classmate
Date 19-Jan-20
Page 1

Compilers Design - Short notes

- **Introduction:**

→ main aim of CD is to convert HLL → LLL (High → Low level lang)

- **Lexical Analyser:**

→ main function of lexical Analyser to converting lexems into token.

* How many token is there — [Printf("%d Hai", &(x+y));] → 8 token.
* ambiguous gramm: To the given gram & str, if we get more than one LMD or RMD or Parse tree then the grammar is called ambiguous grammar.

→ ambiguous problem undecidable.

- ambiguous → unambiguous (grammar should be left recursive) (E) → (E)+ id/id
- grammar: (A) → (A)$B/B
  (B) → (B)#C/C
  (C) → (C)@D/D
  D → d
  } here, $, #, @ are left recursive.
  $ < # < @
- Left recursion:

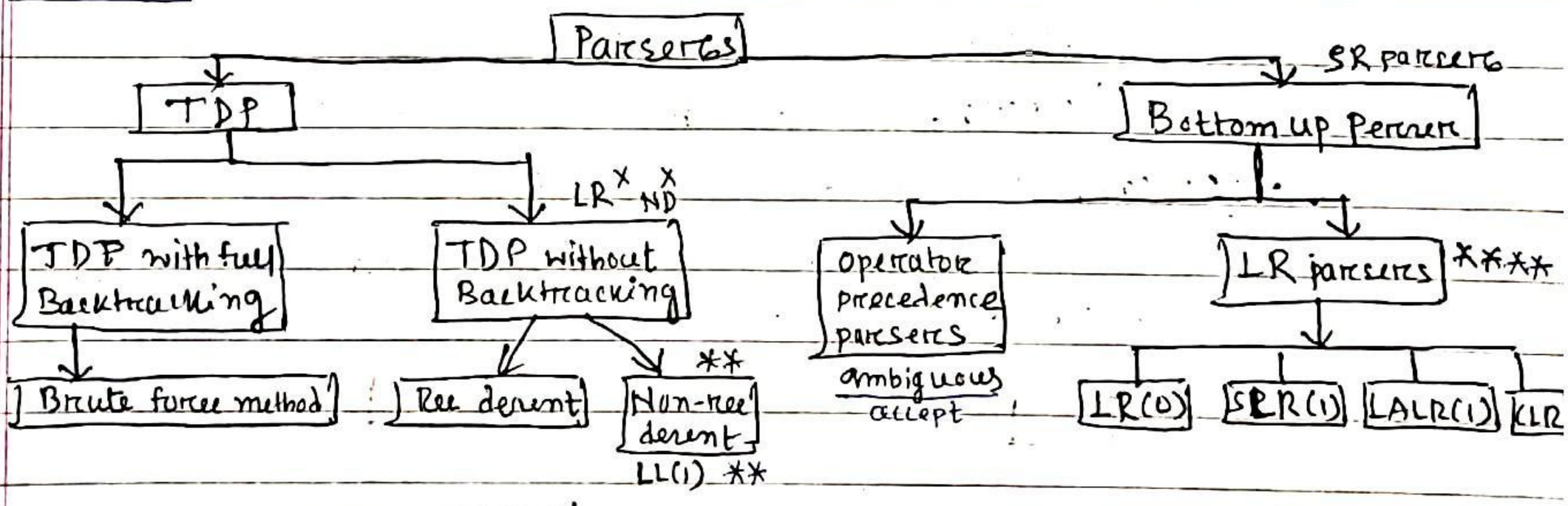
$$(A) \to (A)\alpha/\beta \iff A \to \beta A' \\ A' \to \epsilon/\alpha A'$$ } → left recursion removed.

- Non-Deterministic:

$$A \to \alpha\beta_1/\alpha\beta_2/\alpha\beta_3 \Rightarrow A \to \alpha A' \\ A' \to \beta_1/\beta_2/\beta_3$$ } Non-Deter removed (Left factoring)

Eliminating non-determinism or left-factoring doesn't eliminate ambiguity.

PARSERS:



- **First & Follow:**

	First	Follow
ex: S → aBDh	{a}	{$}
B → cC	{c}	{g, f, h}
C → bC/ϵ	{b, ϵ}	{g, f, h}
D → EF	{g, f, ϵ}	{h}
E → g/ϵ	{g, ϵ}	{f, h}
F → f/ϵ	{f, ϵ}	{h}

*** LL(1) Parsing: (first & follow) → table → table cell contain ┌ (1) production (LL1) ✓
                                                                └ more than (1) produce (LL1) X

⇒ left Recursion & Non-Deterministic grammar can't used for (LL1).

⇒ All ∈-production have to place under follow of "left hand side".

• Operator precedence parser:

→ can parse ambiguous grammar.

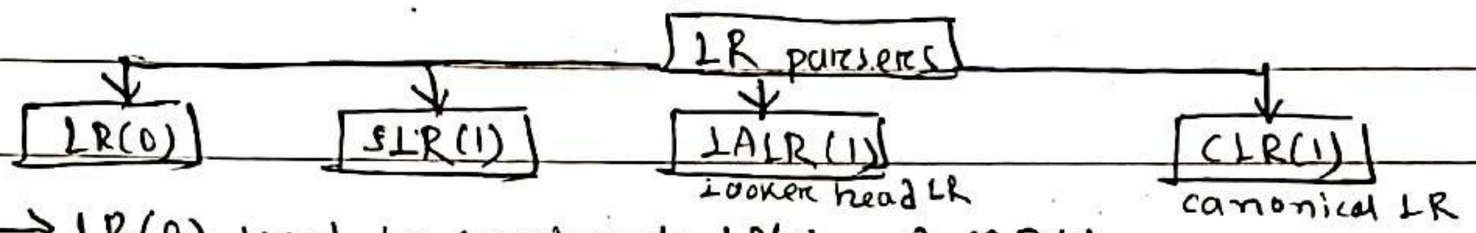
✓ [E → E+E / E*E / id] → no two variable are adjacent. id > * > +

