Compiler Design

Today's Class Topics

-Properties of LL(1) Grammar
-How to Test LL(1) Grammar
-LL(1) PYQ discussion
-Bottom up parsing

Compiler Design

Properties of LL(1) Grammar:

1) A Left Recursive Grammar is not LL(1)

$G = \{S \rightarrow Sa \mid b\}$ not LL(1)

$S \rightarrow Sa$

$S \rightarrow b$

	a	b	$
S		S→b S→Sa	

Not LL(1)

2) A left factor Grammar is not LL(1)

$G = \{S \rightarrow aS \mid a \mid b\}$

	a	b	$
S	S→aS S→a		

not LL(1)

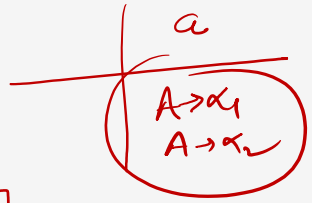
3) An Ambiguous Grammar is not LL(1)

4) Every LL(1) Grammar is UnAmbiguous But every UnAmbiguous Grammar is not LL(1)

Compiler Design

5) If Any Productions are of the form $A \to \alpha_1 | \alpha_2$ Then

$First(\alpha_1) \cap First(\alpha_2) \neq \phi \Rightarrow$ not LL(1)



6) If Any Productions are of the form $A \to \alpha_1 | \alpha_2 | \alpha_3 | \cdots | \alpha_n$ Then any two Productions having Common elements in First set then it is not LL(1)

7) If any Productions are of form $A \to \alpha | \in$, Then

$A \to \alpha$ $A \to \in$

If $First(\alpha) \cap Follow(A) \neq \phi \Rightarrow$ not LL(1)

Ex:

$G = \{ S \to aSa | \in \}$ not LL(1)

$First(aSa) \cap Follow(S) = \{a\} \neq \phi$

	a	$
S	S→aSa S→∈	S→∈

not LL(1)

Compiler Design

(Non left Recursive ∧ left factored ∧ Un Ambiguous) $\not\Rightarrow$ LL(1)

Not LL(1):

1- if any entry of table is multiple defined then G is not LL(1)

2- if G is Ambiguous

3- if G is Left Recursive

4- if G is Left factor

5- And some other cases also

$x \to y \equiv \sim y \to \sim x$

LL(1) → (Non left Recursive) ∧ (left factored) ∧ (Un Ambiguous)

(Left Recursive ∨ left factoring ∨ Ambiguous) → LL'(1)

Ex: ① $S \to AB \mid ab$, $A \to a$, $B \to b$ — not LL(1) — Ambiguous, left factoring

② $S \to Sa \mid b$ — not LL(1) — Because 1) Left Recursive 2) Left factor

③ $S \to a \mid as$ — not LL(1) — because of Contain Common prefix

④ $S \to asa \mid \epsilon$ — 1) it is Un Ambiguous 2) it is non left Recursive 3) it is Left factored

First(asa) ∩ Follow(S) = a ≠ ϕ — not LL(1)

Compiler Design

Testing LL(1) Grammar:

Case 1: $A \to \alpha_1 | \alpha_2 | \alpha_3$

$\in \notin first(\alpha_1)$

$\in \notin First(\alpha_2)$

$\in \notin First(\alpha_3)$

$first(\alpha_1) \cap first(\alpha_2) = \phi$

$first(\alpha_2) \cap first(\alpha_3) = \phi$

$first(\alpha_3) \cap first(\alpha_1) = \phi$

$\Rightarrow$ LL(1)

Case 2: $A \to \alpha_1 | \alpha_2 | \alpha_3$

$\in \notin first(\alpha_1)$

$\in \notin first(\alpha_2)$

$\in \in first(\alpha_3)$

$first(\alpha_1) \cap first(\alpha_2) = \phi$

$first(\alpha_1) \cap [(first(\alpha_3) - \in) \cup follow(A)] = \phi$

$first(\alpha_2) \cap [(first(\alpha_3) - \in) \cup follow(A)] = \phi$

$\Longrightarrow$ LL(1)

Compiler Design

Case 3: $A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3$

$\in \notin first(\alpha_1)$

$\in \in first(\alpha_2)$

$\in \in first(\alpha_3)$

} never LL(1)

$first(\alpha_1) \cap [(first(\alpha_2) - \in) \cup follow(A)] = \phi$

$first(\alpha_1) \cap [(first(\alpha_3) - \in) \cup follow(A)] = \phi$

$[(first(\alpha_2) - \in) \cup follow(A)] \cap [(first(\alpha_3) - \in) \cup follow(A)] \neq \phi$

Compiler Design

LL(1) = ?

$A \to \alpha_1 | \alpha_2$ $\quad$ $A \to \alpha_1 | \alpha_2$

Ex: $G = \{ S \to aS \mid Ab,\ A \to bA \mid b \}$

first(bA) ∩ first(b) = b ≠ ϕ } not LL(1)

Ex: $G = \{ S \to SaAb \mid bA,\ A \to bA \mid \in \}$

not LL(1)

Ex: $G = \{ S \to AaBb \mid \in,\ A \to Bb \mid b,\ B \to aB \mid \in \}$

first(AaBb) ∩ follow(S) = {a, b} ∩ {$} = ϕ ✓

first(Bb) ∩ first(b) = {a, b} ∩ {b} = b ≠ ϕ not LL(1)

first(aB) ∩ follow(B)

Compiler Design

Ex:- $G = \{ S \to aS \mid AbS \mid \in \ , \ A \to aA \mid b \}$

$first(aS) \cap first(AbS)$

$= \{a\} \cap \{a, b\}$

$= a$

$\neq \phi$

] not LL(1)

Compiler Design

$G = \{ S \to eETSS' \mid a,\ S' \to eS \mid \epsilon,\ E \to a \}$

$first(eS) \cap follow(S')$

$e \cap \{e, \$\} \Rightarrow e \neq \$$ } not LL(1)

$follow(S') = follow(S) = \{\$, first(S')\}$

$= \{\$, e, follow(S)\} = \{e, \$\}$

Compiler Design

Compiler Design

Compiler Design

LL(K) :-

① $S \to aA$, $A \to b$ — LL(0)

② $S \to abaA \mid bbA$, $A \to c \mid d$ — LL(1)

③ $S \to abA \mid acA$, $A \to c \mid d$ — LL(2)

④ $S \to abaA \mid abbA$, $A \to c \mid d$ — LL(3)

Ex: $S \to abc \mid def \Rightarrow$ ~~LL(0)~~ $\Rightarrow$ LL(1)

Compiler Design

$S \to Aa$

$A \to bc$

LL(0)

~~LL(2) ⇒ LL(1)~~

$S \to abc \mid ade$ ⇒ LL(2)

LL(0) ⇒ LL(1) ⇒ LL(2) ⇒ LL(3) ⇒ LL(4) ⇒ ... LL(K)

LL(K) ⇒ LL(K+1) = LL'(K+1) ⇒ LL'(K)

~~LL(K+1) ⇒ LL(K)~~

Compiler Design

$LL(0) \Rightarrow LL(1) \Rightarrow LL(2) \Rightarrow LL(3) \Rightarrow \cdots\cdots LL(K)$

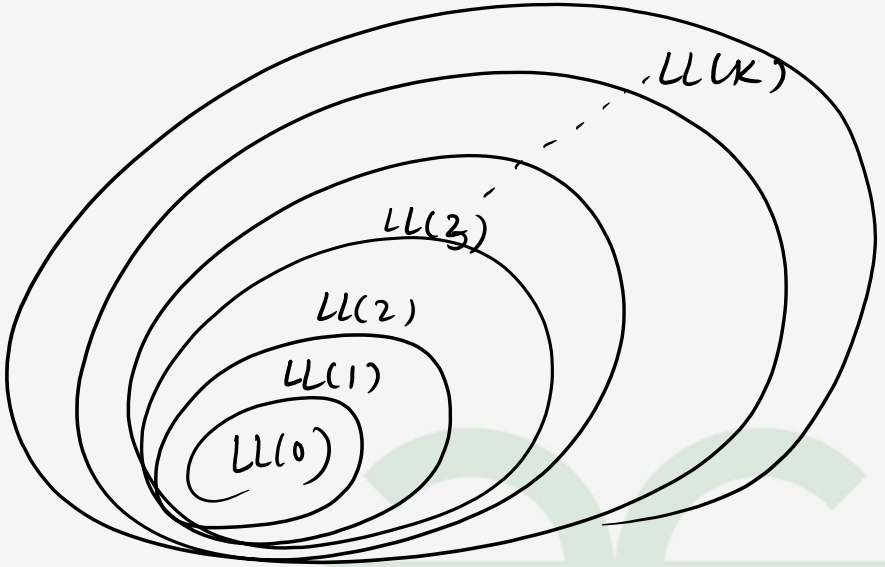
$LL(K) \Rightarrow LL(K+1)$

$LL(K+1) \not\Rightarrow LL(K)$

$LL(K+1) \Rightarrow LL(K)$

Ex: $\left\{ \begin{array}{l} S \rightarrow CC \ \$ \\ C \rightarrow cC \mid d \end{array} \right\} \Rightarrow LL(1) = ?$

$first(cC) \cap first(d)$
$= c \cap d = \phi$



Every $LL(1) \subset LL(2) \subset LL(3) \cdots\cdots \subset LL(K)$

Compiler Design

Question:- Which of the following grammar is LL(3) but not LL(2) and not LL(1)

a) { S → aS | b } LL(1)

b) { S → aaS | ab | b } LL(2)

c) { S → aaaS | aab | ab | b } LL(3) but not LL(2) & not LL(1)

d) None

Compiler Design

Question: The following Grammar $G = \{S \to aA|b, A \to aB|a, B \to bB|a\}$

a) LL(1) but not LL(2)

b) LL(2) but not LL(1) Ans

c) Both LL(1) & LL(2)

d) Neither LL(1) nor LL(2)

LL(2)

$S \to aA|b$

first(aA) ∩ first(b)

a first(A) ∩ first(b)

$a\{a\} \cap \{b\}$

$\Rightarrow aa \cap b = \phi$

LL(2)

$A \to aB|a$

first(aB) ∩ first(a)

$a\{first(B)\} \cap first(a)$

$a\{a, b\} \cap \{a\}$

$\{aa, ab\} \cap \{a\}$

$= \phi \Rightarrow LL(2)$

$B \to bB|a$

first(bB) ∩ first(a)

b first(B) ∩ first(a)

$b\{a, b\} \cap \{a\} \Rightarrow \{ba, bb\} \cap \{a\} = \phi \Rightarrow LL(2)$

Compiler Design

Question: $G = \{ S \to aA/ab,\ A \to aA/b \}$

a) LL(2) but not LL(1)

b) LL(1) but not LL(2)

c) LL(1) & LL(2)

d) None of these.

first(aA) ∩ first(ab)

a first(A) ∩ {ab}

a{a, b} ∩ {ab}

{aa, ab} ∩ {ab}

= ab

≠ ϕ not LL(2)

neither LL(1) nor LL(2) Ans

Compiler Design

Question: LL(1)

$S \rightarrow aBDh$, LL(1) = ?

$B \rightarrow cC$

$C \rightarrow bC \mid \epsilon$ ✓ → $First(bC) \cap follow(C) = \phi$

$D \rightarrow EF$

$E \rightarrow g \mid \epsilon$ ✓ → $First(g) \cap follow(E) = \phi$

$F \rightarrow f \mid \epsilon$ ✓ → $first(f) \cap follow(F) = \phi$

Question:

$S \rightarrow AaAb \mid BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$ ✗

$First(AaAb) \cap First(BbBa)$

$a \cap b = \phi$

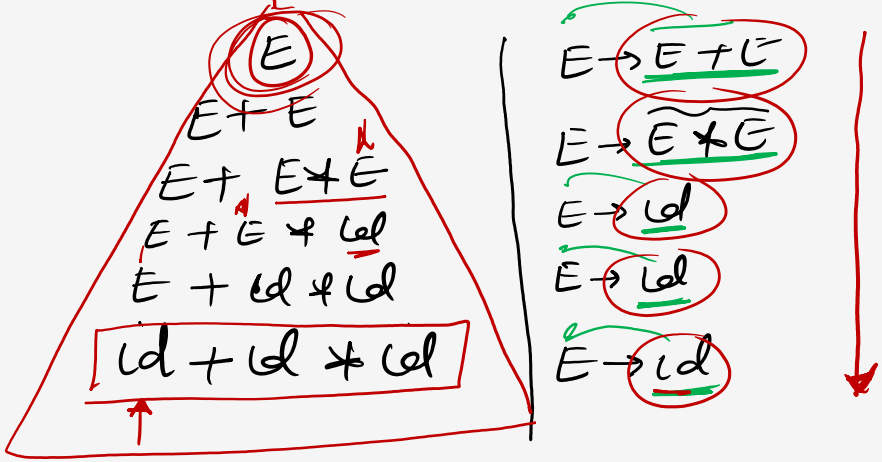
⇒ LL(1)

Compiler Design

LR(K)

Bottom-up parsing: From leaves to root, uses right most derivation in reverse order. (uses reduction process)

Ex:... $G = \{E \to E+E \mid E*E \mid id\}$



$E \to E+E$

$E \to E*E$

$E \to id$

$E \to id$

$E \to id$

Bottom up Parsing

Right most derivation in Reverse order

Compiler Design

Handle Pruning: Reducing the string to start symbol using the handles

Handle: A Handle is a substring of a string that matched with any of the right side of the productions then That handle will be reduced with left side of the production.

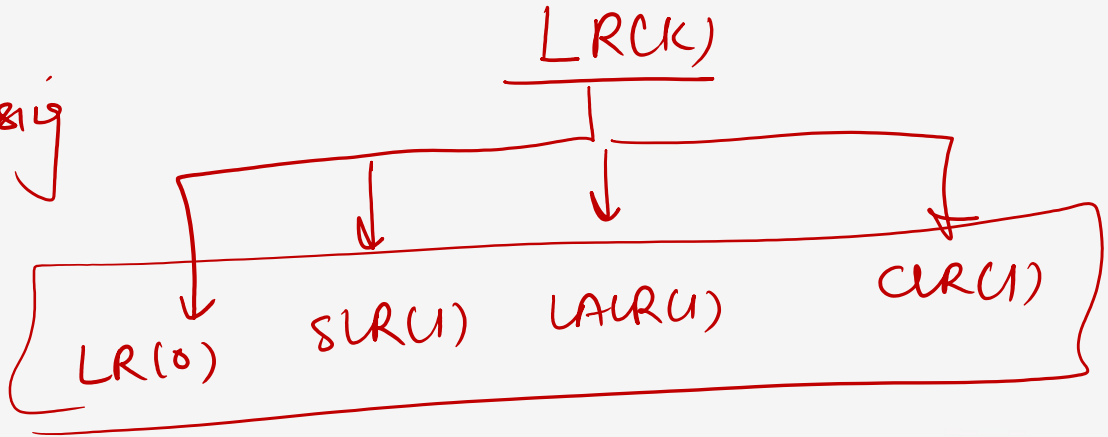
Bottom-up parsing is a process of finding the handles And using them in the reductions to get the start symbol, This entire process of reducing the string to the start symbol is called as handle pruning

Compiler Design

Shift-Reduce Parsing:

or

Bottom up Parsing



Compiler Design

Question:- Consider the following grammar and Parse the input string aab using Shift Reduce Parser

$G_1 = \{ S \to AB, A \to aA \mid a, B \to bB \mid b \}$

Always No. of Shift Action will be Length of String = 3

Reduce Action = 4

Accept = 1

Total Actions = 8

STACK	Input Buffer	Actions
$	aab$	Shift
$a	ab$	Shift
$aa	b$	Reduce A→a (1)
$aA	b$	Reduce A→aA (2)
$A	b$	Shift
$Ab	$	Reduce B→b (3)
$AB	$	Reduce S→AB (4)
$S	$	Accept

Viable prefixes

Handles

Starting Symbol or Rootnode

Compiler Design

Thank You !