프로그래밍 언어론

모호한 Grammar

컴퓨터공학과
조은선

충남대학교
CHUNGNAM NATIONAL UNIVERSITY

학습목표

프로그래밍 언어의 syntax를 컴퓨터 내부적으로 표현하기 위해 grammar에 대한 이해를 더 심화한다.

프로그래밍언어론 개요

학습내용

- syntax를 명세하는 다양한 방법과 특성을 공부한다.
- syntax를 컴퓨터 내부적으로 표현하는 방법을 배운다.

목 차

- 들어 가기
- 학습하기
  - 모호한 grammar
  - EBNF 표기법
  - Abstract Syntax Tree
- 평가하기
- 정리하기

알고가기

Pretest

다음 C 프로그래밍 예제에서 출력하는 값은?

```
int false = 0; int true = 1;
if (false) if (true) printf("2\n"); else printf("3\n");
printf("%d\n" 1+2*3);
```

Ⅰ 모호한 grammar

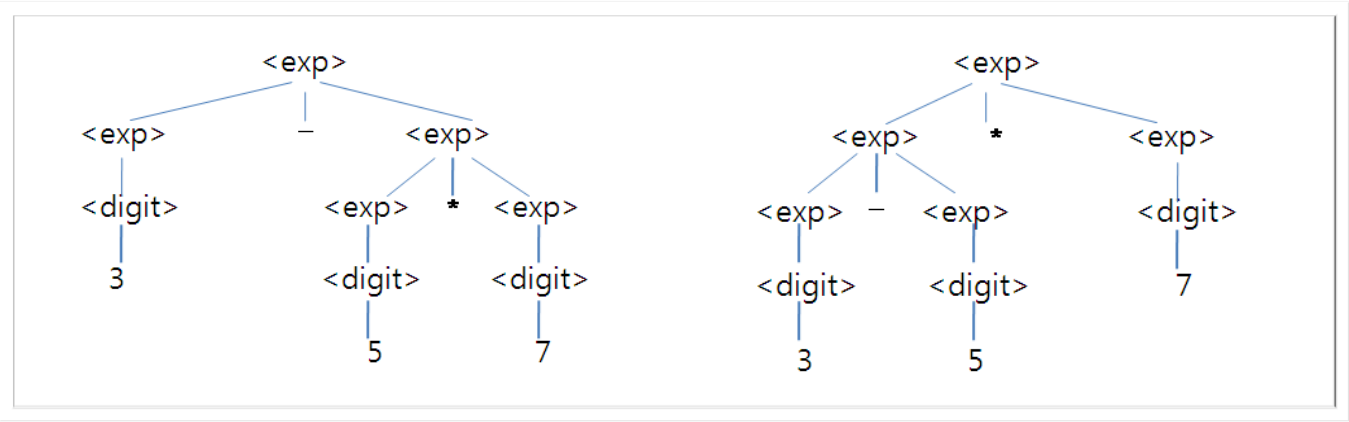
주어진 문장에 대한 두 개 이상의 parsing tree가 가능한 경우 그 grammar는 모호하다고 한다.

간단한 수식 언어의 모호한 grammar

<exp> ::= <exp> * <exp> | <exp> - <exp> | <digit>

<digit> ::= 0 | 1 | 2 | ... | 9

3 - 5 * 7 의 parsing는?



Ⅰ 모호한 grammar의 예

모호한 grammar의 예

<stmt> ::= if <exp> <stmt> else <stmt>

| if <exp> <stmt>

다음 문장의 parsing tree는 ?

if <exp> if <exp> <stmt> else <stmt>

* if <exp> if <exp> <stmt> else <stmt>

* if <exp> if <exp> <stmt> else <stmt>

Ⅰ 모호한 grammar 수정

- 모호함이 발생하지 않도록 원래 의미를 참고하여 grammar를 수정한다.

- <exp> ::= (<exp> * <exp>) | (<exp> - <exp>) | <digit>

  * ((3 - 5) * 7)

  * (3 - (5 * 7))

- <stmt> ::= if <exp> <stmt> else <stmt> endif

  | if <exp> <stmt> endif

  * if <exp> if <exp> <stmt> endif else <stmt> endif

  * if <exp> if <exp> <stmt> else <stmt> endif endif

Ⅰ 모호한 grammar 해결하는 방법

- grammar 외에 우선순위 결합성등 강제하기

- **우선순위**

  예) 3 - 5 * 7에서 *가 높은 우선순위를 갖도록 하면 3 - (5 * 7)의 parse tree를 생성

- **결합성**

  예1) 3 - 5 - 7에서 -를 left associative 하게 하면 (3 - 5) - 7의 parse tree를 생성

  예2) else 부분은 가장 최근에 나타난 if와 짝을 이룬다.

  if <exp> if <exp> <stmt> else <stmt>

  → if <exp> **if** <exp> <stmt> **else** <stmt>

EBNF

EBNF(Extended BNF)

* BNF 표기법에 다양한 매크로를 추가하여 편리하게 사용할 수 있도록 함

* 반복 { }

* 옵션 [ ]

