Syntax Analysis

Files & media	Syntax.pdf
Name	Week 12
Review	☐

Parse Tree

if

then

else

program → blocks → statements

Context free Grammar

tuples

1. non-terminal symbols
2. terminal symbols
3. start nonterminals
4. productions (P in forms of A→ B, and A in N, B - (Nonterminal U terminal)*

Regular Expression

L = {a^n b^n | n > 0} not a regular set but context-free

Stack

Pushdown Automata(use stack to check number of a and b)

recursive language > context-sensitive language > context-free lang>regular set

Derivation: from grammar to derivation

Leftmost deviation

remove left recursion, left factoring, ambiguity

Rightmost Deviation

remove ambiguity

context free grammar, parse tree

Bottom-Up Parsing(using stack)

1. shift: from terminal to stack
2. reduce: from non-terminal to terminal
3. accept
4. error

Question: E→ E - E | E * E | id

input string: id-id * id

priority: id > * > -

Stack	Input buffer	Parser Action
$	id-id*id$	shift
$id	-id*id$	reduce id in stack to E
$E	-id*id$	shift
$E-	id*id$	shift
$E-id	*id$	reduce id in stack to E
$E-E	*id$	shift
$E-E*	id$	shift
$E-E*id	$	reduce E → id
$E-E*E	$	reduce E → E*E
$E-E	$	reduce E → E - E
$E	$	accept

Question: S → 0S0 | 1S1 | 2

input string: 10201

Stack	Input buffer	Parsing Action
$	10201$	shift
$1	0201$	shift
$10	201$	shift
$102	01$	reduce S→ 2
$10S	01$	shift
$10S0	1$	reduce S → 0S0
$1S	1$	shift
$1S1	$	reduce S→1S1
$S	$	accept

Top-Down Parsing

recursive production

same non-terminal at both left and right hand side of production

Recursive grammar

right recursion better(S→aS) for compiler

reduce left recursion

A → Aa | b

⇒ A → bA′ ; A′ → aA′ |(null)

Ambiguous Grammar

more than one parse tree

(both left and right recursive)

Non-deterministic Grammar

common prefix

convert non-deterministic to deterministic:

(**left-factoring** grammar)

Question: S → aSb | abS | ab

⇒ S → aA′; A′ → Sb | bS | b(null)

⇒ S → aA′; A′ → Sb | bA′′; A′′ → S|(null)

Question: S → ab | abc | abcd | b

⇒ S → abA′ | b; A′→ (null) | cA′′; A′′ → (null) | d

First Rule

first terminal small character

bottom to top

First Rules

- **Rule-01:**
- For a production rule $X \rightarrow \in$, First(X) = { $\in$ }
- **Rule-02:**
- For any terminal symbol ‘a’, First(a) = { a }
- **Rule-03:**
- For a production rule $X \rightarrow Y_1Y_2Y_3$,
- **Calculating First(X)**
- If $\in \notin$ First(Y_1), then First(X) = First(Y_1)
- If $\in \in$ First(Y_1), then First(X) = { First(Y_1) – $\in$ } ∪ First(Y_2Y_3)
- **Calculating First(Y_2Y_3)**
- If $\in \notin$ First(Y_2), then First(Y_2Y_3) = First(Y_2)
- If $\in \in$ First(Y_2), then First(Y_2Y_3) = { First(Y_2) – $\in$ } ∪ First(Y_3)
- Similarly, we can make expansion for any production rule $X \rightarrow Y_1Y_2Y_3.....Y_n$.

Follow Rule

Understanding Follow Function

Follow(A) is the set of all terminals that may follow to the right of (A) in any form of sentential Grammar.
Rules:
1) if A is the start symbol then Follow(A) = {$}

2) if $A \rightarrow \alpha A \beta$, $\beta \rightarrow ! \in$
   Follow(A) = First(β)

3) if $S \rightarrow \alpha A$
   Follow(A) = Follow(S)

4) $S \rightarrow \alpha A \beta$, where $\beta \rightarrow \in$
   Follow(A) = First(β) U Follow(S) - $\in$

Steps

1. remove left recursion
2. get first function and follow function
3. Table

LL(1)

Not LL(1): multiple entry in 1 slot

A Grammar which is not LL(1)

$S \rightarrow i\,C\,t\,S\,E \mid a$ FOLLOW(S) = { \$,e }

$E \rightarrow e\,S \mid \varepsilon$ FOLLOW(E) = { \$,e }

$C \rightarrow b$ FOLLOW(C) = { t }

FIRST(iCtSE) = {i}
FIRST(a) = {a}
FIRST(eS) = {e}
FIRST(ε) = {ε}
FIRST(b) = {b}

	a	b	e	i	t	$
S	$S \rightarrow a$			$S \rightarrow iCtSE$		
E			$E \rightarrow e\,S$ $E \rightarrow \varepsilon$			$E \rightarrow \varepsilon$
C		$C \rightarrow b$				

two production rules for M[E,e]

Problem ➔ ambiguity

Fill in (non-terminal*terminal) FIRST() result with different starting function

if (null), check FOLLOW()

⇒ no nultiple entry → no ambiguity → LL(1)

Intermediate code generator

three-address code: address + instruction

address: name - symbol-table entry

instruction: assignment / copy / …conditional jump

Code optimization

1. common subexpression elimination

2. copy propagation
3. dead code elimination
4. constraint folding
5. code motion