2. 구문정의 및 구문중심 컴파일

충북대학교

이 재 성

LaKE LAB

학습내용

- 프로그램언어의 구문정의 방법
- 파싱 방법 및 관련된 문제
- 구문중심 컴파일 기법

구문정의

컴퓨터 언어의 정의

- 언어 구문
  - 일반적으로 문맥자유문법(CFG: context free grammar)이나 BNF(Backus-Naur Form)으로 표현
- 언어 의미
  - 표현의 어려움: 설명 및 예제 사용

문맥 자유 문법의 구성요소

- CFG = $\langle \Sigma, N, S, P \rangle$
- Σ: 단말 기호들(토큰의 집합)
- N: 비단말 의 집합
- P: 생성규칙들
- S: 출발기호(비단말의 특수한 경우)

문법 예

- CFG = < Σ, N, S, P>
  - 단말기호 Σ
    - +, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - 비단말기호 N
    - list, digit
  - 시작기호 S
    - list
  - 생성규칙 P
    - list -> list + digit
    - list -> list – digit
    - list -> digit
    - digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

생성규칙

생성규칙(production rule)

- 왼쪽 문법 기호는 오른쪽 문자열을 생성한다.
- 예: stmt -> if (expr) stmt else stmt
  - if (표현식) 문장 else 문장

생성규칙

- 예: 한자리수의 숫자들의 더하기 빼기 연산
- list -> list + digit
- list -> list – digit
- list -> digit

list -> list + digit | list – digit | digit

- digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

언어

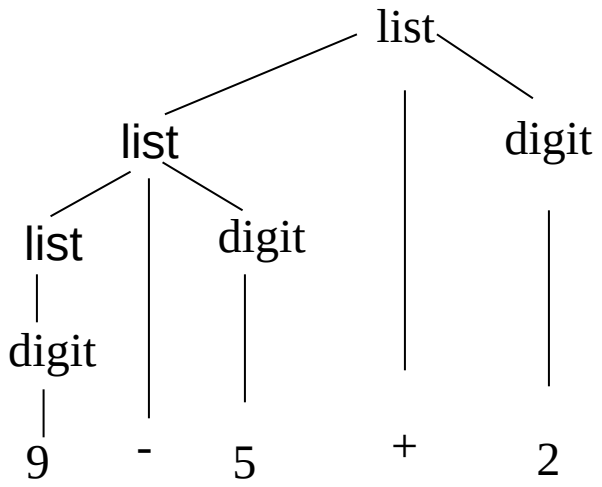
언어

- 생성규칙에 따라 만들어진 토큰 열
- 언어 예

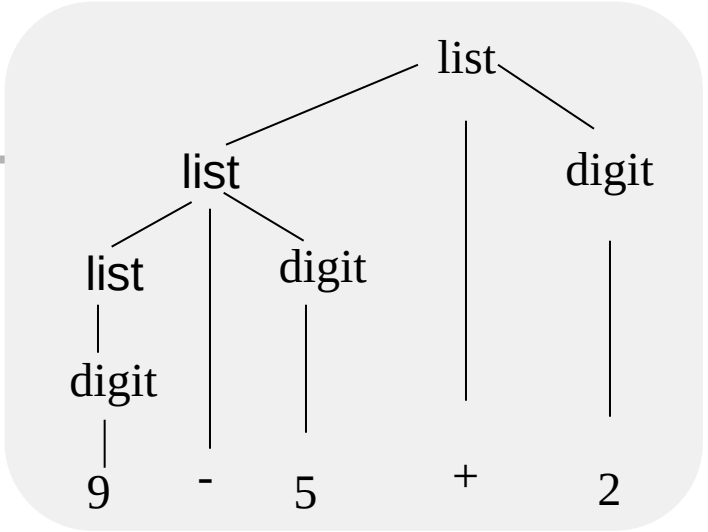
  9-5+2, 3-1

list -> list + digit | list – digit | digit

digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9



파스트리



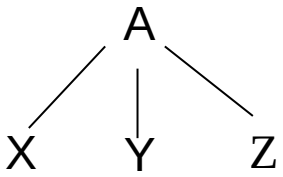
파스트리의 특징

- 트리의 루트는 출발 기호를 이름으로 갖는다.
- 잎(leaf)노드는 토큰이나 ε를 갖는다.
- 중간노드들은 비단말 이름을 갖는다.
- 비단말 노드 A에 X_1, X_2,...X_n이 왼쪽에서 오른쪽으로 붙어있는 자식노드라면 생성규칙 A ->X_1, X_2,...X_n을 나타낸 것이다.

