Attribute Grammar

- An attribute grammar is a CFG in which the grammar symbols have attributes associated with them.

  Later on, we'll see that this actually *extends* the power beyond context-freeness, but the form of the grammar is similar to CFGs in the sense that there is still one symbol on the LHS (in general, this is called a phrase structure grammar).

- AGs help define form-meaning correspondences.

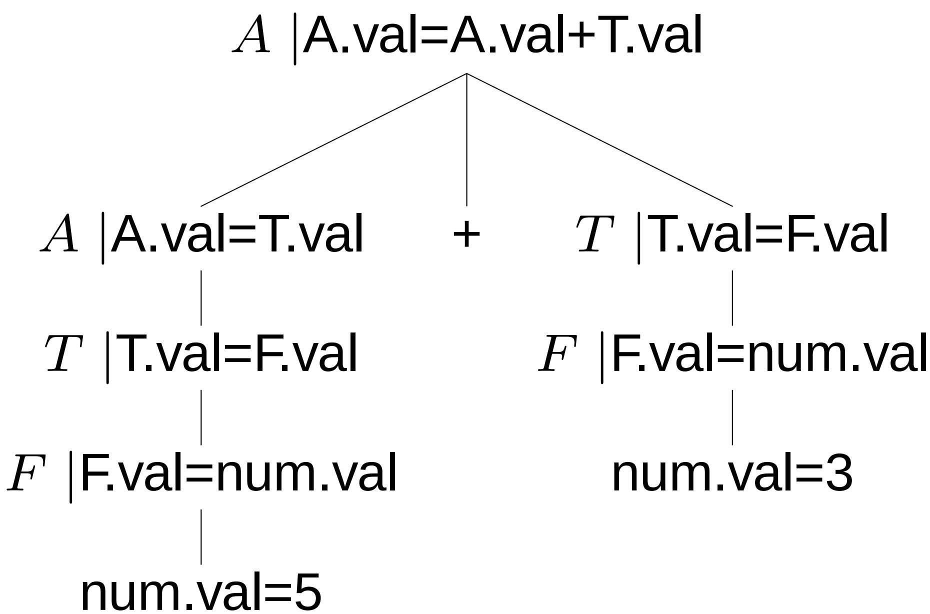
  ex: A calculator (this is syntax-directed evaluation)

```
CF rule     semantic action
--------    --------------------

A -> A+T    {A0.val = add(A1.val,T.val)}

F -> num    {F.val = num.val}
```

ex: a decorated (annotated) parse tree for 5+3



- In what order the information is passed?

  From RHS to LHS: synthesized attributes

From LHS to RHS: inherited attributes

- Synthesized: $X.a \rightarrow Y_1.a \cdots Y_n.a$

  X.a is a function of $Y_i.a$

- Inherited: $X.a \rightarrow Y_1.a \cdots Y_n.a$

  $Y_k.a$ is a function of X and $Y_i.a, i \neq k$

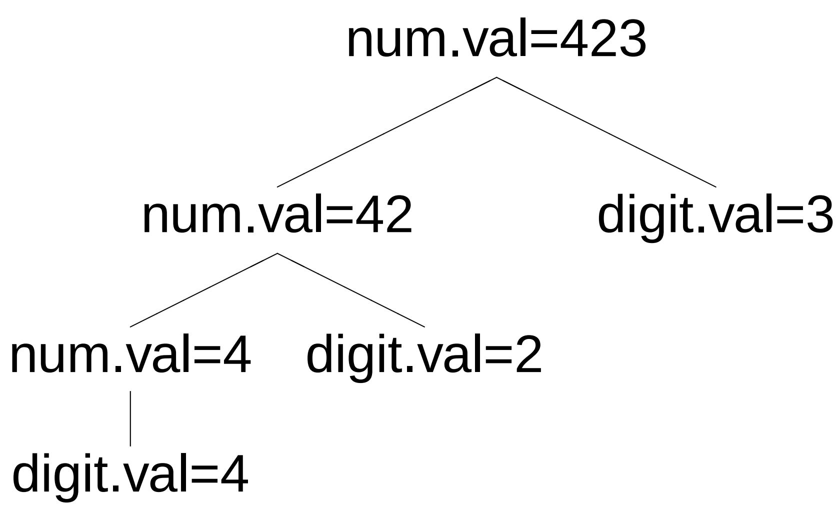
ex: synthesized vs. inherited derivation of numbers