* 다중 선택 ( )

반복 { } : 0번 또는 그 이상의 반복

<number> ::= <number> <digit> | <digit>

<number> ::= <digit> {<digit>}

<exp> ::= <exp> + <term> | <term>

<exp> ::= <term> {+ <term>}

EBNF-옵션과 선택 표현

옵션 [ ]: 한 번 나오거나 나오지 않음을 의미

<if> ::= if <exp> <stmt>

| if <exp> <stmt> else <stmt>

<if> ::= if <exp> <stmt> [else <stmt>]

선택 ( ): 여러 개에서 하나를 선택함

<exp> ::= <exp> + <exp>

| <exp> - <exp>

| <exp> * <exp>

<exp> ::= <exp> (+|-|*) <exp>

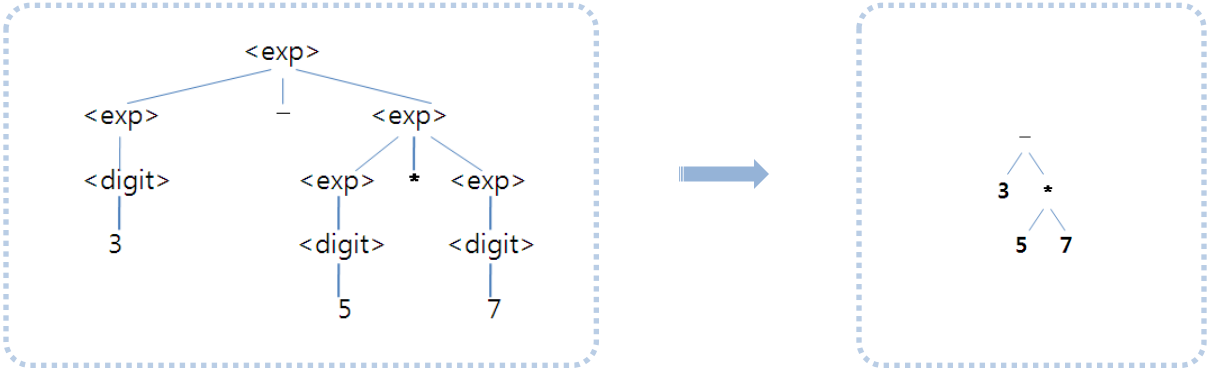
Abstract Syntax Tree

Abstract Syntax Tree(추상구문트리)

- Parsing tree는 문장의 derivation과정이 모두 들어있어서 추후의 다른 작업을 하기에 복잡하다.
- Abstract Syntax Tree (AST)는 parsing tree를 non-terminal symbol들을 제거하고 terminal과 의미 있는 node로만 이루어진 표현으로 간략하게 바꾼것.
  - 문장 내부 구조를 표현하면서 다루기 좋은 자료구조임

    비교) 프로그램을 컴퓨터 내부에 표현하는 다른 방법 들

    예) 문자열, token열, parsing tree..



* AST에서 operator들은 terminal이 root나 중간 node 역할을 하고

  operand는 operator의 children (subtree) 를 구성한다.

평가하기

마지막으로 내가 얼마나 이해했는지를 한번 확인해 볼까요?

총 3문제가 있습니다.

START

평가하기 1

1. 다음 중 모호한 grammar인 것은?

① S -> aSb | ε

② S -> SS | id | ε

③ S -> SaSb | c

④ S -> aS | bS | c

확인

평가하기 2

2. 다음 EBNF grammar 에 대한 설명이 잘못된 것은?

```
<exp> ::= <term> {(+|−) <term>}
<term> ::= <factor> {(*|/) <factor>}
<factor> ::= num
```

① <term>이 나타내는 집합은 num * num을 포함한다.

②<exp>이 나타내는 집합은 <term>이 나타내는 집합을 포함한다.

③ <term>으로부터 유도되는 문장은 반드시 * 또는 /를 포함한다.

④ 문장 num * num + num을 <exp>로부터 유도할 수 있다.

확인

평가하기 3

3. Abstract Syntax Tree (AST) 의 설명이 잘못된 것은?

① AST 는 parsing tree 보다 간단하여 derivation 과정이 드러나지 않는다.

② AST 는 parsing tree에서 terminal symbol들을 제거한 것이다.

③ AST 에는 operator가 root 나 중간 node가 역할을 한다.

④ AST는 문장 내부 구조를 컴퓨터에서 표현하기 좋은 자료 구조이다.

확인

정리하기

모호한 grammar

주어진 문장에 대하여 두 개 이상의 서로 다른 parsing를 생성할 수 있는 grammar는 모호하다고 한다.

EBNF

BNF 표기법에 다양한 매크로를 추가하여 좀 더 편리한 표기법을 제시한다.

Abstract Syntax Tree (AST)

Parsing tree로부터 non-terminal을 제거하여 간략화한 AST는 프로그램을 컴퓨터 내부에서 다루기 좋은 자료구조이다.