표시

- 생성규칙의 왼쪽에 있는 비단말을 하나의 노드로 나타내고 오른쪽의 비단말 및 단말을 그 노드의 자식으로 표현
- 예

  A -> XYZ



파싱

파싱

- 주어진 토큰열(언어)에 대해 적절한 파스트리를 찾는 과정

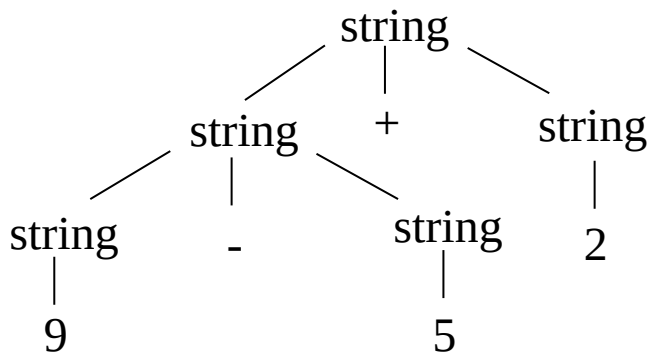
생성(또는 유도)

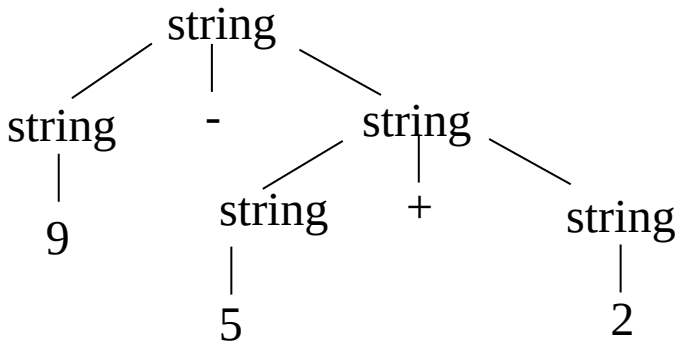
- 파스트리에서 잎노드는 루트노드에 있는 비단말에서 곧바로 생성 혹은 유도된 문자열
- 유도된 문자열은 왼쪽에서 오른쪽으로 읽음

문법 모호성

모호성(Ambiguity)

- 토큰열에 대해 파스트리가 2개 이상 나오는 경우
- 모호성 예:
- string -> string + string | string – string | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9





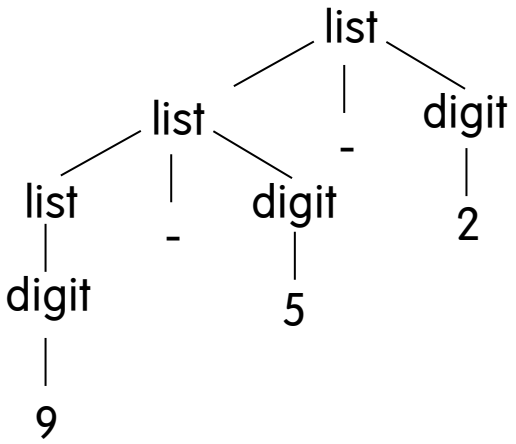
연산자의 연관성(associativity, 결합)

