)$이다.

FIRST 집합 구하기 예 2

- 다음 문법에서 다시 한번 각 논터미널 기호에 대한 FIRST 집합을 구해보자.

$$S \to A \mid B$$

$$A \to aA \mid b$$

$$B \to cB \mid d$$

- 다음 문법에서 다시 한번 각 논터미널 기호에 대한 FIRST 집합을 구해보자.

$$E \to TE'$$

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

$$T \to FT'$$

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

$$F \to (E) \mid id$$

FOLLOW 집합 정의

Definition

문법 $G = (V_N, V_T, P, S)$가 문맥-자유 문법일 때, 논터미널 기호 A의 $\text{FOLLOW}(A)$는 다음과 같이 정의된다.

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

즉, $\text{FOLLOW}(A)$는 어떤 문장 형태에서 논터미널 기호 A 다음에 나오는 터미널 기호의 집합이다. 여기서 $는 입력 문자열의 끝을 나타내는 기호이고, 시작 기호 S에 대한 $\text{FOLLOW}(S)$는 $를 포함한다.

- $\text{FOLLOW}(A)$를 계산하는 규칙은 다음과 같다.
  1. A가 시작 기호이면 , $\$ \in \text{FOLLOW}(A)$이다.
  2. $B \to \alpha A \beta, \beta \neq \varepsilon$인 생성 규칙이 있다면, $(\text{FIRST}(\beta) - \{\varepsilon\}) \subseteq \text{FOLLOW}(A)$이다.
  3. $B \to \alpha A$인 생성 규칙이 있거나, $\varepsilon \in \text{FIRST}(\beta)$일 때 $B \to \alpha A \beta$와 같은 생성 규칙이 있다면, $\text{FOLLOW}(B) \subseteq \text{FOLLOW}(A)$이다.

FOLLOW 집합 구하기 예

- 다음 문법에서 각 논터미널 기호에 대한 FOLLOW 집합을 구해보자.

$$S \to A \mid B$$

$$A \to aA \mid b$$

$$B \to cB \mid d$$

- 다음 문법에서 각 논터미널 기호에 대한 FOLLOW 집합을 구해보자.

$$E \to TE'$$

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

$$T \to FT'$$

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

$$F \to (E) \mid id$$

LL(1) 조건

Definition

LL(1) 조건은 임의의 생성 규칙 $A \to \alpha \mid \beta \in P$에 대해 다음 조건을 만족해야 한다.

- $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \emptyset$이고,
- 만약 $\varepsilon \in \text{FIRST}(\alpha)$이면, $\text{FOLLOW}(A) \cap \text{FIRST}(\beta) = \emptyset$이다.

- 다음 문법이 LL(1) 조건을 만족하는지 알아보자.

$$E \to TE'$$

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

$$T \to FT'$$

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

$$F \to (E) \mid id$$

재귀-하강 구문 분석recursive-descent parsing

- LL 문법을 사용하는 하향식 구문 분석 방법
  - 재귀-하강 구문 분석recursive-descent parsing
  - 예측 구문 분석predictive parsing

- 재귀-하강 구문 분석 방법은 주어진 입력 문자열을 구문 분석하기 위해 재귀적 프로시저를 사용

- 재귀적 프로시저는 각 논터미널에 해당하는 것으로 논터미널에 대한 유도를 재귀적 프로시저 호출로 처리

재귀-하강 구문 분석기 특징

- 재귀적 프로시저recursive procedure의 집합으로 구문 분석을 수행
- 각 논터미널 기호에 대한 프로시저를 작성하고 터미널 기호에 대한 프로시저를 작성한 다음 이를 통합하여 구현
- 문법으로부터 쉽고 빠르게 구현할 수 있다는 장점이 있으나 프로시저를 호출하는 시간이 많이 걸려서 느리고 생성 규칙이 바뀔 때마다 구문 분석기를 새로 작성해야 한다.
- 아래 예제를 살펴보자.

$$
\begin{aligned}
\langle \text{expr} \rangle &::= \langle \text{term} \rangle \{ (+ \mid -) \langle \text{term} \rangle \} \\
\langle \text{term} \rangle &::= \langle \text{factor} \rangle \{ (* \mid /) \langle \text{factor} \rangle \} \\
\langle \text{factor} \rangle &::= \text{`(' } \langle \text{expr} \rangle \text{ `)'} \mid \langle \text{id} \rangle \mid \langle \text{number} \rangle
\end{aligned}
$$

재귀-하강 프로시저 예

- `lex()` 함수는 입력으로부터 다음 토큰을 받아 `nextToken`에 변수에 토큰 번호를 저장하는 역할을 한다고 가정하자.
- 다음 프로시저는 논터미널 기호 ⟨expr⟩를 파싱한다.

```
1 void expr() {
2     printf("Enter␣<expr>\n");
3     term();
4     while (nextToken == ADD_OP || nextToken == SUB_OP) {
5         lex();
6         term();
7     }
8     printf("Exit␣<expr>\n");
9 }
```

재귀-하강 프로시저 예

- 다음 프로시저는 ⟨term⟩을 파싱한다.

```
1 void term() {
2     printf("Enter␣<term>\n");
3     factor();
4     while (nextToken == MULT_OP || nextToken == DIV_OP) {
5         lex();
6         factor();
7     }
8     printf("Exit␣<term>\n");
9 }
```

재귀-하강 프로시저 예

- 다음 프로시저는 ⟨factor⟩를 파싱한다.

```
1  void factor() {
2      printf("Enter <factor>\n");
3      if (nextToken == IDENT || nextToken == INT_LIT)
4          lex();
5      else if (nextToken == LEFT_PAREN) {
6          lex();
7          expr();
8          if (nextToken == RIGHT_PAREN)
9              lex();
10         else
11             error();
12     } else
13         error();
14     printf("Exit <factor>\n");;
15 }
```

재귀-하강 프로시저 작성 예

- 작성된 재귀-하강 프로시저를 통해 아래 문장을 파싱해보자.

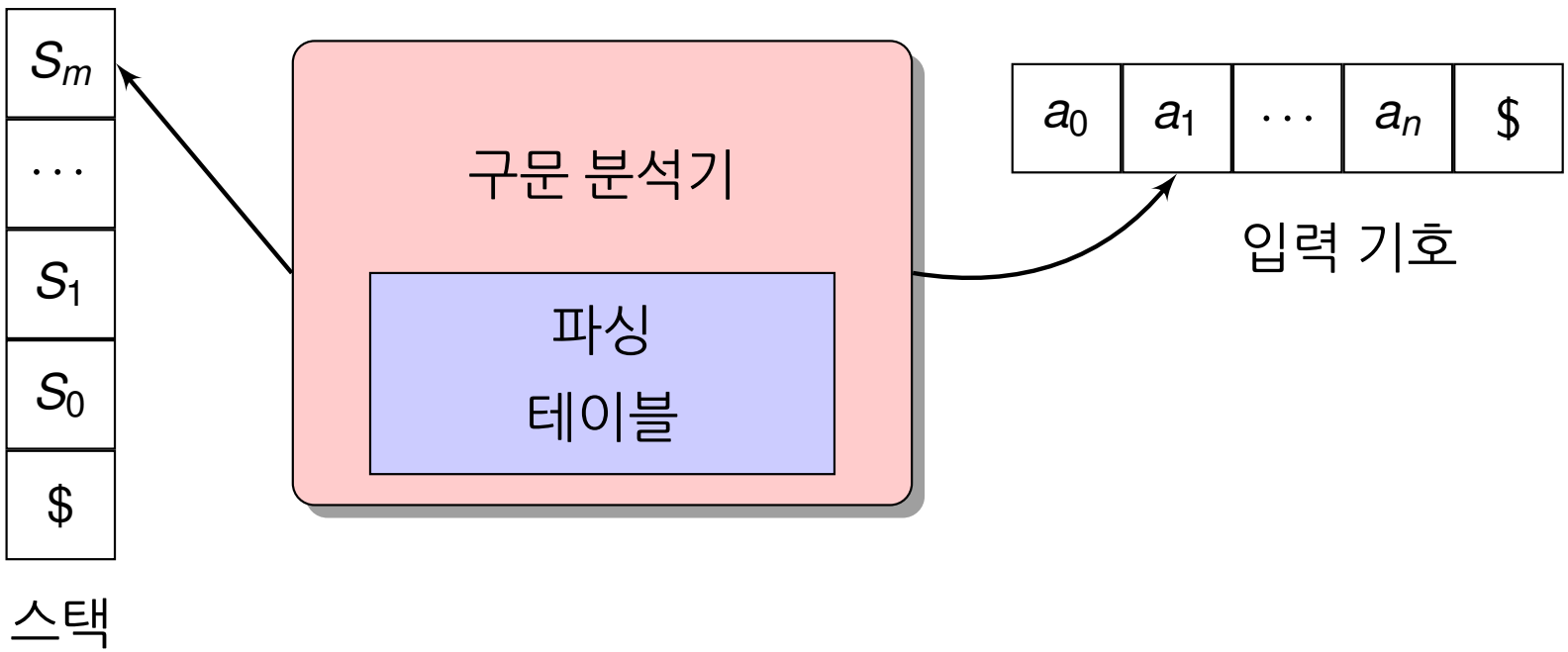
$$(\texttt{num} + 1) * 2 + 3$$

- 다음 구문 규칙에 대한 재귀-하강 프로시저를 작성해보자.

$$\langle \text{if\_stmt} \rangle \quad ::= \quad \texttt{if} \; (\; \langle \text{exp} \rangle \;) \; \langle \text{stmt} \rangle \; [\; \texttt{else} \; \langle \text{stmt} \rangle \;]$$

예측 구문 분석predictive parsing

- 생성 규칙이 바뀌더라도 전체 구문분석기를 고치지 않고 단지 구문 분석기의 행동을 제어하는 파싱 테이블parsing table만 수정해서 구문 분석기를 구현
- 입력은 구문 분석이 될 입력 문자열과 $을 저장하고, 스택은 문장 형태에서 입력 문자열과 아직 매칭되지 않은 부분을 유지



예측 구문 분석기의 행동

- 주어진 입력 문자열을 왼쪽에서 오른쪽으로 읽고, 현재 입력 기호와 스택의 톱 기호에 따라 파싱표에서 주어진 행동을 취하며 구문 분석을 수행

- 예측 구문 분석기의 행동은 제거, 확장, 인식, 오류이며 그 의미는 다음과 같다.
  - 제거$_{pop}$: 스택의 톱과 현재의 입력 기호가 같은 경우로, 스택의 톱 기호를 제거하고 현재의 입력 기호를 입력 문자열에서 제거
  - 확장$_{expand}$: 스택의 톱이 논터미널 기호인 경우로, 생성 규칙을 적용하여 확장한다. 즉 생성 규칙의 왼쪽 기호 대신에 오른쪽 문자열로 대치
  - 인식$_{accept}$: 스택의 톱과 현재의 입력 기호가 모두 $인 경우로, 입력 문자열이 올바른 문자열임을 알려줌
  - 오류$_{error}$: 스택의 톱이 논터미널 기호인 경우로, 그 기호로부터 현재 보고 있는 기호를 유도할 수 없음을 알려줌

예측 파싱표 구성하기

- 주어진 LL 문법 $G = (V_N, V_T, P, S)$에 대해 예측 파싱표 M을 구성하는 방법은 다음과 같다.
  1. $\text{FOLLOW}(A)$와 $\text{FIRST}(\alpha)$를 계산한다.
  2. 만약 $a \in \text{FIRST}(\alpha)$라면, 파싱표를 다음과 같이 작성한다.

$$M[A, a] = A \to \alpha$$

  3. 만약 $\varepsilon \in \text{FIRST}(\alpha)$이고 $b \in \text{FOLLOW}(A)$라면, 파싱표를 다음과 같이 작성한다.

$$M[A, b] = A \to \alpha$$

예측 파싱표 구성하기 예

- 다음 문법에 대한 예측 파싱표를 구성해보자.

$$E \to TE'$$

$$E' \to +TE'$$

$$E' \to \varepsilon$$

$$T \to FT'$$

$$T' \to *FT'$$

$$T' \to \varepsilon$$

$$F \to (E)$$

$$F \to id$$

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

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

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

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

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

	id	+	*	(	)	$
E						
E'						
T						
T'						
F						

예측 파싱표 구성하기 예 2

- 다음 문법에 대한 예측 파싱표를 구성해보자.

$S \to iCtSS'$

$S \to a$

$S' \to eS$

$S' \to \varepsilon$

$C \to b$

	a	b	e	i	t	\$
S						
S'						
C						

예측 구문 분석기의 행동

- 문맥-자유 문법 $G = (V_N, V_T, P, S)$, 문법으로부터 생성된 파싱 테이블 M, 그리고 문자열 w가 주어졌다고 가정하자.
- 예측 구문 분석기는 $w \in L(G)$인 경우 w에 대한 왼쪽 유도를 생성하고, 아닌 경우 오류를 출력한다.
- 예측 구문 분석기의 행동은 다음과 같다.
  1. 문법의 시작 기호를 스택에 넣고, 입력 문자열 w부터 분석을 시작한다.
  2. 스택의 톱 기호를 X라 하고 현재 입력 기호를 a라 하자.
  3. 만약 $X \in V_T \cup \{\$\}$라면,
     - 3.1 $X = a$라면, X를 스택에서 제거하고 문자열의 다음 기호를 처리한다.
     - 3.2 $X \neq a$라면, 오류를 출력한다.
  4. 만약 $X \in V_N$이고 $M[X, a] = X \to Y_1 Y_2 \cdots Y_k$라면,
     - 4.1 X를 스택에서 제거하고 $Y_k, Y_{k-1}, \ldots, Y_2, Y_1$를 순서대로 스택에 넣는다.
     - 4.2 생성 규칙 $X \to Y_1 Y_2 \cdots Y_k$을 출력한다.
  5. 만약 X가 3번이나 4번에 해당하지 않는다면 오류를 출력한다.
  6. $X = \$$일 때까지 2번부터 다시 반복한다.

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \rightarrow TE'$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \rightarrow TE'$
$\$E'T$	$id + id * id\ \$$	$T \rightarrow FT'$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id * id\ \$$	$F \to id$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id * id\ \$$	$F \to id$
$\$E'T'id$	$id * id\ \$$	

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id * id\ \$$	$F \to id$
$\$E'T'id$	$id * id\ \$$	
$\$E'T'$	$*id\ \$$	$T' = *FT'$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id * id\ \$$	$F \to id$
$\$E'T'id$	$id * id\ \$$	
$\$E'T'$	$*id\ \$$	$T' = *FT'$
$\$E'T'F*$	$*id\ \$$	

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id * id\ \$$	$F \to id$
$\$E'T'id$	$id * id\ \$$	
$\$E'T'$	$*id\ \$$	$T' = *FT'$
$\$E'T'F*$	$*id\ \$$	
$\$E'T'F$	$id\ \$$	$F \to id$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id * id\ \$$	$F \to id$
$\$E'T'id$	$id * id\ \$$	
$\$E'T'$	$*id\ \$$	$T' = *FT'$
$\$E'T'F*$	$*id\ \$$	
$\$E'T'F$	$id\ \$$	$F \to id$
$\$E'T'id$	$id\ \$$	

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id * id\ \$$	$F \to id$
$\$E'T'id$	$id * id\ \$$	
$\$E'T'$	$*id\ \$$	$T' = *FT'$
$\$E'T'F*$	$*id\ \$$	
$\$E'T'F$	$id\ \$$	$F \to id$
$\$E'T'id$	$id\ \$$	
$\$E'T'$	$\$$	$T' \to \varepsilon$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id * id\ \$$	$F \to id$
$\$E'T'id$	$id * id\ \$$	
$\$E'T'$	$*id\ \$$	$T' = *FT'$
$\$E'T'F*$	$*id\ \$$	
$\$E'T'F$	$id\ \$$	$F \to id$
$\$E'T'id$	$id\ \$$	
$\$E'T'$	$\$$	$T' \to \varepsilon$
$\$E'$	$\$$	$E' \to \varepsilon$

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id * id\ \$$	$F \to id$
$\$E'T'id$	$id * id\ \$$	
$\$E'T'$	$*id\ \$$	$T' = *FT'$
$\$E'T'F*$	$*id\ \$$	
$\$E'T'F$	$id\ \$$	$F \to id$
$\$E'T'id$	$id\ \$$	
$\$E'T'$	$\$$	$T' \to \varepsilon$
$\$E'$	$\$$	$E' \to \varepsilon$
$\$$	$\$$	

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

예측 구문 분석기 예

- 아래 파싱표를 이용하여 문장 $id + id * id$에 대한 구문 분석을 수행해보자.

스택	입력 기호	생성 규칙
$\$E$	$id + id * id\ \$$	$E \to TE'$
$\$E'T$	$id + id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id + id * id\ \$$	$F \to id$
$\$E'T'id$	$id + id * id\ \$$	
$\$E'T'$	$+id * id\ \$$	$T' \to \varepsilon$
$\$E'$	$+id * id\ \$$	$E' \to +TE'$
$\$E'T+$	$+id * id\ \$$	
$\$E'T$	$id * id\ \$$	$T \to FT'$
$\$E'T'F$	$id * id\ \$$	$F \to id$
$\$E'T'id$	$id * id\ \$$	
$\$E'T'$	$*id\ \$$	$T' = *FT'$
$\$E'T'F*$	$*id\ \$$	
$\$E'T'F$	$id\ \$$	$F \to id$
$\$E'T'id$	$id\ \$$	
$\$E'T'$	$\$$	$T' \to \varepsilon$
$\$E'$	$\$$	$E' \to \varepsilon$
$\$$	$\$$	

	id	$+$	$*$	$($	$)$	$\$$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

상향식 구문 분석bottom-up parsing

Definition

$S \overset{*}{\Rightarrow} \alpha Aw \Rightarrow \alpha\beta w$의 유도 과정이 존재할 때 문장 형태 $\alpha\beta w$에서 β를 A로 대체하는 것을 감축reduce이라 하고, β를 문장 형태 $\alpha\beta w$의 핸들handle이라 한다.

- 주어진 문장이 문법에 맞는지 아닌지를 검사하기 위해 입력된 문자열을 읽어가면서 감축에 의해 시작 기호를 찾아가는 방법
- 주어진 문자열이 시작 기호로 감축되면 올바른 문장이라고 판단하여 파스 트리를 생성
- 그렇지 않으면 올바르지 않은 문장으로서 오류 메시지를 출력
- 상향식 파싱은 문장으로부터 오른쪽 유도의 핸들을 반복적으로 찾고 이를 감축하여 오른쪽 유도의 역순을 계산하는 작업이다.

오른쪽 유도에서 핸들 찾기 예

- 아래의 문법을 생각해보자.

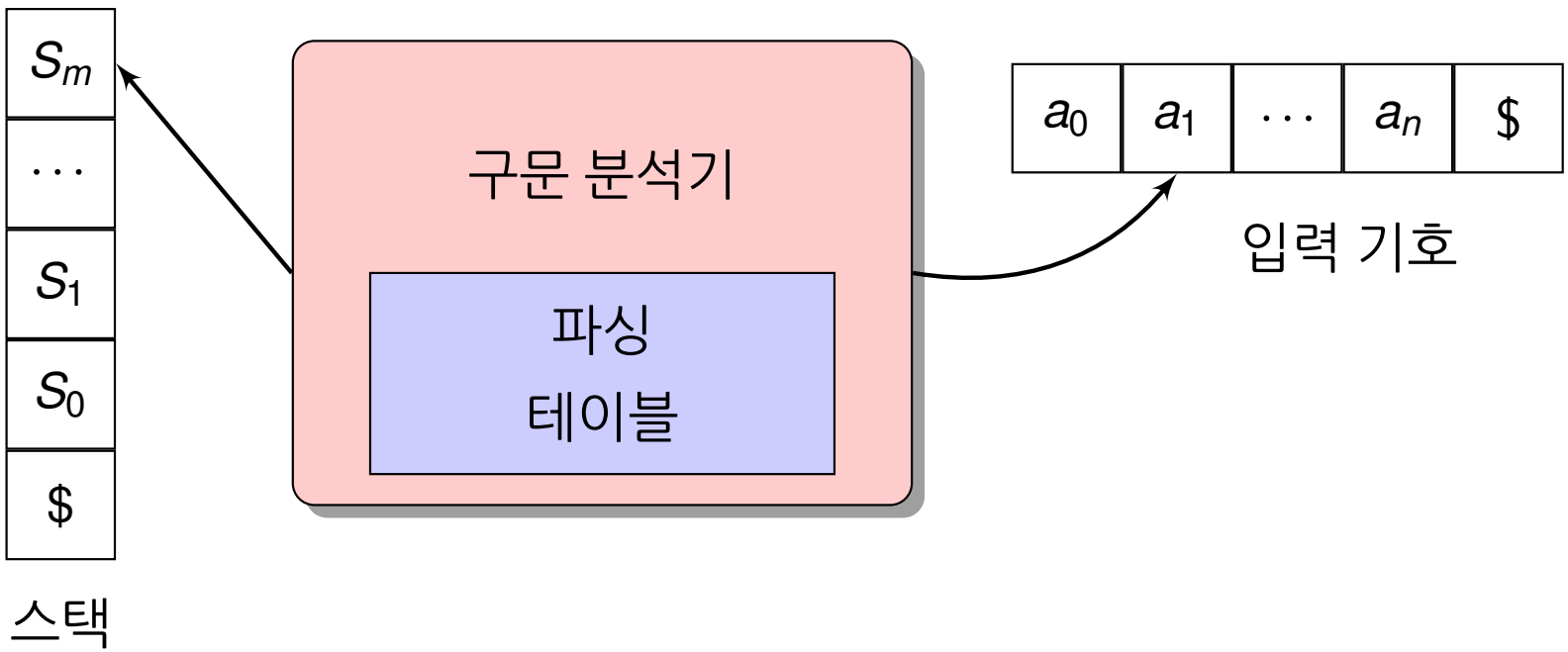
$$E \to E + T \mid T, \;\; T \to T * F \mid F, \;\; F \to (E) \mid id$$

- 위 문법에서 문장 $id + id * id$에 대한 오른쪽 유도 과정을 보이고 핸들을 찾아보자.

$$\begin{aligned}
E &\Rightarrow \underline{E + T} \\
&\Rightarrow E + \underline{T * F} \\
&\Rightarrow E + T * \underline{id} \\
&\Rightarrow E + \underline{F} * id \\
&\Rightarrow E + \underline{id} * id \\
&\Rightarrow \underline{T} + id * id \\
&\Rightarrow \underline{F} + id * id \\
&\Rightarrow \underline{id} + id * id
\end{aligned}$$

이동-감축shift-reduce 구문 분석

- 이동-감축 구문 분석은 스택stack 자료구조를 기반으로 구현한다.
  - 이동 행동은 다음 입력 토큰을 파서의 스택으로 이동한다.
  - 감축 행동은 스택 꼭대기에 위치한 핸들을 생성 규칙의 왼편으로 대체한다.
  - 초기 상태에서는 스택이 비어 있으며 성공적으로 구문 분석이 수행되면 최종 상태에서는 스택에 시작 기호가 남는다.



이동-감축 구문 분석 예

- 아래 문법에서 문장 $id + id * id$에 대한 이동-감축 구문 분석을 해보자.

$E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow id$

단계	스택	입력 기호	구문 분석 행동
1	\$	$id + id * id$ \$	이동 id
2	\$$id$	$+id * id$ \$	감축 $F \rightarrow id$
3	\$$F$	$+id * id$ \$	감축 $T \rightarrow F$
4	\$$T$	$+id * id$ \$	감축 $E \rightarrow T$
5	\$$E$	$+id * id$ \$	이동 $+$
6	\$$E+$	$id * id$ \$	이동 id
7	\$$E + id$	$*id$ \$	감축 $F \rightarrow id$
8	\$$E + F$	$*id$ \$	감축 $T \rightarrow F$
9	\$$E + T$	$*id$ \$	이동 $*$
10	\$$E + T*$	id \$	이동 id
11	\$$E + T * id$	\$	감축 $F \rightarrow id$
12	\$$E + T * F$	\$	감축 $T \rightarrow T * F$
13	\$$E + T$	\$	감축 $E \rightarrow E + T$
14	\$$E$	\$	수락

이동-감축 구문 분석 예 2

- 아래 문법에서 문장 $id + id * id$에 대한 이동-감축 구문 분석을 해보자.

$E \to E + E$

$E \to E * E$

$E \to id$

$E \to (E)$

단계	스택	입력 기호	구문 분석 행동
1	\$	$id + id * id$ \$	이동 id
2	\$$id$	$+id * id$ \$	감축 $E \to id$
3	\$$E$	$+id * id$ \$	이동 $+$
4	\$$E+$	$id * id$ \$	이동 id
5	\$$E + id$	$*id$ \$	감축 $E \to id$
6	\$$E + E$	$*id$ \$	이동 $*$
7	\$$E + E*$	id \$	이동 id
8	\$$E + E * id$	\$	감축 $E \to id$
9	\$$E + E * E$	\$	감축 $E \to E * E$
10	\$$E + E$	\$	감축 $E \to E + E$
11	\$$E$	\$	수락

Any questions?

참고문헌

- A. Aho, J. Ullman, R. Sethi, M. S. Lam, Compilers: Principles, Techniques, and Tools (2nd Edition), Addison Wesley, 2006
- R. Sebesta, Concepts of Programming Languages, 5th Edition, Addison-Wesley, 2001
- K. C. Louden, Compiler Construction: Principles and Practice, Cengage Learning, 1997
- K. C. Louden and K. A. Lambert, Programming languages : Principles and Practice, 3rd Edition, Cengage Learning, 2012
- 박두순, 컴파일러의 이해, 한빛아카데미, 2016
- 김종훈, 김종진, 프로그래밍 언어론 : 쉽게 배우는 언어의 원리와 구조, 한빛미디어, 2013