컴파일러: 5장

국민대학교 소프트웨어학부
강 승 식

제5장 Top-down 구문 분석

- LL 파싱
  - Left-to-right scanning & Leftmost derivation
  - Deterministic parsing
  - *좌단 유도* 방식에 따라 *결정적인* 방법으로 파싱

- 좌단유도에서 비결정성 문제

  A → $\alpha \mid \beta \mid \gamma$

- 백트래킹이 없는 결정적 파싱이 가능하려면…

  A → $a\alpha \mid b\beta \mid \mathrm{c}\gamma$

백트래킹 없이 파싱 가능한 문법

- 예1

  $S \rightarrow aA \mid bB$

  $A \rightarrow aBb \mid bBb \mid cBb$

  $B \rightarrow d \mid e \mid f$

- 예2

  $S \rightarrow AbB \mid BbB$

  $A \rightarrow aBb \mid bBb \mid cBb$

  $B \rightarrow d \mid e \mid f$

LL(1) 문법

- 각 생성규칙을 적용할 때 현재 위치에서 생성되어야 할 터미널 1개를 보고 어떤 생성규칙을 적용할지 알 수 있는 결정적 파싱이 가능한 문법

- 예) A → $\alpha \mid \beta$
  - A→α를 적용했을 때 생성되는 첫 번째 터미널 집합: FIRST(α)
  - A→β를 적용했을 때 생성되는 첫 번째 터미널 집합: FIRST(β)
  - LL(1) 조건: 모든 생성규칙들이 결정적 파싱이 가능

$$\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \phi$$

LL(2) 문법

- 예제 문법

  S → aAb | aBb | acc

  A → a | Acc

  B → b

  - S-생성규칙을 적용할 때 현재 위치에서 생성되는 첫 번째 문자는 모두 a
    - 터미널 문자 1개만 보았을 때 결정적 파싱이 불가능
  - 생성할 문자를 2개로 확장하여 aa/ab/ac를 보면 결정적 파싱이 가능함

- 이 문법은 LL(1) 문법이 아니고 LL(2) 문법

LL(k) 문법

- LL(k) 문법은 현재 위치에서 생성해야 할 터미널 k개를 보고서 적용할 생성규칙을 결정적으로 선택할 수 있는 문법

- 각 생성규칙 $A \rightarrow \alpha \mid \beta$ 들이 있을 때 항상
  - $FIRST_k(\alpha) \cap FIRST_k(\beta) = \phi$

FIRST와 FOLLOW

- FIRST(A): 논터미널 A에 대해 생성규칙을 적용했을 때 생성되는 첫 번째 터미널들의 집합
- FIRST(α): sentential form α 로부터 생성되는 첫 번째 터미널들의 집합
- 예

$S \rightarrow AbB \mid BbB$
$A \rightarrow aBb \mid bBb \mid cBb$
$B \rightarrow d \mid e \mid f$

FIRST(A) = FIRST(aBb) $\cup$ FIRST(bBb) $\cup$ FIRST(cBb) = { a, b, c }
FIRST(B) = FIRST(d) $\cup$ FIRST(e) $\cup$ FIRST(f) = { d, e, f }
FIRST(S) = FIRST(AbB) $\cup$ FIRST(BbB) = { a, b, c, d, e, f }

- 예 4) ε-생성규칙에 대한 FIRST
  $A \rightarrow BabA \mid c$
  $B \rightarrow b \mid \varepsilon$

- ε-생성규칙에 대한 FIRST는 ε을 포함

  – $FIRST(B) = FIRST(b) \cup FIRST(\varepsilon) = \{ b, \varepsilon \}$

- Nullable 논터미널
  – 어떤 논터미널이 ε-생성규칙으로 기술되는 경우 이 논터미널은 nullable하다고 한다

  – $FIRST(A) = FIRST(BabA) \cup FIRST(c) = \{ a, b, c \}$

생성규칙 $X \rightarrow Y_1Y_2Y_3 \cdots Y_k$

- $\varepsilon \notin FIRST(Y_1)$ 인 경우
  $FIRST(Y_1Y_2Y_3 \cdots Y_k) = FIRST(Y_1)$

- $\varepsilon \in FIRST(Y_1)$ 인 경우
  $FIRST(Y_1Y_2Y_3 \cdots Y_k) = \{ FIRST(Y_1) - \{\varepsilon\}) \cup FIRST(Y_2Y_3 \cdots Y_k)$

- Ring sum 연산의 정의
  $A \oplus B =$
  A , if $\varepsilon \notin A$
  $(A-\{\varepsilon\}) \cup B$, if $\varepsilon \in A$

- $FIRST(Y_1Y_2Y_3 \cdots Y_k) = FIRST(Y_1) \oplus FIRST(Y_2Y_3 \cdots Y_k)$
  $= FIRST(Y_1) \oplus FIRST(Y_2) \oplus \ldots \oplus FIRST(Y_k)$

FIRST 구하는 연습

- 문법

  $S \rightarrow aBb \mid Bcb \mid \varepsilon$

  $A \rightarrow aAb \mid BBd$

  $B \rightarrow b \mid \varepsilon$

- FIRST(S) =
- FIRST(A) =
- FIRST(B) =

- 예 5) 아래 문법은 결정적 파싱이 가능한가?