연산자 우선순위

- 곱셈, 나눗셈이 덧셈, 뺄셈보다 우선

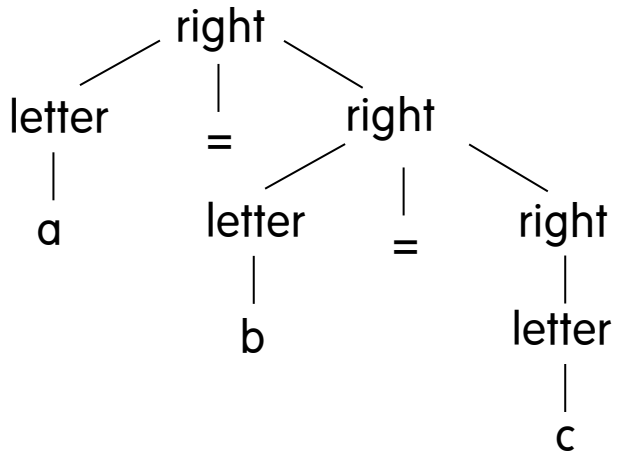
연산자 연관성

- 같은 연산의 묵시적 수행 순서
  - 왼쪽연관: 덧셈, 뺄셈, 곱셈, 나눗셈
    - 예: 9-5-2 -> (9-5)-2



  - 오른쪽연관: 지수, C언어의 =
    - 예: a=b=c -> a=(b=c)

오른쪽 연관 문법 예
right-> letter = right | letter
letter -> a | b | ... | z



우선순위를 고려한 수식 구문

우선순위

- 왼쪽 연관성 + -
- 왼쪽 연관성 * /

기본 단위

- factor -> digit | (expr)

곱셈/나눗셈

- term -> term * factor
  - | term / factor
  - | factor

덧셈/뺄셈

- expr -> expr + term
  - | expr – term
  - | term

구문 중심 변환

구문중심변환(syntax-directed translation)

- 컴파일러 전반부(분석부분)을 주로 사용하여 변환하는 기술

구문 중심 정의법

- 문맥자유문법을 이용하여 컴파일러가 입력문자열에 대해 구문적 분석
- 문법기호와 속성 연결
- 생성규칙 기호와 속성 처리를 위한 의미 규칙 연결

구문 중심 컴파일

애노테이티드 파스트리

- 각 노드의 속성들의 값이 쓰여 있는 파스트리
- 문법기호 X에 대한 속성 a를 X.a로 표시

속성

- 생성에 관련된 타입, 문자열, 할당된 메모리 등

합성속성 (Synthesized Attributes)

