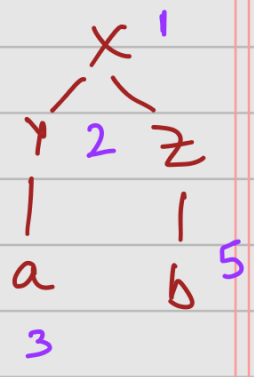
Lec 8/4

SDD ⇒ we do not write the order of evaluating statements.

each production has an attribute "ptr".

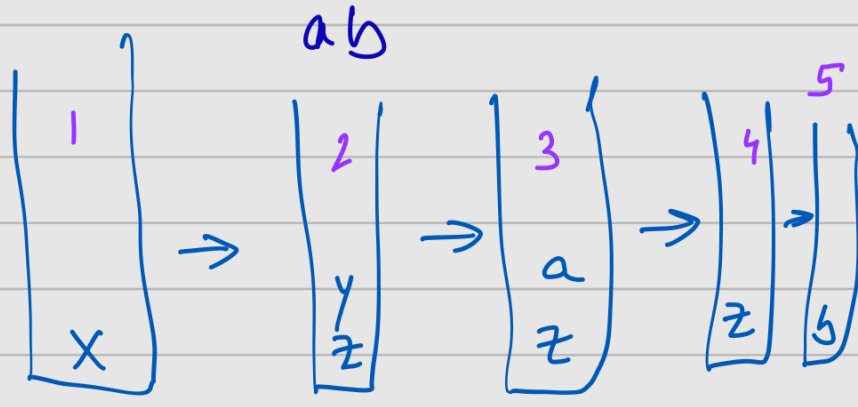
$T_1.ptr \leftarrow mknode(*, T_2.ptr, F_2.ptr)$

same non terminal not different.

