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

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. && & A && B ⟼ & Cmd(A) & B

↓ Cmd(A)

&

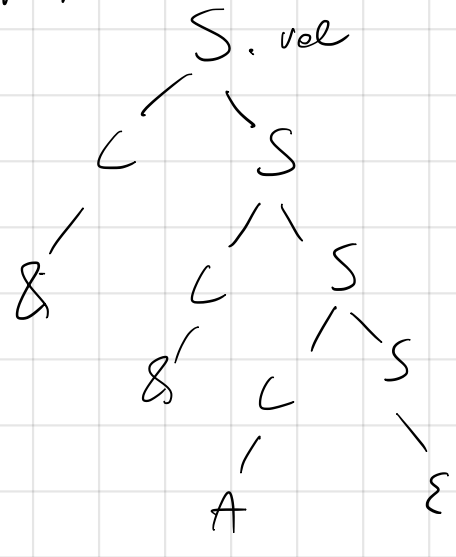
1) give a grammar for the language

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.

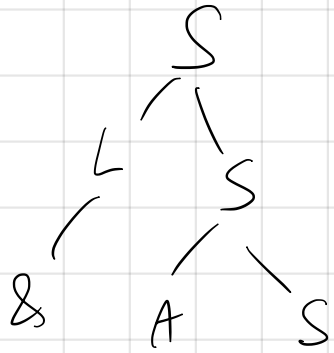
A grammar LL(1) for the language

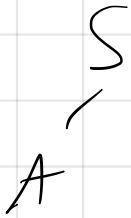
$S \to LS \mid \varepsilon$

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



&& A





$S.val$ synth

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

$L.val$ synth

SDD

$S' \to S$ { $S.flag = false$; }

$S \to L\ S_1$ { if ($S.flag$) then
if ($L.val = '\&'$)
then [$S.val = '\&' . S_1.val$
$S_1.flag = false$
else [$S.val = Cmd(L.val)' \cdot S_1.val$
else // before not & $S_1.flag = false$
if ($L.val \neq '\&'$)
then [$S.val = L.val \cdot S_2.val$
$S_1.flag = false$
else [$S.val = S_2.val$
$S_1.flag = true$