- 노드의 속성값이 그 자식노드들의 속성값으로 계산된 것

컴파일러 예제

예제 컴파일러

- 1 패스 컴파일러
- 중위표기를 후위표기로 변환

1 패스

- 입력파일을 처음부터 끝까지 1번만 읽고 필요한 출력파일을 생성
- 어휘분석, 구문분석, 의미분석, 중간코드 생성을 하나로 통합

후의표기의 귀납적 정의

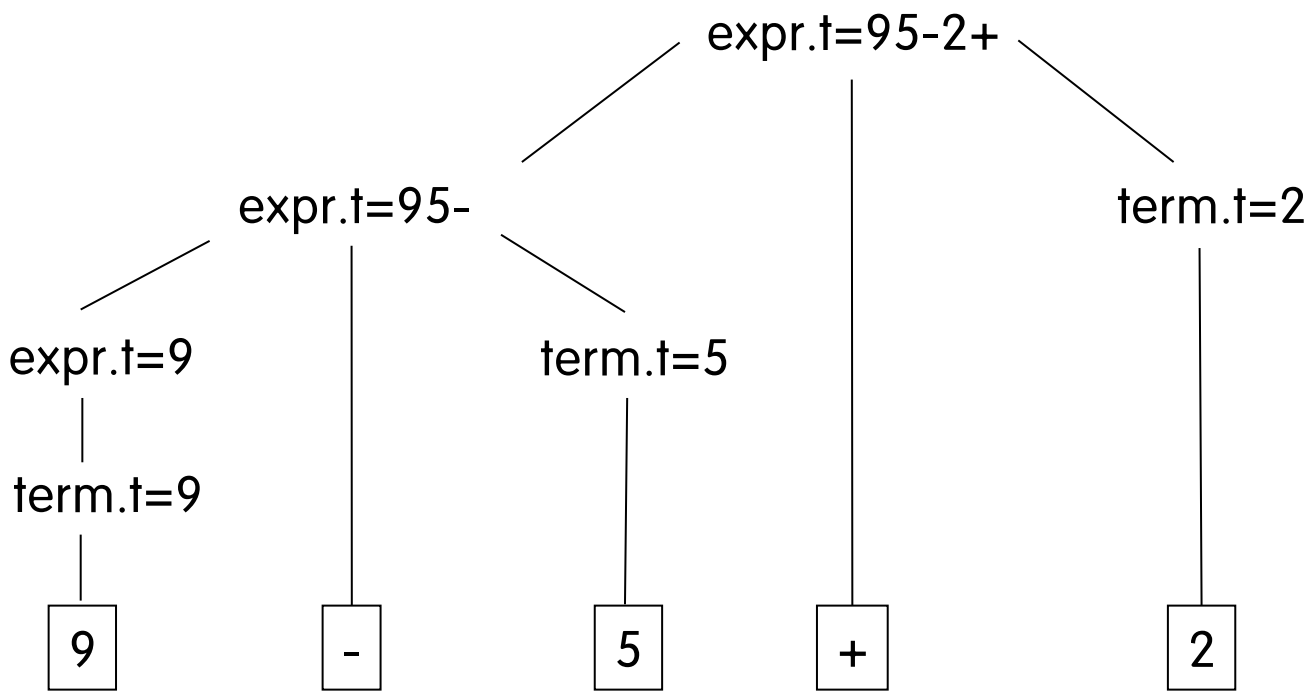
- E가 변수이거나 상수이면 후위표기는 E
- "E1 연산자 E2" 의 후위 표기는 "E1' E2' 연산자"
  (여기에서 E1' E2' 는 E1, E2의 후위표기)
- E가 "(E1)" 형태의 수식이면 후위 표기는 E1 그대로

- 중위식을 후위식으로 변환하는 구문중심정의

생성규칙	의미규칙
expr -> $expr_1$ + term	expr.t := $expr_1$.t ll term.t ll '+'
expr -> $expr_1$ – term	expr .t := $expr_1$.t ll term.t ll ' –'
expr -> term	expr.t := term.t
term -> 0	term.t := '0'
term -> 1	term.t := '1'
...	...
term -> 9	term.t := '9'

