Compiler Design

Today's Class Topics

-LL(k) Parser and its necessary condition
-How to construct LL(1) Parsing Table
-How to Test LL(1) Grammar etc.

Compiler Design

H.W

Ex: $G = \{S \to SAB \mid \epsilon,\ A \to AB \mid \epsilon,\ B \to BC \mid Ca,\ C \to cCb \mid a\}$

Soln:

$follow(S) = \{\$, a, c\}$

$follow(A) = \{a, c\}$

$follow(B) = \{a, c, \$\}$

$follow(C) = \{a, b\}$

$first(S) = \{\epsilon, a, c\}$

$first(A) = \{a, c, \epsilon\}$

$first(B) = \{a, c\}$

$first(C) = \{a, c\}$

Compiler Design

H.W

Ex:- $G = \{S \to ACB \mid CbB \mid Ba, A \to db \mid BC, B \to c \mid \epsilon, C \to e \mid \epsilon\}$

Soln:

follow(S) = {$}

follow(A) = {e, c, $}

follow(B) = {$, a, c, e}

follow(C) = {b, c, e, $}

Compiler Design

H.W

Ex:- $G = \{E \to E+T/T,\ T \to T*F/F,\ F \to (E)/id\}$

$Follow(E) = \{+,), \$\}$

$Follow(T) = \{+,), \$, *\}$

$Follow(F) = \{+, *,), \$\}$

Compiler Design

Top down parser:

left to right scanning — LL(K) — leftmost derivation — lookahead component

LL(K) → LL(0), LL(1), LL(2), ... LL(K)

LL(1) Parser:

1. LL(1) Grammar ✓
2. LL(1) Parsing Table
3. LL(1) Grammar Testing
4. LL(1) Parsing

Necessary Condition:

LL(1) Grammar: it is a CFG

1. it is UnAmbiguous
2. it is non Left Recursive
3. it is Left factored (Deterministic CFG)

$x \to y \approx \neg y \to \neg x$ ✓

$x \to y \not\equiv y \to x$ ✗

$(\text{Left Recursive} \vee \text{Left factor} \vee \text{Ambiguous}) \to LL(1)$

$LL'(1) \to (\text{Left Recursive} \vee \text{Left factor} \vee \text{Ambiguous})'$

$LL(1) \to (\text{Non Left Recursive} \wedge \text{Left factored} \wedge \text{UnAmbiguous})$

Compiler Design

CFG to LL(1) Grammar:

Step1: Take any CFG

↓ No Algo. Exist (Undecidable)

Step2: Convert to UnAmbiguous CFG ✓

↓ Elimination of Left Recursion (Algo. Exist)

Step3: Non Left Recusive CFG ✓

↓ Left factoring Algo

Step4: Left factored CFG ✓

↓

Step5: LL(1) Grammer ⇒

Compiler Design

M	x	y
A	A→α	A→α

Construction of LL(1) Parsing Table:

We consider every Production of the form $A \to \alpha$ and Proceed as follows

i) We write $A \to \alpha$ in $M[A, x]$ where $x \in First(\alpha)$

ii) if $First(\alpha) = \{\epsilon\}$ then Add $A \to \alpha$ in $M[A, y]$ where $y \in follow(A)$

iii) The Unfilled entries will be Considered as Syntax error

iv) if all the entries in the table are single then the Grammar is LL(1). Otherwise not LL(1)

$A \to \alpha$, $first(\alpha) = \{\epsilon, \dots\}$, $follow(A) = y$

Ex:- Construct LL(1) Parsing Table for the following Grammar

LL(1) Grammar $G = \{S \to AB, A \to aA|b, B \to bB|\epsilon\}$

$G \Rightarrow \{S \to AB,\ A \to aA|b,\ B \to bB|\epsilon\}$

$A \to \epsilon A$

$A \to b$

$first(S) = \{a, b\}$

$first(A) = \{a, b\}$

$first(B) = \{b, \epsilon\}$

$follow(B) = \{follow(S)\}$

$follow(B) = \{\$\}$

$B \to bB$

$B \to \epsilon$

$\to follow(B) = \{\$\}$

Size of Table $= |V| * (|T|+1)$

$= 3 * (2+1)$

$= 9$

$V = \{S, A, B\}$

$T = \{a, b\}$

M (LL(1)) Parsing Table

	a	b	$
S	(1) S→AB	(2) S→AB	(3) Syntax error
A	(4) A→aA	(5) A→b	(6) Syntax error
B	(7) Syntax error	(8) B→bB	(9) B→ε

Construct LL(1) Passing Table for the following Grammar:

G: $E \to E+T \mid T$
$T \to T*F \mid F$
$F \to (E) \mid id$
not LL(1)

Remove Left Recursion

G': LL(1) Grammar

$E \to TE'$
$E' \to +TE' \mid \epsilon$
$T \to FT'$
$T' \to *FT' \mid \epsilon$
$F \to (E) \mid id$

$L(G') = \{ id+id, id*id, id+id*id, \ldots \infty \}$

First(E) = {(, id}	Follow(E) = {), $}
First(E') = {+, ϵ}	Follow(E') = {), $}
First(T) = {(, id}	Follow(T) = {+,), $}
First(T') = {*, ϵ}	Follow(T') = {+,), $}
First(F) = {(, id}	Follow(F) = {*, +,), $}

LL(1) Passing Table (M)

	id	+	*	(	)	$
E	$E \to TE'$	Syntax error	Syntax error	$E \to TE'$	—	—
E'		$E' \to +TE'$	—	—	$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$	—	—	$T \to FT'$	—	—
T'	—	$T' \to \epsilon$	$T' \to *FT'$	—	$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to id$	—	—	$F \to (E)$	—	—

$T' \to *FT'$ — id+*id

$T' \to \epsilon$ — $id+*id \notin L(G')$

follow(T') = {+,), $}

$F \to (E)$

$F \to id$

Left to Right Scanning → Leftmost derivation

LL(1) Passing Table (M1)

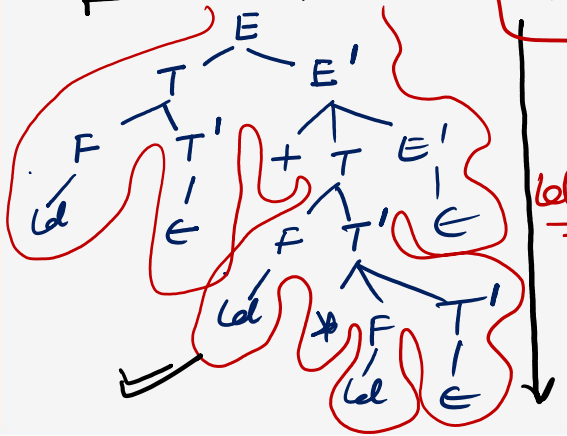
	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ϵ	E'→ϵ
T	T→FT'			T→FT'		
T'		T'→ϵ	T'→*FT'		T'→ϵ	T'→ϵ
F	F→id			F→(E)		

= id + * id → Extra operator Remove

STACK	Input	Output
$E	id + * id $	E→TE' ①
$E'T		T→FT' ②
$E'T'F		F→id ③
$E'T'id	id + * id $	
$E'T'	+ * id $	T'→ϵ ④
$E'		E'→+TE' ⑤
$E'T+	* id $	T→FT' ⑥
$E'T		F→id ⑦
$E'T'F		
$E'T'id		
$E'T'	* id $	T'→*FT' ⑧
$E'T'F*		
$E'T'F	id $	F→id ⑨
$E'T'id		
$E'T'	$	T'→ϵ ⑩
$E'	$	E'→ϵ ⑪
$	$	

Syntax error
* is Extra Removed

id + id * id



Compiler Design

Question GATE-2012

For the grammar below, a partial LL(1) parsing table is also presented along with the grammar. Entries that need to be filled are indicated as E1, E2, and E3. [Tex]\\epsilon[/Tex] is the empty string, $ indicates end of input, and, | separates alternate right hand sides of productions.

S → a A b B | b A a B | ε

A → S

B → S

First(A) = First(S) = First(B) = {a, b, ∈}

Follow(A) = {a, b}

Follow(B) = {a, b, $}

	a	b	$
S	**E1** = ?	**E2** = ?	S → ε
A	A → S	A → S	error
B	B → S	B → S	**E3** = ?

follow(B) = {follow(S)}

follow(S) = {$, follow(A), follow(B)}

follow(B) = {$, a, b, follow(B)}

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

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

(C) FIRST(A) = {a, b, ε} = FIRST(B)
FOLLOW(A) = {a, b}
FOLLOW(B) = ∅

(D) FIRST(A) = {a, b} = FIRST(B)
FOLLOW(A) = {a, b}
FOLLOW(B) = {a, b}

Compiler Design

H.W.

Consider the date same as above question. The appropriate entries for E1, E2, and E3 are

(A) E1: S → aAbB, A → S
E2: S → bAaB, B → S
E3: B → S

(B) E1: S → aAbB, S → ε
E2: S → bAaB, S → ε
E3: S → ε

(C) E1: S → aAbB, S → ε
E2: S → bAaB, S → ε
E3: B → S

(D) E1: A → S, S → ε
E2: B → S, S → ε
E3: B → S

Compiler Design

Consider the following grammar:

```
S → FR
R → S | ϵ
F → id
```

In the predictive parser table, M, of the grammar the entries M[S, id] and M[R, $] respectively.

- [ ] {S → FR} and {R → ε }
- [ ] {S → FR} and { }
- [ ] {S → FR} and {R → *S}
- [ ] {F → id} and {R → ε}

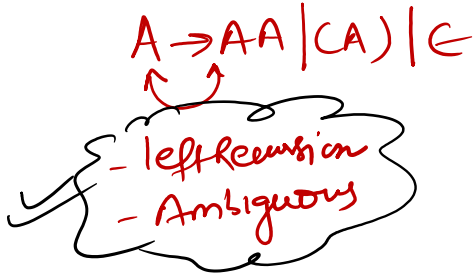
R→S,
R→∈
R→∈

M	id	$
S	S→FR	
R		R→∈

Compiler Design

The grammar A → AA | (A) | ε is not suitable for predictive-parsing because the grammar is

- [x] ambiguous
- [ ] left-recursive
- [ ] right-recursive
- [ ] an operator-grammar



Compiler Design

Which one of the following grammars is free from left recursion?

(A)
$$S \rightarrow AB$$
$$A \rightarrow Aa \mid b$$
$$B \rightarrow c$$

(B)
$$S \rightarrow Ab \mid Bb \mid c$$
$$A \rightarrow Bd \mid \varepsilon$$
$$B \rightarrow e$$

(C)
$$S \rightarrow Aa \mid B$$
$$A \rightarrow Bb \mid Sc \mid \varepsilon$$
$$B \rightarrow d$$

(D)
$$S \rightarrow Aa \mid Bb \mid c$$
$$A \rightarrow Bd \mid \varepsilon$$
$$B \rightarrow Ae \mid \varepsilon$$

Compiler Design

Properties of LL(1) Grammar:

Compiler Design

Compiler Design

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

Ex: $S \to AB \mid ab$, $A \to a$, $B \to b$ | $S \to Sa \mid b$ | $S \to a \mid as$ | $S \to asa \mid \epsilon$

Compiler Design

Testing LL(1) Grammar:

Compiler Design

Compiler Design

Compiler Design

LL(K) :-

$S \to aA$
$A \to b$

|

$S \to abaA \mid bbA$
$A \to c \mid d$

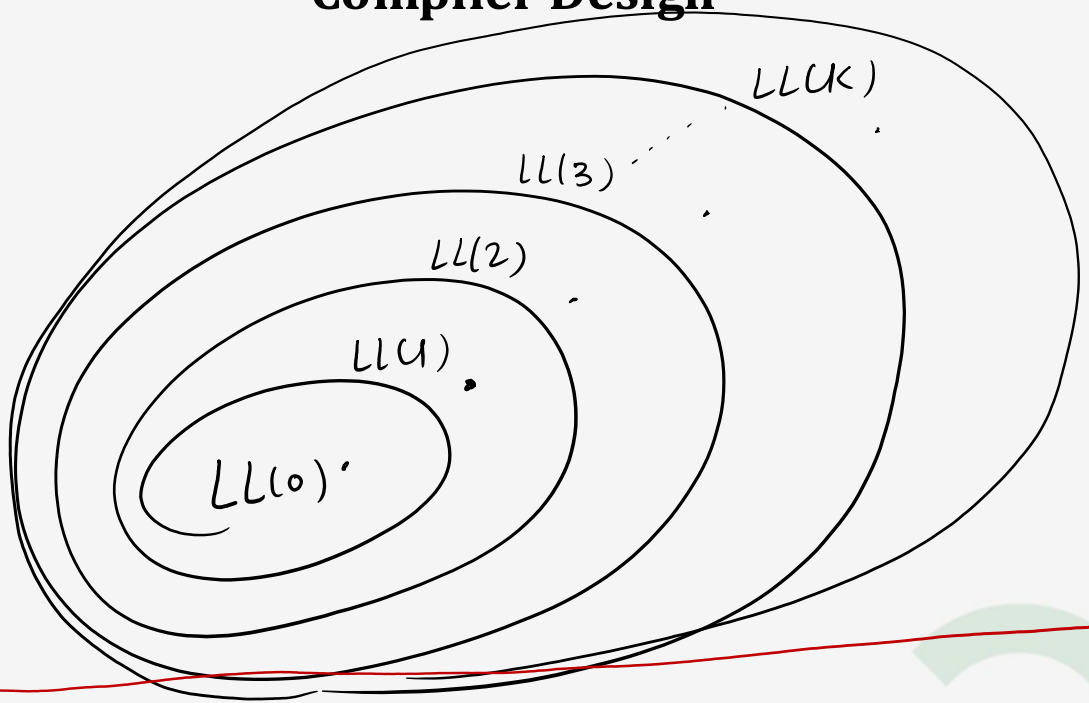
|

$S \to abA \mid aaA$
$A \to c \mid d$

|

$S \to abaA \mid abbA$
$A \to c \mid d$

Compiler Design



Every: $LL(0) \subset LL(1) \subset LL(2) \subset LL(3) \subset LL(4) \cdots \subset LL(K)$

Compiler Design

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

a) $\{S \rightarrow aS|b\}$

b) $\{S \rightarrow aaS|ab|b\}$

c) $\{S \rightarrow aaaS|aab|ab|b\}$

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)

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

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

Compiler Design

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

Compiler Design

Thank You !