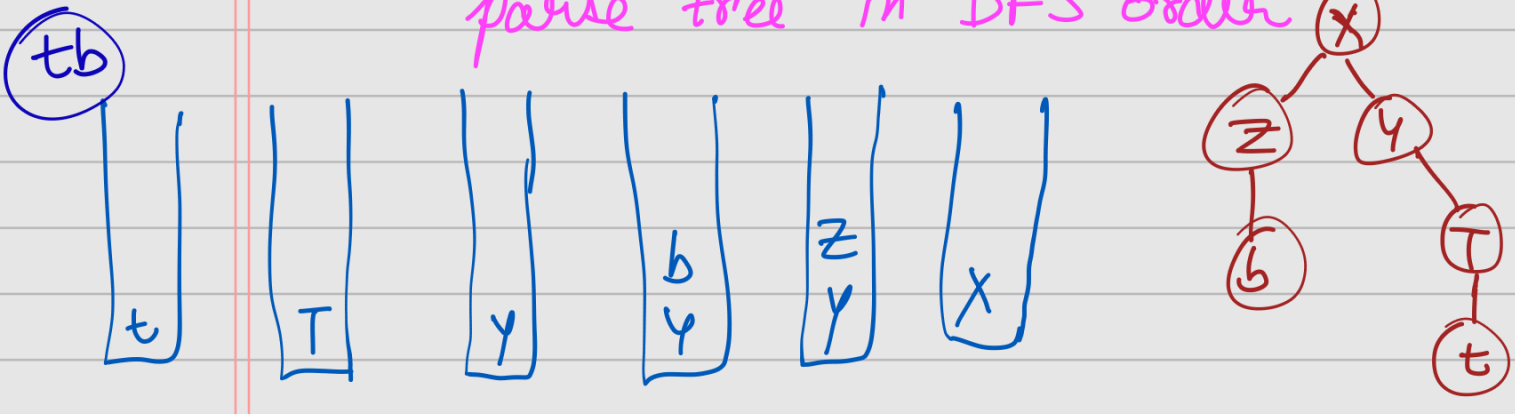


$X \rightarrow YZ$
$Y \rightarrow a$
$Z \rightarrow b$

DFS



LL parser traverses the parse tree in DFS order



$X \rightarrow YZ$
$Y \rightarrow T$
$Z \rightarrow b$
$T \rightarrow t$

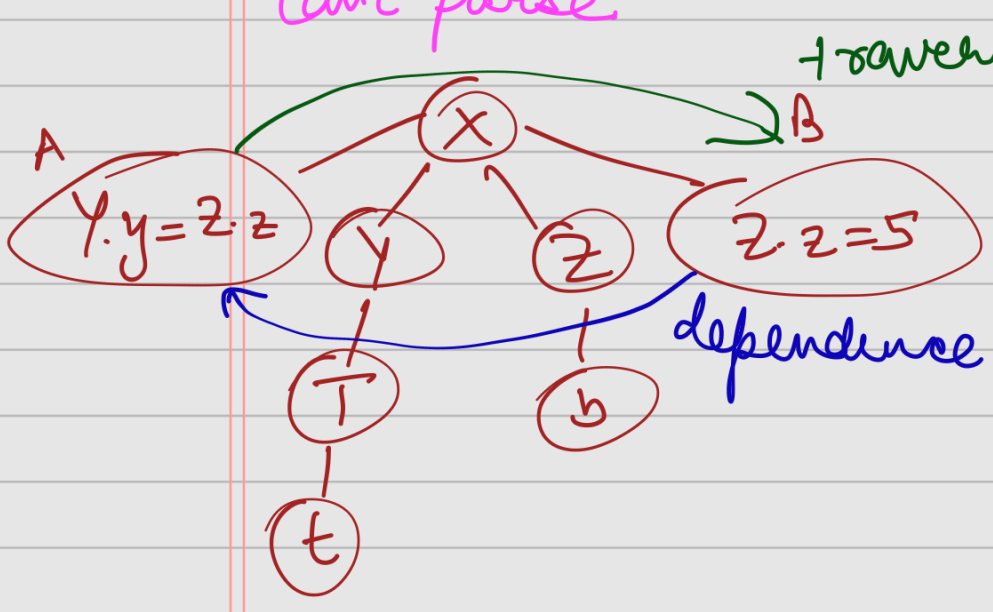
X Y T t T Y X Z b Z X

LR parser also traversed the parse tree in a DFS order, just that it does actions on backtracking time.

Terminals are visited only once

terminals are visited only once.

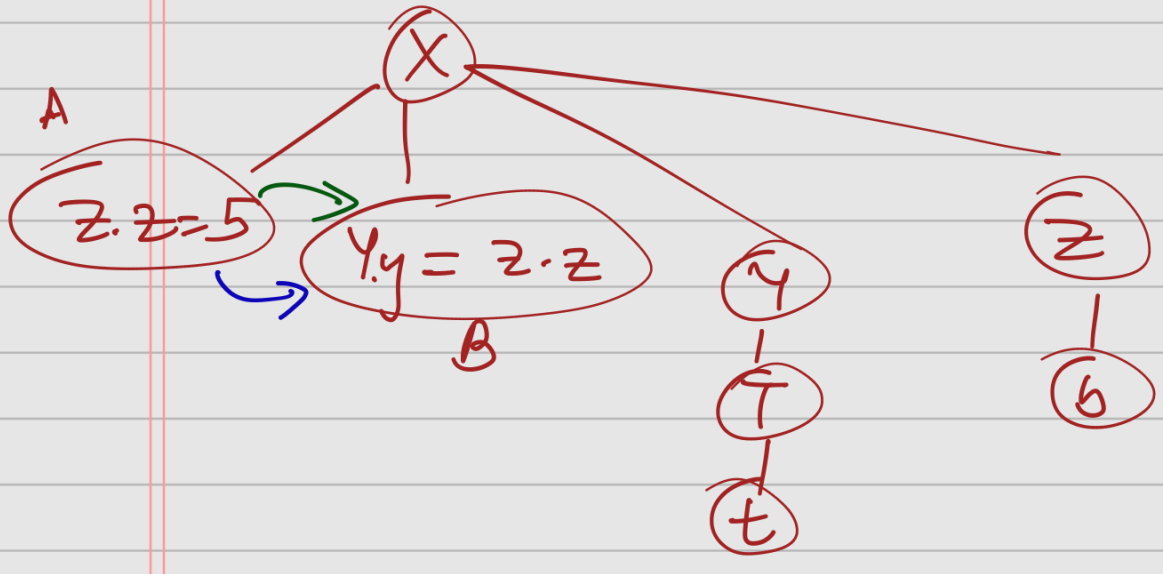
- AST also now contains code nodes of computations which come from dependence graph. There must not be any where traversal order b/w such nodes is ⟶ and dependency edge is ⟵ . Such an SDD we cant parse



Can't Parse

XAXYTtTYXZbZXBX

Traversal order



CAN parse

XAXBX YTtTYX Z b ZX

Such compute nodes will always be terminals after going to perform that.

computation when we come to that node, and since they are terminals, each will come only once in traversal order.

1) Check ⇒ ∀ instances of the SDD parser succeeds.
We want to figure out this by just looking at the parse tree. Not in the middle of generating parse tree we get error.

2) Clever ⇒ Attaches code nodes cleverly so that it gives correct order of putting SDD and compute nodes, so that we can parse it. ⟹

E → TR
R → addop T { print (addop) } R | ε
T → num { print (num) }

95 - 2 +

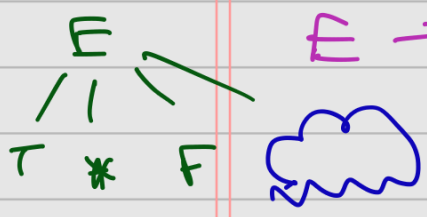
SDT (Syntax directed translations)

S- attributed Grammar — all attributes in the grammar are synthesized attributes NONE of them are inherited.

SDD - Content Free Grammar + Semantic actions
(construct dependence graph etc)

SDT – extend Context free Grammar with code symbols.

in S- attributed grammars, always put the code syntaces at the end of the production.



E → T * F