ll는 문자열 연결 연산자

- 애노테이티드 파스 트리



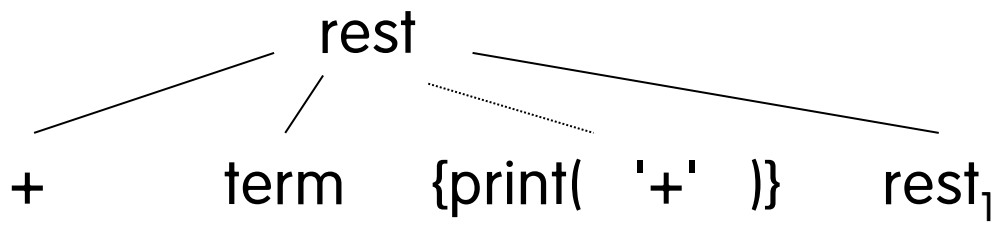
번역계획

정의

- 생성규칙의 오른쪽에 의미동작(semantic actions)이라는 프로그램 일부를 넣은 것
- 문맥 자유문법의 일종

예

- term 과 $rest_1$의 트리 운행 중간에 의미동작 실행
- rest -> + term { print ('+')} $rest_1$



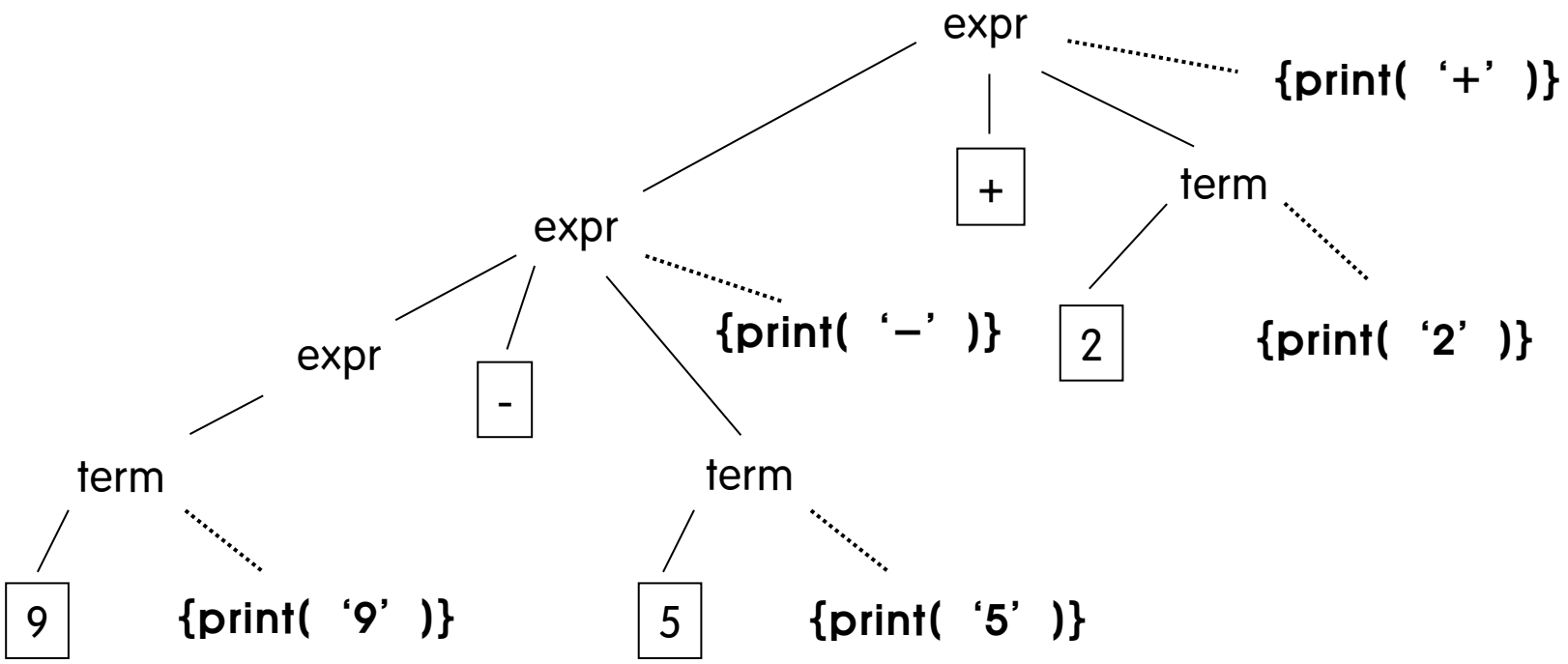
번역 결과의 출력

간단한 구문 중심 정의

- 오른쪽 비단말의 번역결과를 원래 순서대로 연결하면서 약간의 문자열을 추가하여(또는 추가없이) 생성규칙의 왼쪽에 있는 비단말을 정의

expr -> $expr_1$ + term	{ print('+') }
expr -> $expr_1$ – term	{ print('-') }
expr -> term	
term -> 0	{ print('0') }
term -> 1	{ print('1') }
...	
term -> 9	{ print('9') }

- 번역계획 파스 트리 (깊이 우선 탐색)



연습

- 후위표기를 중위표기로 바꾸는 번역계획 작성
  - +, -, 1자리숫자 연산
- 번역계획 파스트리 작성
  - 예: 234*+

참고 문헌

- [1] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, "Compilers – Principles, Techniques, and Tools," Bell Telephone Laboratories, Incorporated, 1986.
- [2] 오세만, "컴파일러 입문", 정익사, 2004.