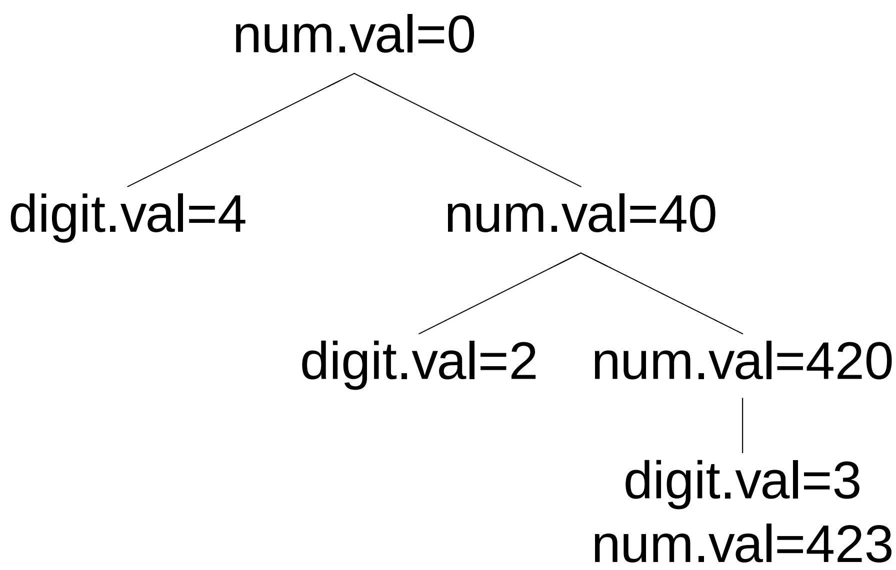
```
Num -> Digit     {num.val=digit.val}
Num -> Num Digit {Num1.val=Num2.val*10
                            +Digit.val}
```

```
Num -> Digit     {Num1.val=Num1.val
                           +Digit.val}
Num -> Digit Num {Num2.val=(Num1.val
                           +Digit.val)*10}

        assume initially num.val=0
```

423=





- Composition of semantics reflects the underlying parsing strategy as well.

  ex: checking the declaration of variables in top-down parse (assume

D.dl=nil initially)

```
P -> D S        {S.dl = D.dl}

D -> var V ; D  {D2.dl=addlist(V.name,D1.dl)}

D -> null       {}

S -> V := E ; S {check(V.name,S1.dl);
                 S2.dl=S1.dl}

V -> id         {V.name=id.val}
```

At what time do we execute the semantic action? In above convention, dependency of one attribute over another tells you when to execute (after D is recognized in 1st rule)

But, the time of semantic action can be made explicit by putting it in a position where it can be evaluated

```
P -> D {S.dl = D.dl} S
```

The latter convention is known as the *translation scheme*. It is a special case of syntax-directed definition in which rule evaluation and attribute evaluation use the same order and strategy.

But, in general, syntax-directed definitions can separate rule and attribute evaluation by dependency graphs.

- S-attributed grammars: only synthesized attributes

L-attributed grammars: All inherited attributes in a rule are a function only of symbols to their left

- if L-valued, a grammar can be used to parse top-down depth-first.

  If not, leftmost derivations are unable to evaluate Y_j for some $j > k$.

- YACC uses synthesized attributes
- antLR can do both: tree parsing
- Tree parsing decouples parsing strategy and semantic composition by

building Abstract Syntax Trees (AST), which can be traversed in any order to maintain the attribute dependency.