9-5+2 ⤳ 95-2+

expr → $expr_2$ + term {print('+');}

expr → $expr_2$ − term {print('−');}

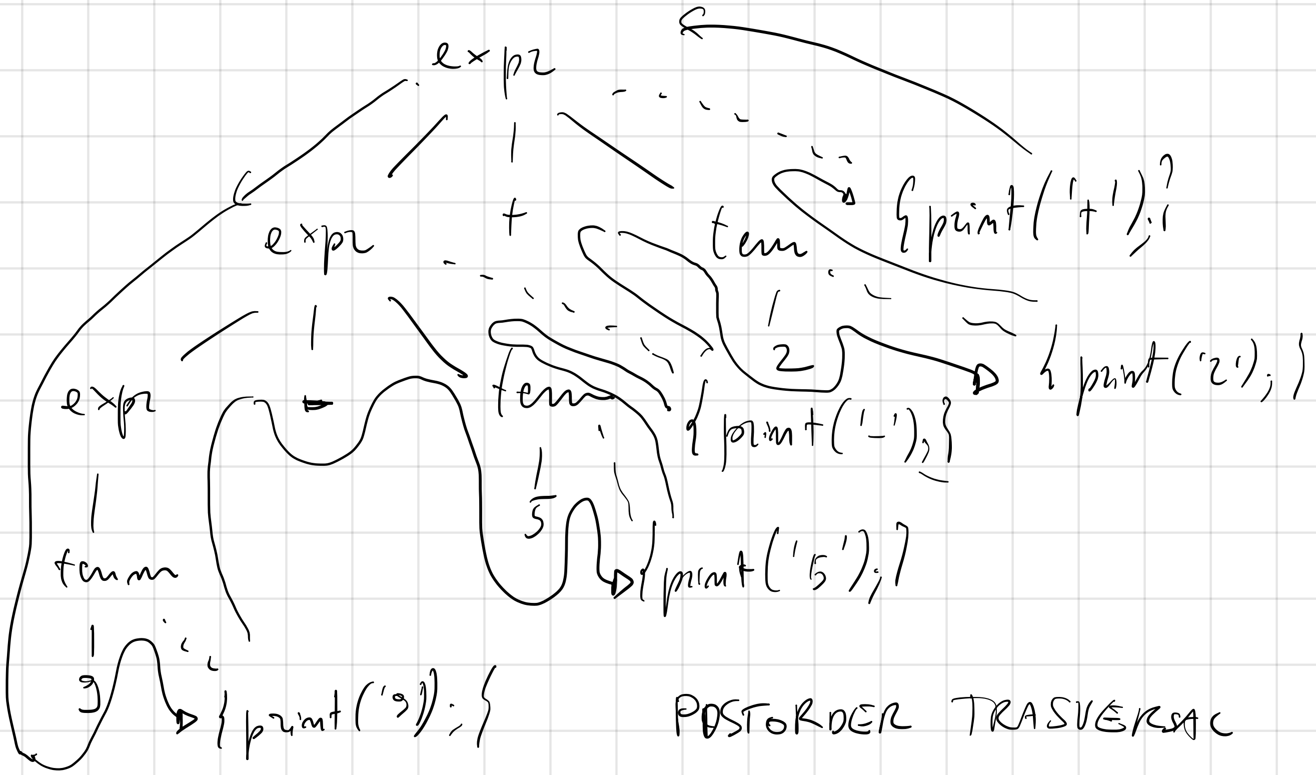
expr → term

term → 0 {print('0');}

⋮

term → 9 {print('9');}

9-5+2



POSTORDER TRASVERSAL

print('9); print('5'); print('−'); print('2'); print('+');

95-2+

$L \rightarrow E\ \underline{n}$ {print(E.val)} STD POSTFIX

$E \rightarrow E_1 + T$ $\{E.val = E_1.val + T.val\}$

$E \rightarrow T$ $\{E.val = T.val\}$

$T \rightarrow T_1 * F$ $\{T.val = T_1.val * F.val\}$

$T \rightarrow F$ $\{T.val = F.val\}$

$F \rightarrow (E)$ $\{E.val = E.val\}$

$F \rightarrow \underline{digit}$ $\{F.val = digit.lexval\}$

$3*5+2 \longrightarrow$ PREFIX notation $\rightsquigarrow$ $\boxed{+}\boxed{*\boxed{3}\boxed{5}}\boxed{2}$

