UNIT-III

SDD (Syntax Direlted Definition)

SDD is a context free grammar together with attributes and rules

Attributes are associated with grammer symobls & rules are associated with productions

If x is a symbol and a is one of its attribute then we write X.a.

To kinds of attributes for non terminals ~~sy~~

1) Synthesized attribute
2) Inherited attribute.

1) Synthesized attribute :

A synthesized attribute at node N is defined only in terms of attribute values at the children N and afet 'N' itself

2) Inherited attribute :

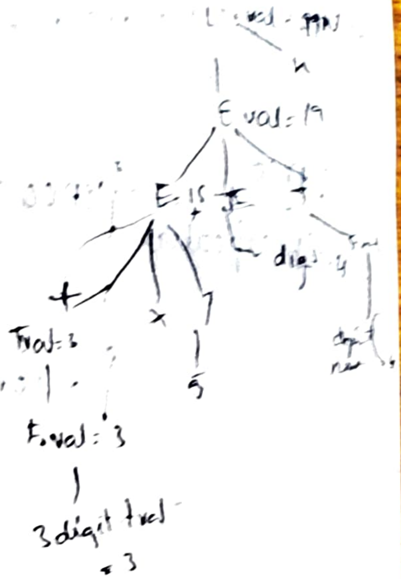
An iherited att at node N is defined only in terms of attribute values at parent , sibilings at itself.

* Annotated parse tree

A parse tree show the vals of at the attributes is called annotated parse tree

Ex.

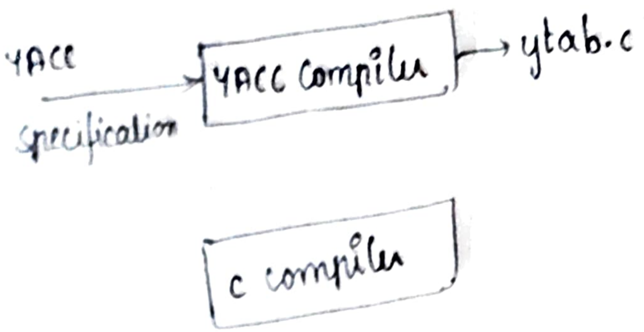
Production	Semantic rules
L → En	L.val = E.val v
E → E + T	P. val = E1.val + T.val
E → T	E. val = T val
T → T * F	T.val = T1 .val * F.val
T → F	Tn val = F.val
F → (E)|d	F.val = (E.val)



★ diff b/w S-attributed & L-attributed

1) uses only synthesized attri	1) Uses both synthesized & inherited attribute
2) Symantic actions are placed at right end of a production A → BC{}	2) Symantic actions are placed any where. A →{}BC / A→BC{}|A→B{}C
3) attributes are evaluated during bottom up parsing	3) att are evalued by translating parse tree to depth first
4) S-att grammar is based on LR grammar.	4) L att. grammer is based on LL grammar.
5) These are implemented using LALR parser	5) Implemented using predictor parser

★ YACC (Yet another compiler-compiler)



In the part of YACC specification a set of production

<head> → <body1> / <body2> | . . . <body n>

could be written in YACC as

★ <head> : <body1> {semantic action 1}

<body2> {semantic action 2}

:

<body n> {semantic action}

A YACC semantic action is a sequence of C statement the symbol $$ refers to the attribute value associated with the non-terminal ~~non~~ on left hand side $i refers to the value associated the ith grammar symbol on the right hand side.

```
E → E+T/T
T → T*F|F
F → (E) |Digit

%%
E. E+T { $$ = $1+ $3}
    |T
T:T *F { $$ - $1 * $3}
    |F
F:(E) { $$ = ( $2)}
    |digit
%%
```

Auxiliary Procedure

The 3rd section YACC specification consists of supporting C Routines

```
%{ #include <ctype.h>
          %}
  % token DIGIT
   %%
expr: expr+term{$$ = $1+$3}
        |term
term. term* factor { $$ = $1* $3}
        |factor
```

```
factor: (expr) {$$ = ($2)}
        |digit
        %%

yylex()
{ intc;
  c = getchar();
  if(isdigit(c))
  {
     yylval = c-'0';
      return DIGIT
  }
  return c;
  }

yacc filename.y
./a.out
```