X [E → EAE / id
   A → + / *]

→ if we have 'n' operator then size of operator precedence table n^2 approx

→ to Decrease table size (use function table) $n \xrightarrow[\text{no-operator}]{} 2n$ (size of table)

*** • LR parsers: (Bottom up)



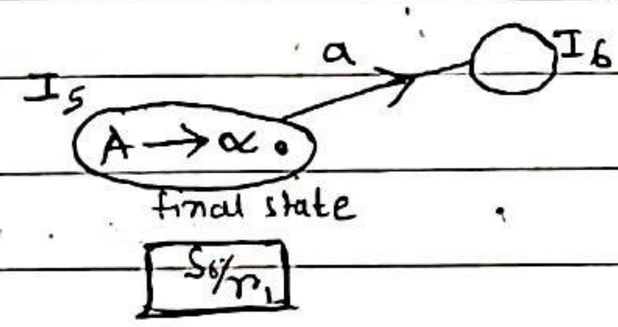
→ LR(0) used to construct LR(0) and SLR(1).

→ LR(1) used to construct LALR(1) and CLR(1).

• LR(0) → [grammar + add one more (S'→.S)] → Diagram → transition table.
                                                  ↳ ** find the no. of state ?

• SLR(1): main Diff b/w LR(0) & SLR(1) is Reduce (r) move s.o
   ↳ SLR(1) more powerful than LR(0)
   ↳ SLR(1) has less reduce move compare to LR(0).

→ (1) SR Conflict (shift Reduce) → (2) RR conflict (Reduce Reduce)



** → a grammar can't be LR(0) when it is RR conflict and SR conflict.

→ when a grammar is LR(0), then this grammar also SLR(1).

— But when grammar is SLR(1) then it may or may not LR(0).

• LR(1): (CLR, LALR(1)) +

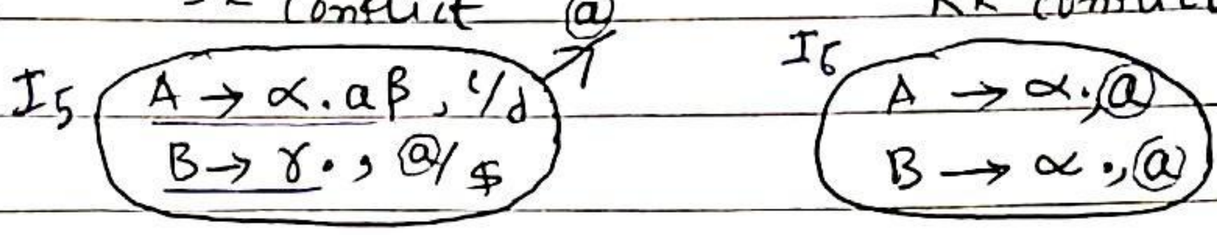
→ state may increase in LR(1) compare to LR(0) parser.

↪ no. of state increase because of lookerhead.

** ⇒ no. of state: LR(0) = SLR(1) ≤ CLR(1).
                        = LALR(1)

**

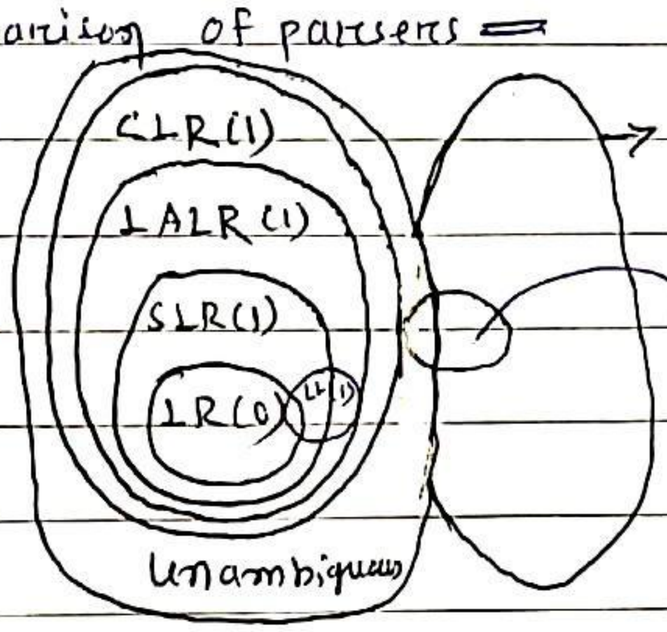
→ LR(1) items:



→ if a grammar not accepted by CLR(1) parser that can't be accepted by LALR(1) parser.

→ One grammar accepted by CLR(1), that gramm may or may not be accepted by LALR(1) parser.

• Comparison of parsers =



→ When a grammar are accepted by LL(1), then it also accepted by LALR(1).

• Syntax Directed Transiation:

⇒ E → E*T / T
T → F+T / F
F → 2/4
} In this gramm [+ > *] '+' has higher precedence.

• SDT (2) Types → (1) S-attributed [only synthsized] (Bottom up per)
↳ (2) L-attributed [both inherited & synthesized] (dept first, Lto Right)

• A → BCD
A.S → f(B.S, C.S, D.S)
'A' is taking value from children.
↳ Synthesized attribute

• A → BCD 'C' taking value from
C.i → A.i from parents &
C.i → B.i siblings
C.i → D.i
↳ inherited attribute.
↳ A → XYZ { Y.S = A.S ✓
Y.S = X.S ✓
Y.S = Z.S X }

• Intermediate Dode Generation: