컴파일러: 4장

국민대학교 소프트웨어학부
강 승 식

제4장 CFG와 구문 분석

- 구문 분석(syntactic analysis)
  - 파싱(parsing) 이라고 함
    - 입력: 스트링(문장)
    - 출력: 문장의 구조를 tree 형태로 출력
  - 문맥 자유 문법(CFG)으로 기술된 생성규칙 적용

- 유도(derivation)
  - 문법의 시작 기호로부터 생성규칙을 적용하여 입력 스트링을 생성하는 과정
    - 생성규칙을 하나씩 적용

a + a * a 를 유도하는 과정

- 생성규칙

  1. E → E + E
  2. E → E * E
  3. E → a

  E ⇒ E + E (1번 생성규칙 적용)
  ⇒ a + E (3번 생성규칙 적용)
  ⇒ a + E * E (2번 생성규칙 적용)
  ⇒ a + a * E (3번 생성규칙 적용)
  ⇒ a + a * a (3번 생성규칙 적용)

- 적용된 생성규칙의 순서: 1 3 2 3 3

좌단 유도와 우단 유도

- 유도 과정에서 생성되는 문장 형태(sentential form)
  - 2개 이상의 논터미널을 포함하는 경우가 많음
  - 이 경우에 어떤 논터미널에 대해 우선적으로 생성규칙을 적용할 것인지…

- 좌단 유도(leftmost derivation)
  - 각 문장 형태에 대해 항상 왼쪽 끝에 있는 논터미널에 대해 생성규칙을 적용
  - 좌파스(left parse)
    - 좌단 유도에서 적용된 생성규칙의 순서
    - 예) 1 3 2 3 3

- 우단 유도(rightmost derivation)
  - 유도 과정에서 항상 가장 오른쪽 끝에 있는 논터미널에 대해 생성규칙을 적용

우단 유도와 우단 역유도

- 우단유도의 예

  $E \Rightarrow E + E$ (1번 생성규칙 적용)
  $\Rightarrow E + E * E$ (2번 생성규칙 적용)
  $\Rightarrow E + E * a$ (3번 생성규칙 적용)
  $\Rightarrow E + a * a$ (3번 생성규칙 적용)
  $\Rightarrow a + a * a$ (3번 생성규칙 적용)

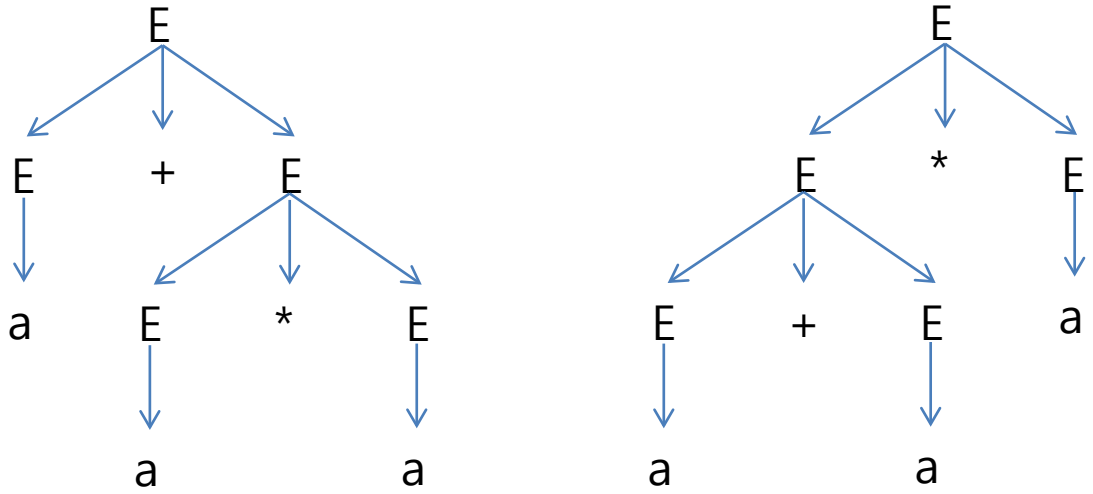
- 우단 역유도
  - 우단유도의 역순

  $E \Leftarrow E + E$ (1번 생성규칙 적용)
  $\Leftarrow E + E * E$ (2번 생성규칙 적용)
  $\Leftarrow E + E * a$ (3번 생성규칙 적용)
  $\Leftarrow E + a * a$ (3번 생성규칙 적용)
  $\Leftarrow a + a * a$ (3번 생성규칙 적용)

- 우파스(right parse): 우단역유도에서 적용된 생성규칙의 순서
  - 예) 3 3 3 2 1

파스 트리와 모호한 문법

- 파스 트리(parse tree)



  - 루트 노트(root node) -- 시작 기호
  - 적용된 생성규칙을 sub-tree 형태로 구성
  - 내부 노드(internal node)는 논터미널 기호(문법 기호)
  - 단말 노드(terminal nocde)는 터미널 기호

문법의 모호성(ambiguity)

- 어떤 입력 스트링에 대해 2개 이상의 다른 모양으로 유도 가능한 문법

E ⇒ E * E	(2번 생성규칙 적용)
⇒ E + E * E	(1번 생성규칙 적용)
⇒ a + E * E	(3번 생성규칙 적용)
⇒ a + a * E	(3번 생성규칙 적용)
⇒ a + a * a	(3번 생성규칙 적용)

- 모호성을 유발하는 생성규칙: $A \rightarrow A\alpha A$ 유형
  - LHS에 기술된 논터미널이 생성규칙의 오른쪽(RHS)에 2회 이상 사용되는 경우

모호하지 않은 문법으로 기술

$E \rightarrow E + F \mid F$

$F \rightarrow F * G \mid G$

$G \rightarrow a$

1. $E \rightarrow E + E$
2. $E \rightarrow E * E$
3. $E \rightarrow a$

좌단 유도:

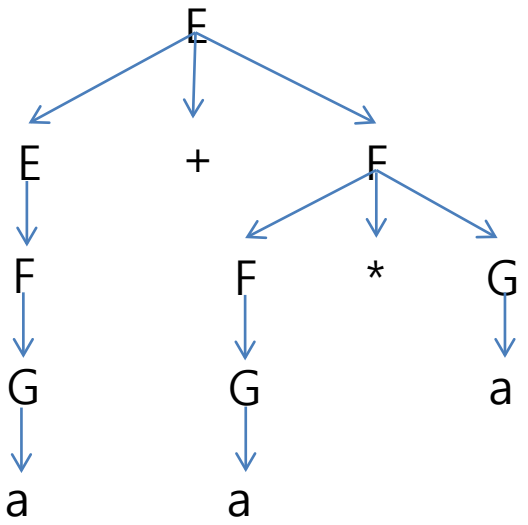
$E \Rightarrow E + F$	(1번 생성규칙 적용)
$\Rightarrow F + F$	(2번 생성규칙 적용)
$\Rightarrow G + F$	(4번 생성규칙 적용)
$\Rightarrow a + F$	(5번 생성규칙 적용)
$\Rightarrow a + F * G$	(3번 생성규칙 적용)
$\Rightarrow a + G * G$	(4번 생성규칙 적용)
$\Rightarrow a + a * G$	(5번 생성규칙 적용)
$\Rightarrow a + a * a$	(5번 생성규칙 적용)

우단 유도:

$E \Rightarrow E + F$	(1번 생성규칙 적용)
$\Rightarrow E + F * G$	(3번 생성규칙 적용)
$\Rightarrow E + F * a$	(5번 생성규칙 적용)
$\Rightarrow E + G * a$	(4번 생성규칙 적용)
$\Rightarrow E + a * a$	(5번 생성규칙 적용)
$\Rightarrow F + a * a$	(2번 생성규칙 적용)
$\Rightarrow G + a * a$	(4번 생성규칙 적용)
$\Rightarrow a + a * a$	(5번 생성규칙 적용)

모호하지 않은 문법

- 좌단 유도와 우단 유도 과정에서 적용된 생성규칙의 순서는 다르더라도 파스 트리 모양은 동일
- 적용된 생성규칙의 순서와 무관하게 파스 트리의 모양이 항상 유일하게 결정되는 문법

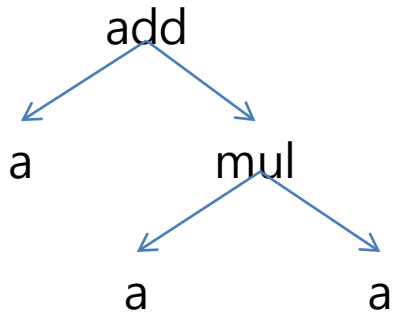


구문 트리(AST: abstract syntax tree)

- 목적 코드로 생성될 필요가 없는 sub-tree는 파스 트리로 생성하지 않아도 된다

- 목적 코드 생성에 영향을 미치지 않는 불필요한 부분을 제거하고 목적 코드 생성에 필요한 요소들만 파스 트리로 구성

  - 예) 단일 생성 규칙
  - 내부노드: sub-tree 연산의 의미
    - 목적코드의 명령어



좌단 유도에 의한 구문 분석 방법

- 시작 기호 → 입력 스트링을 생성
  - 왼쪽 끝부터 오른쪽 방향으로 입력 스트링의 터미널들이 생성
- 스택(stack) 사용
  - 시작 기호로부터 입력 스트링을 생성하는 구문 분석기를 작성하기 위하여 유도 과정에서 생성되는 문장 형태를 저장
- 스택의 초기 상태(initial state)는 시작 기호