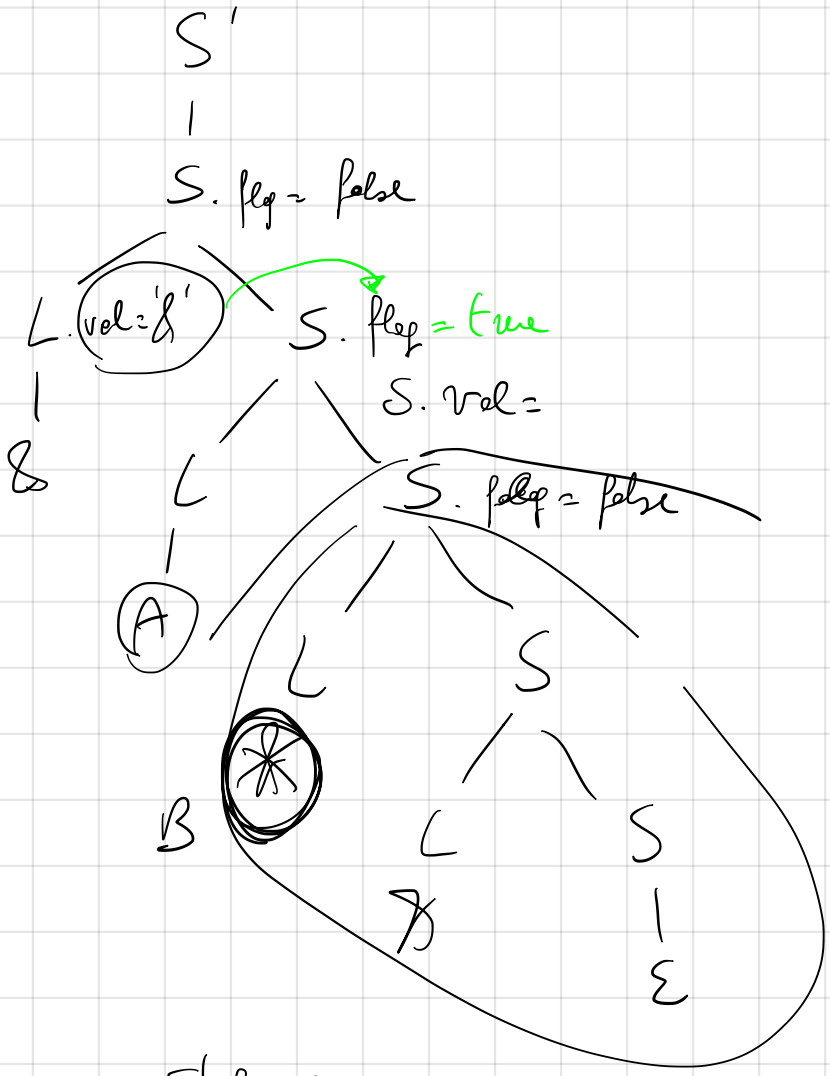
$S \to \varepsilon$

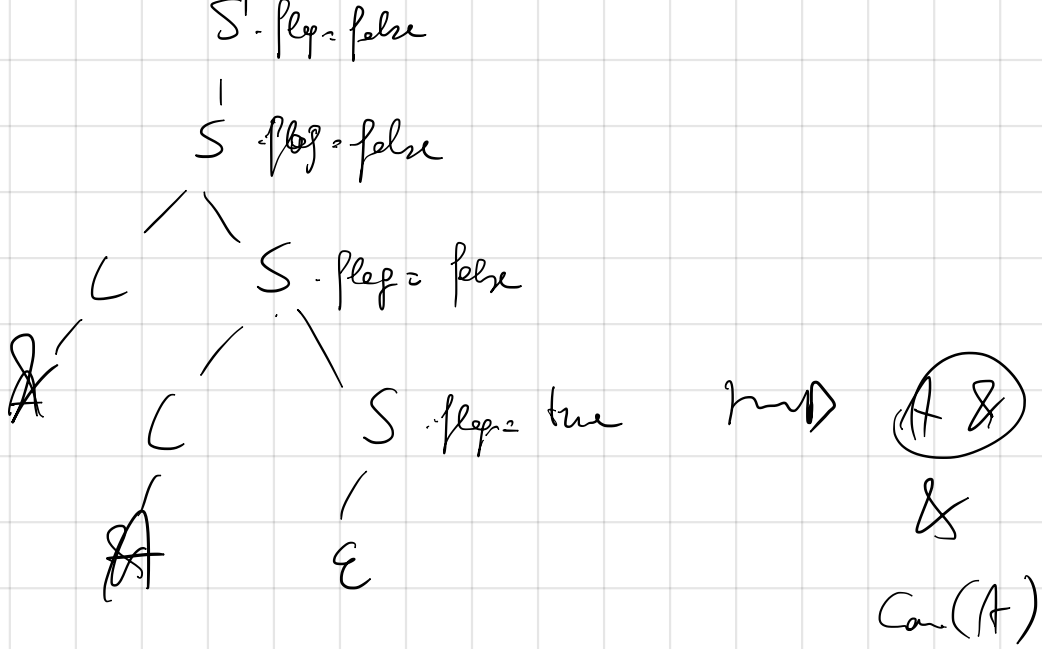
$L \to \&$

$L \to A$

$L \to B$

$L \to C$


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


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

SDD

$S' \rightarrow S$ $\quad S.flag = false;\ S'.val = S.val$

$S \rightarrow L\ S_1$ $\quad S_1.flag = (!S.flag \wedge L.val = '\&');$

$S.val =$ if (S.flag) then
| if (L.val = '&') then
| | '&' . S_1.val
| else 'Cmd(L.val)' . S_1.val
else if (L.val ≠ '&')
then L.val . S_1.val
else S_1.val ;

$S \rightarrow \varepsilon$ $\quad S.val = if(S.flag)$ then '&' else '' ;

$L \rightarrow \&$ $\quad L.val = '\&'$

$L \rightarrow A$ $\quad L.val = 'A'$

$L \rightarrow B$ $\quad L.val = 'B'$

$L \rightarrow C$ $\quad L.val = 'C'$

The SDD is L-attributed

The derived STD parsable TOP-DOWN is:

