COMPILER DESIGN - SHORT NOTES.

* Preprocessor Functionality: To understand any statement that begins with #; e.g - #define, #if - - -

* Assembler converts Assembly Code to M/C code.

* Linker resolves all external references. e.g - extern.

* Loader performs allocation/deallocation/reallocation of data in the Main Memory. It knows which location on the memory is free as it resides in the main memory.

Analysers		Synthesizers	
FA ←	Lexical Analysis	I.C.G	CG
PDA ←	Syntax Analysis	I.C.O.	CO
LBA ←	Semantic Analysis		

Frontend (M/C independent) — Backend (M/C dependent)

SYMBOL TABLE:

* Every function/scope can have its own Symbol table.
* It is a DS that stores info about attrs of each iden.

LEX: — Scanner / Linear phase Analyser / Token recognizer or Generator.

* lexeme → smallest unit in the program.
* Token → represents lexeme internally.

temp xyz — te × x, tem × xy, temx × xyz, temp ✓

Scanner identifies lexeme/Token using "longest prefix match rule" (a.k.a. Maximal Munch Rule).

E.g: $T_1 = a^*ba$ and $T_2 = b^*ab$, now, a string "bababa" is read by the lexer as

b → ba → bab ⟷ baba

b	ba	bab	baba
T_1	T_1	(T_2) ✓	~~T_2~~
T_2	T_2	~~T_1~~	F

∴ $\boxed{bab}\ T_2$

Now in string ~~bab~~aba

a	ab	aba
T_1	T_1	T_1
T_2	T_2	~~T_2~~

∴ $\boxed{aba}\ T_1$

NOTE: Comment is ignored in length/no. of tokens.

• String → "abc" is 1 token.

SYNTAX : ~~tokens~~ stream of tokens $\longrightarrow$ Parse Tree.

Parsers.

(*) #states (LR(0)) = #states (SLR(1)) = #states (LALR(1)) $\leq$ #states (LR(1))

Topdown (LMD)

- LL(1)

$\boxed{\text{→ Unambiguous, → No left Recursion, → left factored}}$

OR ***

For each production

$S \to Sa \mid Aa \mid Bd \mid bc$

- Check $Fi(Sa) \cap Fi(Aa) \cap Fi(Bd) \cap Fi(bc) == \emptyset$
- If $\neq \emptyset$, then not LL(1)

(*) Every entry in LL(1) Table has atmost (only) 1 production.

AMBIGUOUS → NOT LR(1) → NOT LL(1)

NOT LR(1) → NOT LALR(1) → NOT SLR(1) → NOT LR(0)

LR(0) → SLR(1) → LALR(1) → CLR(1) → LR(1)

$\boxed{LL(1) \subseteq LR(1)}$

LL(1) → LR(1)

LL(1) $\not\leftarrow$ LR(1)

Bottom up. (Rev. of RMD)

- LR(0) least powerful
  - SHIFT-REDUCE : 1 state contains both S & R
  - REDUCE-REDUCE : 1 state " two Rs.
  - → If Ambiguous CFG or conflict then not LR(0).

- SLR(1)

  Computes Lookahead based on Grammar.

  SHIFT-REDUCE

  $X \to \alpha . t\beta$

  $Y \to \gamma .$

  $\boxed{t \in Fo(Y)}$

  Reduce-Reduce

  $X \to \alpha .$

  $Y \to \beta .$

  $\boxed{Fo(X) \cap Fo(Y) \neq \emptyset}$

- LALR(1) Most preffered (path of conflict)

  write lookaheads based on follow.

  SR-Conflict

  $X \to \alpha . t\beta, L_1, L_3$

  $Y \to \gamma ., L_2, L_4$

  $\boxed{t \in (L_2, L_4)}$

  RR-Conflict

  $X \to \alpha ., L_1, L_3$

  $Y \to \beta ., L_2, L_4$

  $\boxed{(L_1, L_3) \cap (L_2, L_4) \neq \emptyset}$

- CLR(1) Most powerful

  Same as LALR, but same production with two different lookahead, should be in different states.

  e.g.