$L \rightarrow E\ \underline{n}$

$E \rightarrow \underbrace{\{print('+');\}}_{\rightarrow M_1} E_1 + T$ $\quad M_1 \rightarrow \varepsilon$

$E \rightarrow T$

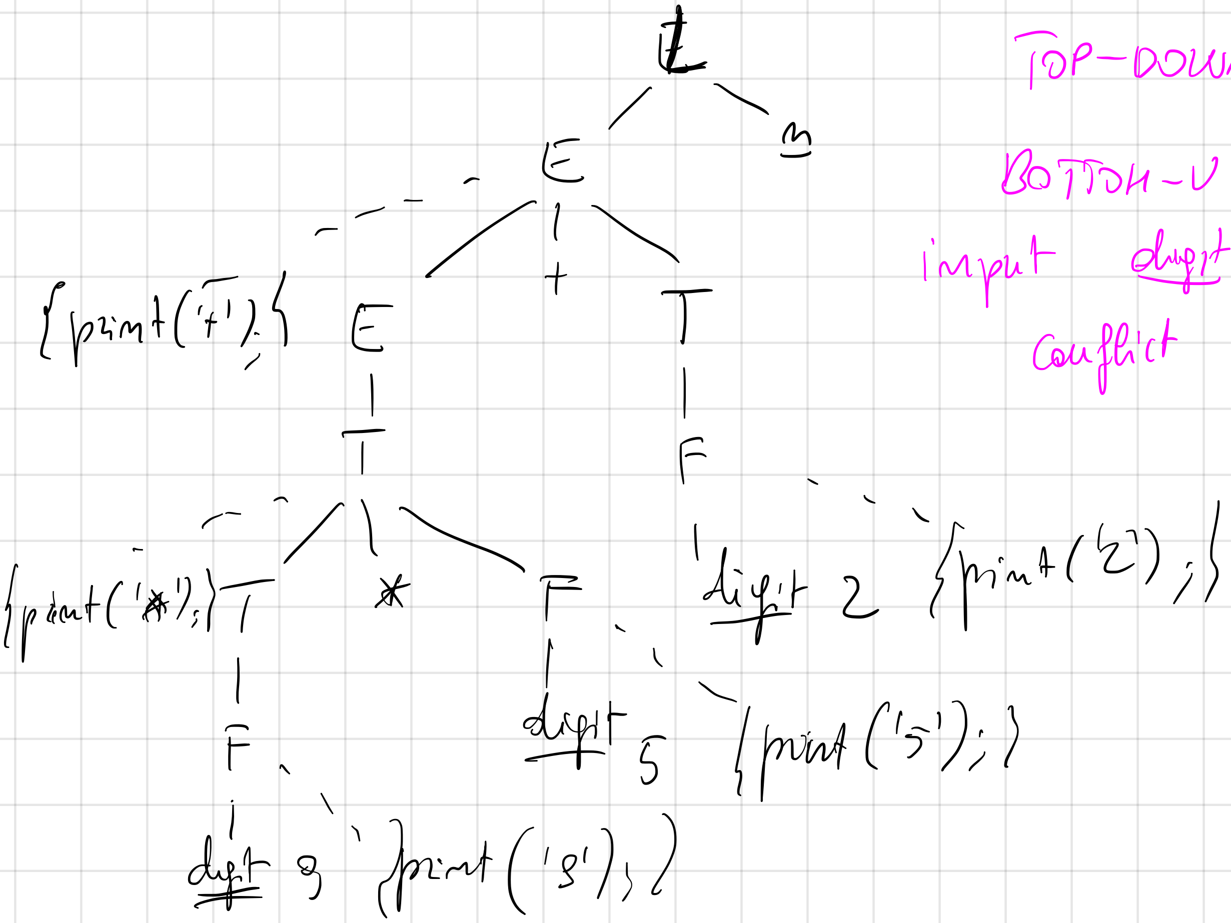
$T \rightarrow \underbrace{\{print('*');\}}_{\rightarrow M_2} T_1 * F$ $\quad M_2 \rightarrow \varepsilon$

$T \rightarrow F$

$F \rightarrow (E)$

$F \rightarrow \underline{digit}\ \{print(digit.lexval);\}$ $\quad \rightarrow M_3 \rightarrow \varepsilon$

$3*5+2$



TOP-DOWN (NO)

BOTTOM-UP

input $\underline{digit}$

conflict

Reduce M_1
Reduce M_2
Shift digit

print('+'); print('*'); print(3); print('5'); print('2');

$+ * 3 5 2$

$E \rightarrow E_1 + T \ \underbrace{\{print('+');\}}_{\text{as a symbol}}$ $\quad A \rightarrow A\alpha$

$E \rightarrow T$ $\quad\quad A \rightarrow \beta$

$\rightsquigarrow$

$A \rightarrow \beta A'$

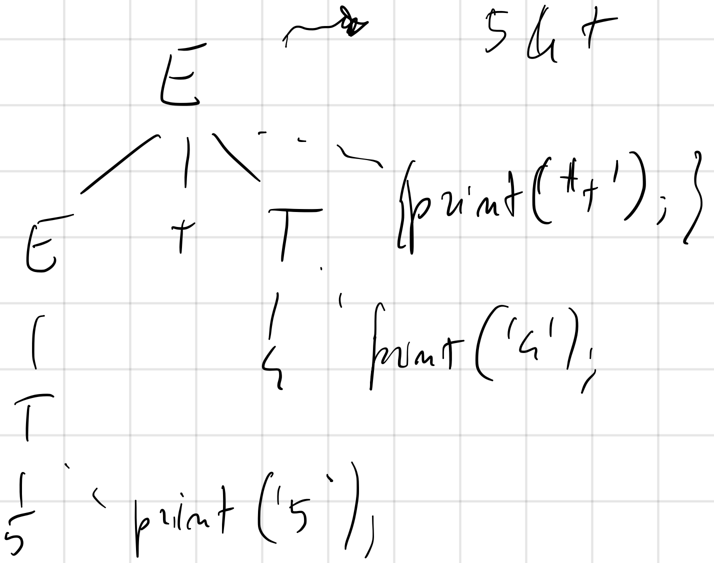
$A' \rightarrow \alpha A' \mid \varepsilon$

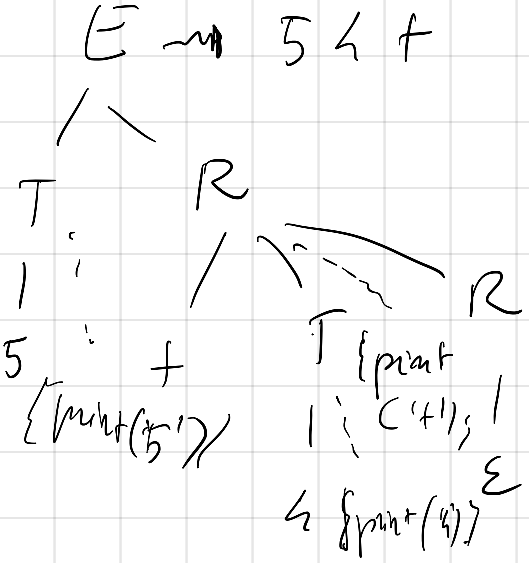
$\rightsquigarrow$

$E \rightarrow T R$

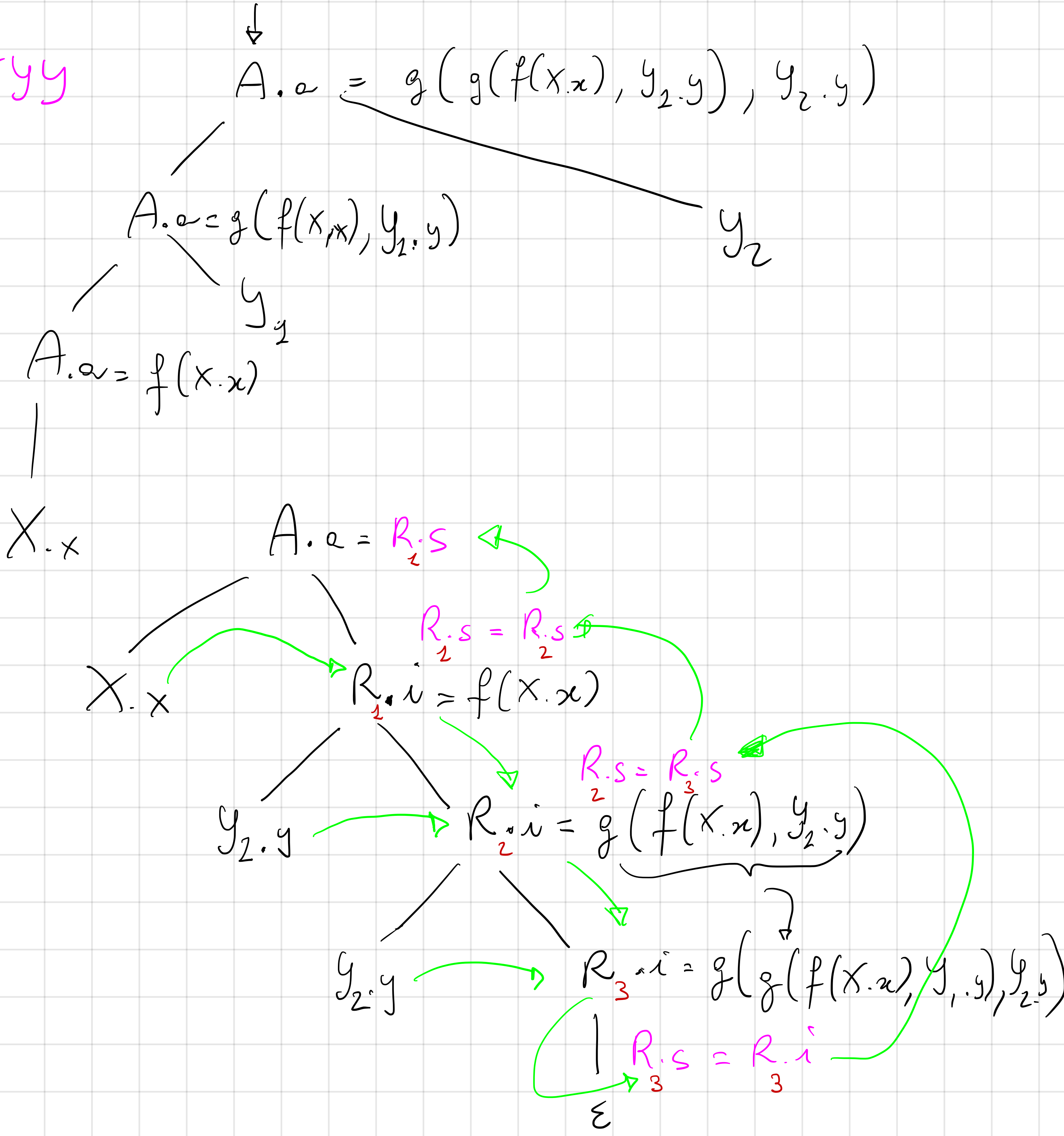
$R \rightarrow + T \ \{print('+');\} \ R$

$R \rightarrow \varepsilon$





XYY



$A.a = g(g(f(X.x), y_2.y), y_2.y)$
$A.a = g(f(X.x), y_1.y)$
y_2
y_1
$A.a = f(X.x)$
$X.x$
$A.a = R_1.s$
$R_1.s = R_2.s$
$X.x$
$R_1.i = f(X.x)$
$R_2.s = R_3.s$
$y_2.y$
$R_2.i = g(f(X.x), y_2.y)$
$y_2.y$
$R_3.i = g(g(f(X.x), y_1.y), y_2.y)$
$R_3.s = R_3.i$
ε

$\Sigma = \{ \&, A, B, C \}$

L = sequence of elements of Σ, else empty

& can be a quote to say that the next character must be interpreted as a command, unless the next character is & itself, which means that the character to translate is &

e.g. $\underbrace{\&\&}_{\&}\underbrace{\&A}_{Cmd(A)}\&\&B \longmapsto \& \underline{Cmd(A)} \& B$

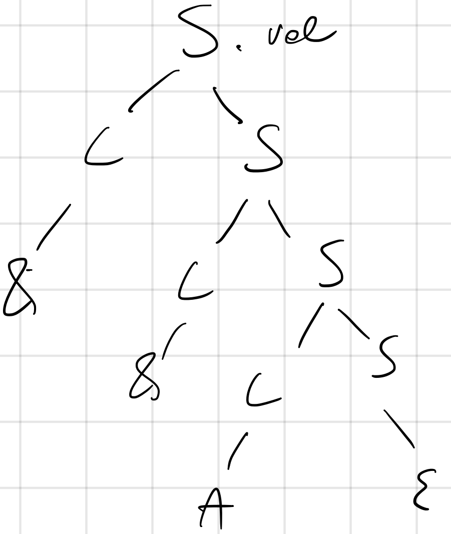
1) give a grammar for the language

2) give an SDT suitable for TOP-DOWN parsing that calculates as attribute of the initial symbol of the grammar the sequence in which the & quotes are resolved.