- 스택의 top에 있는 논터미널에 대해 생성규칙 적용
  - 해당 논터미널 대신에 적용된 생성규칙의 RHS를 스택에 넣음
  - 논터미널에 적용할 수 있는 생성규칙의 개수가 2개 이상인 경우 각 생성규칙을 적용, 입력 스트링을 생성하는 생성규칙으로 좌단 유도를 수행함
- 스택의 top이 터미널 기호인 경우
  - 입력 스트링과 비교하여 동일한 터미널 기호이면 pop하고, 입력 스트링의 현재 기호를 다음 기호로 이동

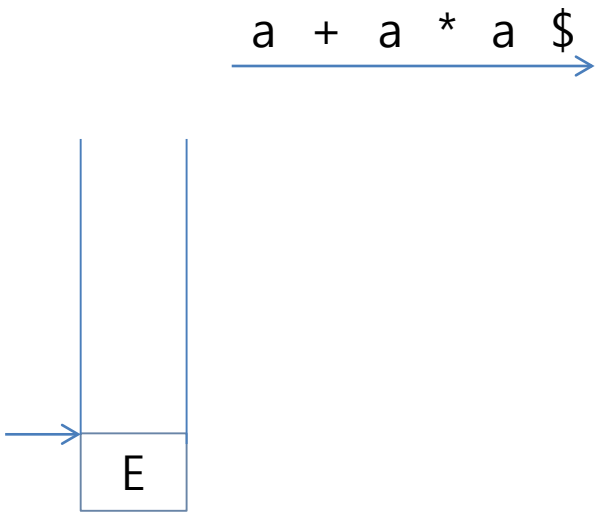
- 입력 스트링이 모두 생성되면 스택은 empty 상태가 되고 구문 분석 종료
- 구문 분석 과정에서 적용된 생성규칙들을 순서대로 sub-tree를 구성하고 파스 트리를 완성하여 그 결과를 출력

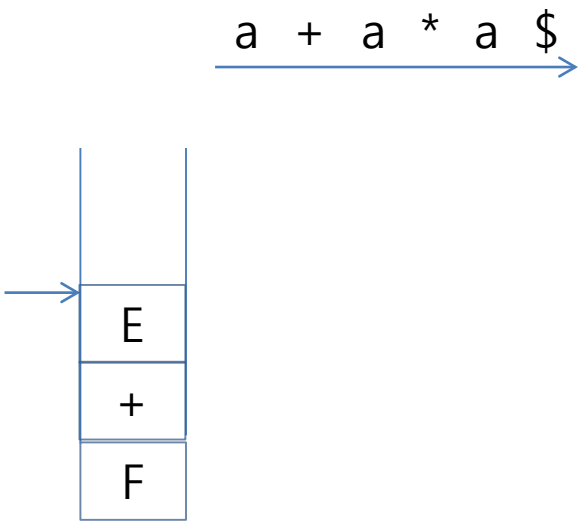
(입력 버퍼, 스택) 구조를 이용한 좌단 유도 과정

(a+a*a, E) ⇒ (a+a*a, E+F)(1번 생성규칙 적용)
⇒ (a+a*a, F+F) (2번 생성규칙 적용)
⇒ (a+a*a, G+F) (4번 생성규칙 적용)
⇒ (a+a*a, a+F) (5번 생성규칙 적용)
⇒ (+a*a, +F) pop a
⇒ (a*a, F) pop +
⇒ (a*a, F*G) (3번 생성규칙 적용)
⇒ (a*a, G*G) (4번 생성규칙 적용)
⇒ (a*a, a*G) (5번 생성규칙 적용)
⇒ (*a, *G) pop a
⇒ (a, G) pop *
⇒ (a, a) (5번 생성규칙 적용)
⇒ (ε, ε) pop a

스택을 이용한 좌단유도 과정

- 연산: pop, expand





우단 역유도에 의한 구문 분석 방법

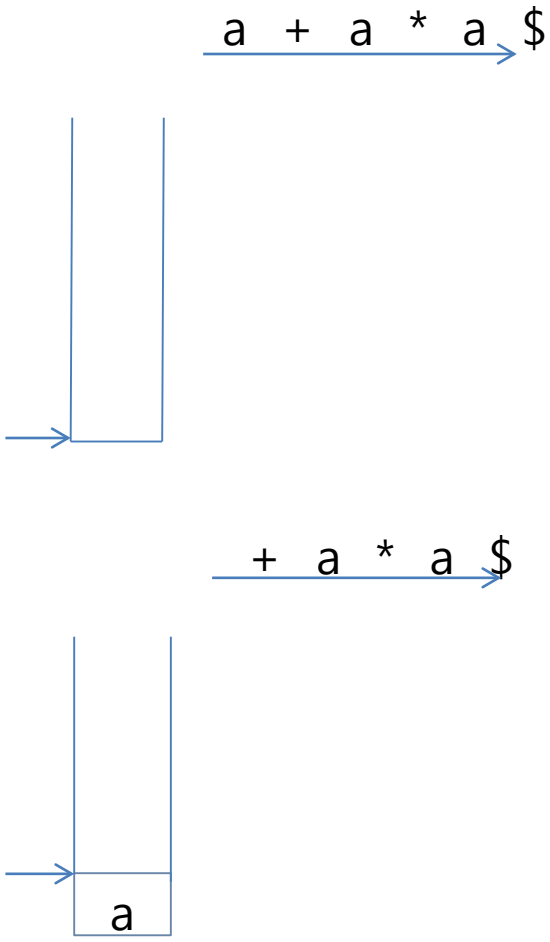
- 입력 스트링 → 시작 기호를 우단 역유도
- 스택의 초기 상태는 empty
  - 입력 버퍼에는 입력 스트링

- 연산
  - Shift: 입력 버퍼에 있는 터미널 기호를 스택으로 이동
  - Reduce: 스택의 top 부분에 있는 substring이 생성규칙의 RHS와 일치하는 것이 발견되면 RHS를 LHS로 대치

- 구문 분석 종료
  - 스택에 시작 기호만 남고, 입력 버퍼가 empty인 상태

(스택, 입력 버퍼) 구조를 이용한 우단 역유도 과정

(ε, a+a*a) ⇒ (a, +a*a)	shift a
⇒ (G, +a*a)	(reduce 5)
⇒ (F, +a*a)	(reduce 4)
⇒ (E, +a*a)	(reduce 2)
⇒ (E+, a*a)	shift +
⇒ (E+a, *a)	shift a
⇒ (E+G, *a)	(reduce 5)
⇒ (E+F, *a)	(reduce 4)
⇒ (E+F*, a)	shift *
⇒ (E+F*a, ε)	shift a
⇒ (E+F*G, ε)	(reduce 5)
⇒ (E+F, ε)	(reduce 3)
⇒ (E, ε)	(reduce 1)



상향식 파싱과 하향식 파싱

- 하향식 파싱(top-down parsing)
  - 좌단 유도에 의한 구문 분석
  - 생성규칙이 루트 노트인 시작 기호로부터 적용
  - 파스 트리 구성할 때 루트 노드에서 시작하여 단말 노드 방향으로 구성
- 상향식 파싱(bottom-up parsing)
  - 우단 역유도에 의한 구문 분석
  - 생성규칙이 적용될 때 입력 스트링으로부터 시작하여 루트 노드 방향으로 생성규칙 적용
  - 파스 트리 구성할 때 단말 노드로부터 루트 노드 방향으로 구성

구문 분석의 비결정성

- 좌단 유도에 의한 구문 분석의 비결정성
  - 좌단 유도 과정에서 논터미널에 대해 2개 이상의 생성규칙이 적용 가능한 경우 백트래킹(backtracking) 기법으로 탐색
    - Ex) A → α | β | γ
  - 각 단계에서 항상 어떤 생성규칙을 적용하면 입력 스트링을 생성할 수 있는지 미리 알 수 있다면 백트래킹이 없는 결정적 파싱(deterministic parsing)이 가능
    - Ex) A → $a\alpha$ | $b\beta$ | cγ

- 우단 역유도에 의한 구문 분석의 비결정성
  - 우단 역유도 과정의 각 단계마다 입력 기호를 shift할지, 스택의 top 부분에 생성규칙을 적용하여 reduce할지
    - Shift-reduce conflict
  - 스택의 top 부분에 RHS를 적용할 때: reduce-reduce conflict
    - 부분 문자열 크기에 따라 2가지 이상의 생성규칙 적용 가능
      - Handle이 α 또는 $\beta\alpha$: $A \rightarrow \alpha$, $B \rightarrow \beta\alpha$

        $F \rightarrow F * G \mid G$
    - 부분 문자열 α가 일치되는 경우: $A \rightarrow \alpha$, $B \rightarrow \alpha$, $C \rightarrow \alpha$
  - 각 단계에서 shift할지 reduce할지, reduce할 경우에 어떤 생성규칙을 적용하는 것이 파싱에 성공할지 미리 알 수 있다면 백트래킹이 없는 결정적 파싱(deterministic parsing)이 가능