  $A \to d., a$ / $B \to d., b$

  $A \to d., \$$ / $B \to d., \$$

  SR-Conflict

  $X \to \alpha . t\beta, L_1$

  $Y \to \gamma ., L_2$

  $\boxed{t \in L_2}$

  RR-Conflict

  $X \to \alpha ., L_1$

  $Y \to \gamma ., L_2$

  $\boxed{L_1 \cap L_2 \neq \emptyset}$

*** First Set (X)

- $X \to t\alpha$ $\quad Fi(X) = \{t\}$
- $X \to Y\alpha$ $\quad$ If $Y \neq \varepsilon$, $Fi(X) = Fi(Y)$

  If $Y = \varepsilon$, NOT EASY

** Never Contains ($).

*** Follow Set (X)

- $A \to \alpha X t \beta$, $Fo(X) = \{t\}$
- $Y \to \alpha X$, $Fo(X) = Fo(Y)$
- $Y \to \alpha X Z$,

  If $Z \neq \varepsilon$, $Fo(X) = Fi(Z)$

  If $Z = \varepsilon$, NOT EASY

** Never Contains (ε)

NOTE: If we combine states of CLR(1) without conflict, then the resultant LALR(1) ~~will~~ may have only R-R conflict.
It will never have S-R conflict.

as → $t \notin L_1$ and $t \notin L_2$, then $t \notin (L_1 \cup L_2)$

Similarly, LALR(1) without conflict → CLR(1).
Then CLR(1) may have only RR conflict.

- SizeOf (LR Tables) = $|Q| \times (|T| + |V| + 1)$
- Sizeof (LL Tables) = ~~$|T| \times (|V| + 1)$~~ ✗
- #Reduced Entries (LR(0)) ≥ #Reduced Entries (SLR(1)) ≥ #RE (LALR(1)) ≥ #RE (CLR(1))

OPERATOR PRECEDENCE PARSER (Bottom-up Parsers) (Can be Amb/Unam. (CFGs))

OPERATOR GRAMMAR (OG) + Precedence Relations = Opr. Prec. Gram.

- No Consecutive NTs
- No Null Production.

A → A+B | ε ✗ OG

C → DE | a ✗ OG

$l <\cdot r$

$l \cdot> r$

$l \doteq r$

$r \cdot> l$ ✗

If OG = Ambiguous
— explicit P. Rules

If OG = UnAmbiguous
— derive P. Rules.

Q. For the below given O.G.

E → T+F

T → G*H | id

F → I-J | id

G → id

H → G*id | id

I → id

J → id

(i) + is ___ associative.

(ii) * is ___ associative.

(iii) − is ___ associative.

(iv) Relation between

(a) + & *

(b) + & −

(c) * & −

(d) +, * & − .

Answer: (i) NO

(ii) RIGHT

(iii) NO

(a)(iv) $* \cdot> +$

(b)(v) $+ <\cdot -$

(c)(vi) No relation

(d)(vii) $* \cdot> +$, $+ <\cdot -$, no relation between * & −

SEMANTIC: PARSE TREE $\longrightarrow$ SEMANTIC TREE (Annotated P.T) (Decorated ").

STEP 1: Scope of variables.
" 2: Type checking and Add type to Symbol table.
" 3: Declaration and its use (for variables).
" 4: Function Compatibality.

SYNTAX DIRECTED TRANSLATION: CFG + Translation. ≅ ~~STD~~ SDT
(Semantic, output, ...)

***ATTRIBUTE

** (*) We donot consider function attributes.
(*) If attr. = constant, then, x is both.

Inherited
↓
→ Computation depends on parents or Siblings

$S \to AaBC$
$\{A.\tau \leftarrow S.\tau - B.\tau\}$

Synthesized
↓
Comp. depends on children.

$S \to AaBC$
$\{S.\tau \leftarrow A.x - B.x\}$

***ATTRIBUTED GRAM./DEFN.

L-Attr. S-Attr.

L-Attr.:
(i) depends on parent/children/left sibling*.
(ii) {...} can be anywhere.

S-Attr.:
(i) depends on children.
(ii) {...} at the end of production

EVALUATION OF STD's: $\longrightarrow$ without attributes (multiple ways)
$\longrightarrow$ with Attributes (S-Attr and L-Attr Grammar)

- L-Attributed Evaluation: Depth left $\to$ left to Right.
- S-Attributed Evaluation: Reverse of RMD.

Q. Check whether the attributes are Inherited or Synthesized.

① $S \to (S_1)$ $\{S.x = S_1.x + 1\}$
$S \to S_1 S_2$ $\{S.x = S_1.x + S_2.x\}$
$S \to \varepsilon$ $\{S.x = 0\}$

② $S \to a$ $\{S.x = 10\}$
$S \to b$ $\{\ \}$

③ $S \to a$ $\{S.x = 10\}$

④ $S \to a$ $\{\ \}$

⑤ $S \to AB$ $\{A.\cancel{x} = B.x\}$
$A \to a$ $\{A.x = a.value\}$
$B \to b$ $\{B.x = b.value\}$

⑥ $S \to D\,L;$ $\{L.type = D.type\}$
$D \to int$ $\{D.type = integer\}$
$L \to id$ {AddST(id.type, id.lexeme)}

⑦ $E_1 \to E_2 + a$ $\{E_2.x = E_1.x\}$
$E \to b$ $\{E.x = 100\}$

Answers: ① Syn. ② Both Syn & Inh.
③ Both ④ No attribute.
⑤ neither inh. nor syn.
⑥ ——— " ———
⑦ Inherited.