  $A \rightarrow BbA \mid c$

  $B \rightarrow b \mid \varepsilon$

- FIRST(A) = FIRST(BbA) ∪ FIRST(c) = { b, c }
- FIRST(B) = FIRST(b) ∪ FIRST(ε) = { b, ε }

- B-생성규칙에 대해 결정적 파싱 검사의 문제점
  - FOLLOW(B) 정의가 필요함!

FOLLOW(B)

- 생성규칙: $A \rightarrow \alpha B \beta$

1) FIRST(β)에 ε이 포함되지 않은 경우
   FOLLOW(B) = FIRST(β)

2) FIRST(β)에 ε이 포함된 경우
   FOLLOW(B) = (FIRST(β)-{ε}) ∪ FOLLOW(A)

3) β=ε 일 때, 즉 A→ ε 인 경우
   FOLLOW(B) = FOLLOW(A)

4) B가 시작기호인 경우, 입력 스트링의 끝 표시 '$' 추가
   FOLLOW(B) = FOLLOW(B) ∪ { $ }

5) 논터미널 B가 생성규칙의 RHS에서 2회 이상 발견되는 경우는 각각 FOLLOW(B)를 구하여 합집합

FOLLOW 구하는 연습

- 문법
  $S \rightarrow aBb \mid Bcb \mid \varepsilon$
  $A \rightarrow aAb \mid BBd$
  $B \rightarrow b \mid \varepsilon$

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

- FOLLOW(S) =
- FOLLOW(A) =
- FOLLOW(B) =

LOOKAHEAD($A \rightarrow \alpha$)

- FIRST와 FOLLOW를 이용

1) $\alpha \neq \varepsilon$ 이고, $\varepsilon \notin$ FIRST(α) 인 경우

   LOOKAHEAD($A \rightarrow \alpha$) = FIRST(α)

2) $\alpha \neq \varepsilon$ 이고, $\varepsilon \in$ FIRST(α) 인 경우

   LOOKAHEAD($A \rightarrow \alpha$) = (FIRST(α) - {ε}) $\cup$ FOLLOW(A)

3) $\alpha = \varepsilon$ 인 경우

   LOOKAHEAD($A \rightarrow \varepsilon$) = FOLLOW(A)

좌순환 규칙의 문제점

- 좌단 유도에서 좌순환(left recursion) 문제
  $A \rightarrow A\alpha \mid \beta$

- 무한 루프: $A \Rightarrow A\alpha \Rightarrow A\alpha\alpha \Rightarrow A\alpha\alpha \Rightarrow A\alpha\alpha\alpha \Rightarrow \ldots$

- 해결 방법: 좌순환 규칙을 우순환 규칙으로…
  - 문법 $A \rightarrow A\alpha \mid \beta$ 이 생성하는 스트링 유형은 $\beta\alpha^*$
  - α^*를 우순환 규칙으로 기술하면
    - $A' \rightarrow \alpha A' \mid \varepsilon$

- $\beta\alpha^*$에 대한 우순환 규칙

  $A \rightarrow \beta A'$

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

Left factoring

- 문법

  $A \rightarrow aB \mid aC$

- Left factoring

  $A \rightarrow aD$

  $D \rightarrow B \mid C$

CNF(Chomsky Normal Form)

- CNF는 모든 생성규칙의 RHS가 논터미널 2개로 구성되거나 혹은 터미널 1개로 구성

$$A \rightarrow BC$$
$$A \rightarrow a$$

- 파스 트리 형식이 이진 트리로 구성

GNF(Greibach Normal Form)

- GNF는 모든 생성규칙의 RHS가 터미널로 시작
- 두 번째 이하는 논터미널들로만 구성

- 즉, 모든 생성규칙들이 첫 번째 기호는 반드시 터미널이어야 하고, 두 번째 이하는 논터미널만 허용되는 형태의 문법

- $A \rightarrow a\alpha$, a는 터미널, α는 논터미널 스트링

- GNF는 정규 문법과 유사함
  - 정규 문법의 첫 번째 기호가 터미널, 두 번째 기호가 논터미널 1개인데 GNF는 논터미널 개수를 1개 이상으로 확장한 것이다.

- <참고> 어떤 문법을 기술할 때 현실적으로 모든 생성규칙들이 GNF 요건을 만족하도록 문법을 기술하기가 어렵다.

Recursive-descent 파서

- 구현 방법
  - Recursive 프로시저 작성 방법으로 구현
  - LL(1) 문법에 대하여 좌단유도 방식으로
  - 각 터미널, 논터미널에 대해 프로시저 작성

- 각 생성규칙 $A \rightarrow \alpha | \beta$ 에 대해
  - $A \rightarrow \alpha$ 와 $A \rightarrow \beta$ 중에서 어떤 생성규칙을 적용할지는 LOOKAHEAD에 의해 결정

- <참고> $L.A.(A \rightarrow \alpha) \cap L.A.(A \rightarrow \beta) = \phi$

터미널, 논터미널 구현

- 각 터미널 a에 대한 프로시져

```
void pa()
{
    if (nextsymbol == 'a')
        nextsymbol = get_nextsymbol();
    else error;
}
```

- 각 논터미널 A에 대한 프로시져

```
void pA()
{
    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();
        otherwise: error;
}
```

예1)
S → aA | bB
A → aBb | bBb | cBb
B → d | e | f

예2)
S → AbB | BbB
A → aBb | bBb | cBb
B → d | e | f

main() 함수

- 시작기호 S에 대한 프로시져 호출
  – 호출 결과로 모든 입력 스트링이 생성되고 입력 버퍼에 입력 스트링의 끝 표시인 '$'만 남아 있으면 파싱 성공

```
void main()
{
    nextsymbol = get_nextsymbol();
    pS();
    if (nextsymbol == '$')
            accept;
    else error;
}
```

- Recursive descent 파서의 스택 자료구조
  – 프로시져 호출 과정에서 실시간 스택(runtime stack) 사용

구현: Recursive-descent 파서

- 아래 문법에 대해 파서 구현

  $A \to Ba \mid c$

  $B \to bAB \mid \varepsilon$

- 구문 분석 테스트

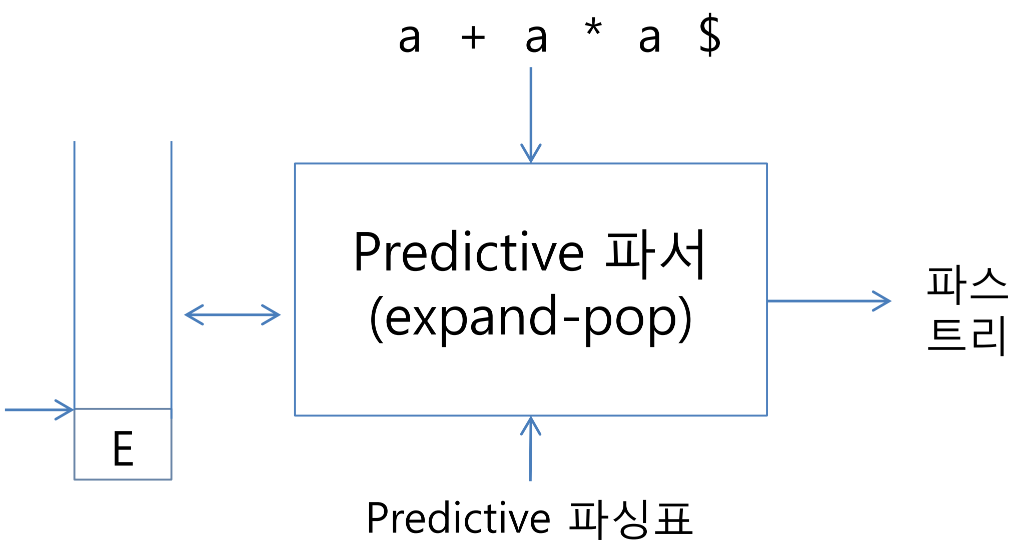
  bcbcbca, babababaa, babcbabca 등

Recursive descent 파서는…

- 구현 방법이 매우 쉽고 간단하다
- 문법을 hard coding
  - 문법을 수정했을 때 프로시져를 직접 수정

Predictive 파서

- Recursive descent 파서의 단점 극복
- 문법을 파싱표에 저장
- 좌단유도 과정을 스택 자료구조로 구현
- 스택의 초기값: 시작기호



Predictive 파싱표 작성

- 문법

  S → bAb | aB | ε
  A → aAb | bBa
  B → b | ε

- 1) FIRST 구하기

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

- 2) FOLLOW 구하기

  FOLLOW(S) = { $ }
  FOLLOW(A) = { b }
  FOLLOW(B) = { a, $ }

	a	b	$
S	2	1	3
A	4	5	
B	7	6	7

LL 조건을 만족하지 않는 문법

- 파싱표 작성할 때 <논터미널, 터미널> 항에 두 개 이상의 생성규칙이 기술되는 문법은 LL 조건을 만족하지 않음

- LL 조건을 만족하지 않는 문법은 그 의미에 따라 강제로 하나의 생성규칙을 선택한다면 결정적인 파서를 구현할 수 있다

Predictive 파싱표 작성 연습

- 문법

  $E \rightarrow E + E \mid E * E \mid a$

- Top-down 파싱의 결정적인 파싱 조건
  - 모호성 제거
    - 연산자 우선순위와 결합 규칙 반영하여 다시 작성
  - left factoring
  - 좌순환 규칙은 우순환 규칙으로 변환