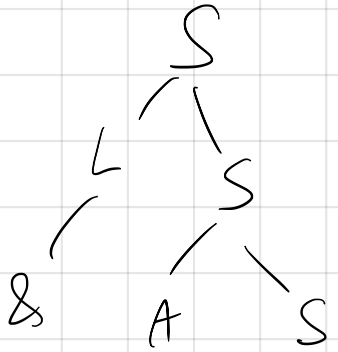
A grammar LL(1) for the language

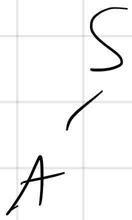
$S \to LS \mid \varepsilon$

$L \to \& \mid A \mid B \mid C$



&& A



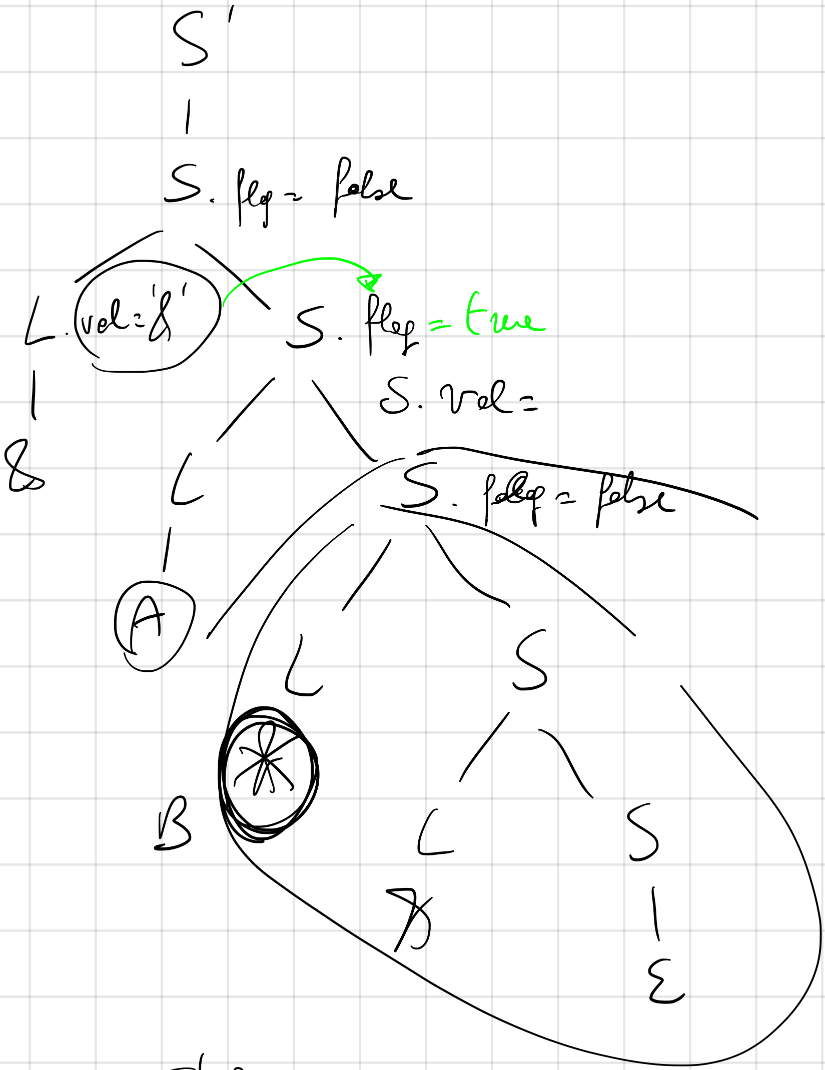


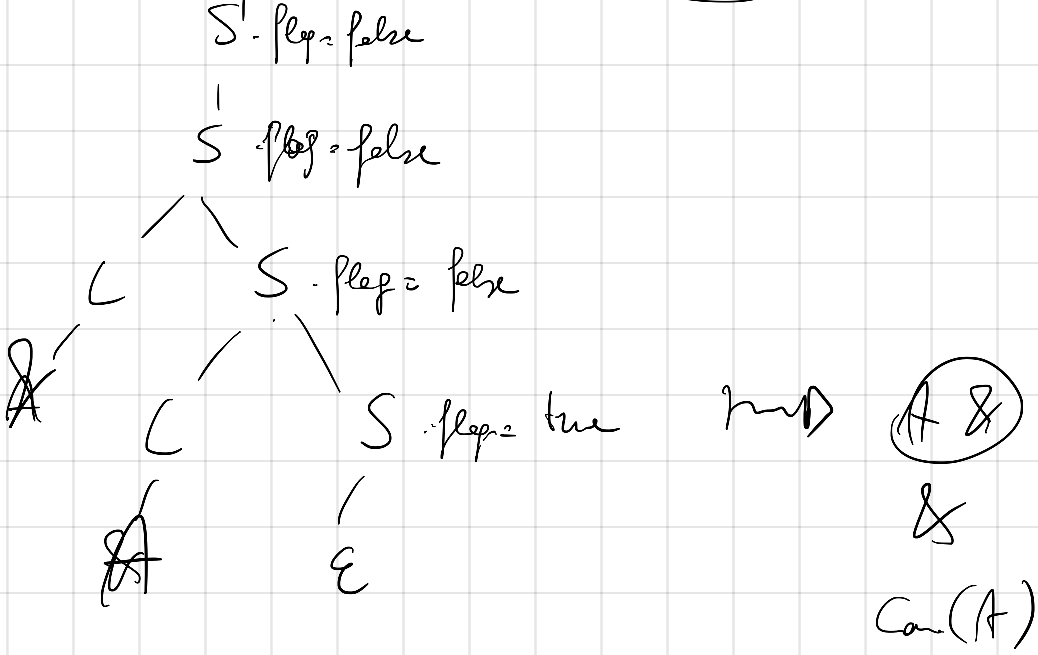
$S.val$ synth

$S.flag$ inherited records if "before" it was seen an &

$L.val$ synth SDD

```
S' → S          { S.flag = false; }
S → L S_1       { if ( S.flag ) then
S → ε                 if ( L.val = '&' )
L → &                    then [ S.val = '&' . S_1.val
                               S_1.flag = false
L → A                 else [ S.val = 'Cmd(L.val)' . S_1.val
L → B            else // before not &      S_1.flag = false
L → C                 if ( L.val ≠ '&' )
                         then [ S.val = L.val . S_1.val
                                S_1.flag = false
                      else [ S.val = S_1.Val
                             S_1.flag = true
```


S'
S. flag = false
L. val = '&'
S. flag = true
S. val =
&
L
A
S. flag = false
L
S
B
*
L
X
S
ε


S'. flag = false
S flag = false
L
S. flag = false
A
L
S flag = true
A
ε
A &
&
Con(A)