ICG: Semantic Tree ⟶ Intermediate Code

Postfix 3AC SSA ⋮	Syn. Tree DAG CFG ⋮
Linear IC	Non-Linear IC

Consider:

$x = (a+b)+(a+b);$

Postfix

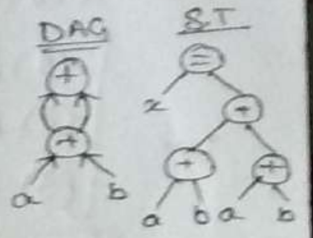
$xab+ab++=$

3AC

$a = a+b$
$b = a+a$

SSA

$a_1 = a+b$
$b_1 = a_1+a_1$



Three Address Code Notations: Eg: $x = a+b$ where $\&x = 1000$

ⓐ Triple Notation

opr	opn1	op2
+	a	b

1000

ⓑ Quadruple Notation

opr	opn1	opn2	result
+	a	b	x

1000

ⓒ Indirect Triple.

opr	op1	op2
+	a	b

2000

result
2000

1000

I Q. Find min. no. of variables in Best 3AC of the following expr-

① $x = a+b$ ② $x = a+b$, $y = x+a$ ③ $x = a+b$, $y = x-a$ ④ $x = a+b$, $y = a+b$, $z = x*y$ ⑤ $x = a+b$, $a = x+c$, $z = a+b$, $y = y*z$.

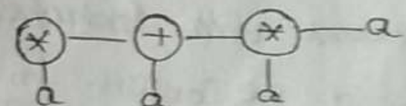
II Q. Find the min. no. of nodes and edges in DAG

① $x = a+b$ ② $x = a+\cancel{b}*a$ ③ $x = a+b*a+b$

④ $x = a*b+c+a*b*d+c$

III Q. Find min no. of variables in equivalent SSA Code

① $x = (a+b)*(a+b)$ ②



Answer: (I) ① = ② = ③ = ④ = 2 ⑤ = 4 (II) ① 3n & 2e ② 4n & 4e ③ 5n & 6e ④ 9n & 10e (III) ① 4 ② 4

⊛ SSA is always 3AC but best 3AC is never SSA.

—×—

CG: Control Flow Graph.

Q. Find the number of (i) Basic Block & Control lines
(ii) Nodes & Edges

ⓐ
```
   x = 2
A: y = x+i
   if(y>100) goto A
   x = j+k
   y = x+i
   If(y>100) goto A
   z = y+x
```

ⓑ
```
x=20
for(i=0; i<n; i++){
    y=x+i;
}
z=x+y
```

Ans: ⓐ (i) 4 BB & 5 CL
(ii) 6n & 7e

ⓑ (i) 4 & 4
(ii) 6 & 6.

NOTE

#nodes = #BB+2

and

#edges = #CL+2.

* PREVIOUS QUESTIONS.

Lex

(1) DAGs are used for Code Optimization.

(2) Abelian Groups are associated with Push Down Automata.

(3) Keywords are recognized in Lexical Analysis.

(4) Front End is reffered to as Pass 1, and backend as Pass 2

(5) LMD and RMD for unambiguous/ambiguous Grammar may/maynot be different.

(6) Operator Grammar can be LL(1).

$S \to aAcA \mid bBcA$
$A \to a$
$B \to a$

(7) Handle : RHS of a production, Only for Reduce operations, in the immediate next step.

$\underbrace{S \to A}_{\text{First step}} \to \omega$, $\underbrace{A \to \omega}_{\text{next step}}$, then ω is the handle.

***

(8) Viable prefix — Sequence on Top of STACK.
— Prefix of Right Sentential Form.

(9) Every Uambiguous CFG $\not\to$ / $\leftarrow$ LR

(10) If Yacc detects SR conflict, it resolves S-R conflict in favour of Shift over reduce. (LR, LL not OP)

~~In parsers with lookahead and~~

(11) In Case of SR Conflict in Yakk, shift (on stack) is preffered so always Right associative grammar.

(12) Every regular → DCFL gramar ~~does~~ is LR but not LL.

(13) $S \to aSb \mid Sb \mid d$ ⇒ Left Rec. Gramar.

⇓

$S \to aSbS' \mid dS'$
$S' \to bS' \mid \varepsilon$.
] Not Left Rec. Grammar.

Study about shared, dynamically and statically linked libraries.