$S' \rightarrow \{ S.flag = false; \}\ S\ \{ S'.val = S.val \}$

$S \rightarrow L\ \{ S_1.flag = (!S.flag \wedge L.val = '\&'); \}$

$S_1\ \{ S.val = if\ldots\ ; \}$

$S \rightarrow \varepsilon\ \{ S.val = if(S.flag)\ then\ '\&'\ else\ ''; \}$

$L \rightarrow \&\ \{ L.val = '\&' \}$

STD

---

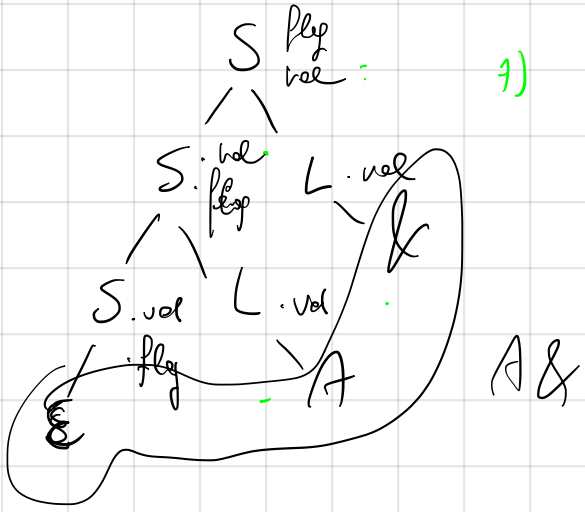
3) Give an SDT that can be parsable BOTTOM-UP.

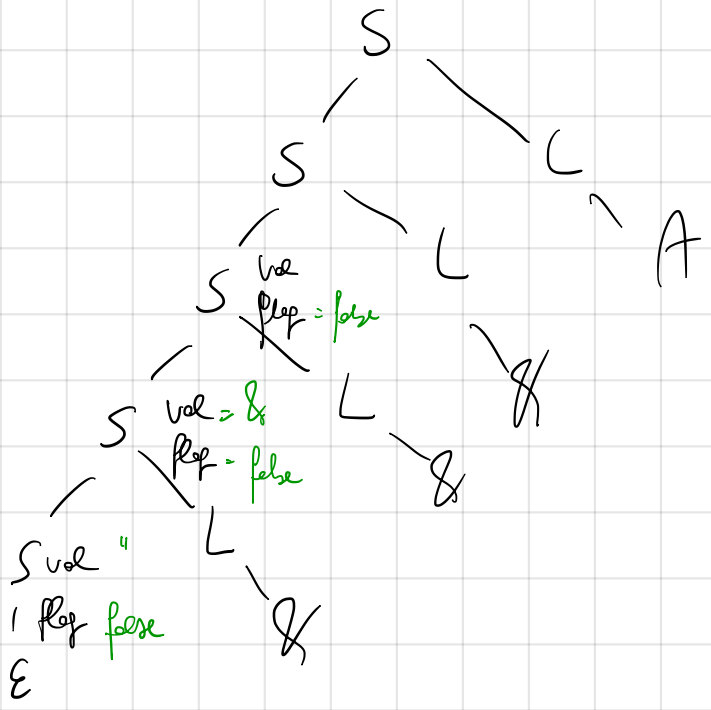
- We can change the grammar, that now needs to be LR(1), and define an associated S-attributed SDD

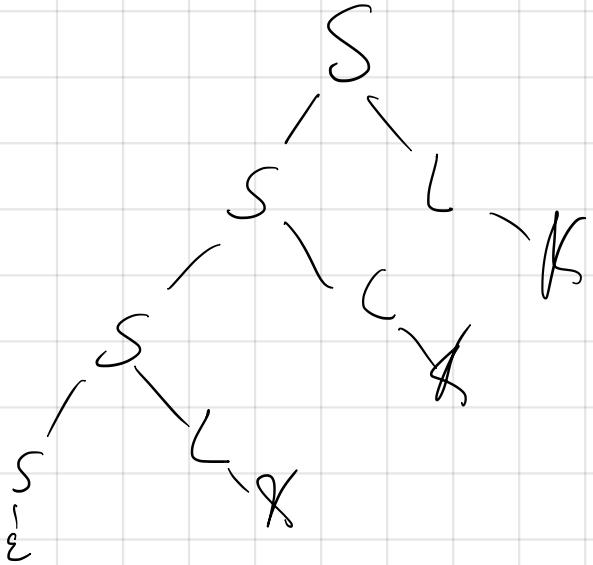
$S' \rightarrow S$

$S \rightarrow S\ L \mid \varepsilon$

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







SDD:

$S' \longrightarrow S$ $\quad S'.val = if(S.flg)$ then $S.val \cdot '\&'$

else $S.val$

$S \longrightarrow S_2 L$ $\quad S.val = if(S_1.flg)$

then if $(L.val = '\&')$

then $S_2.val \cdot '\&'$

else $S_2.val \cdot 'Cmd(L.val)'$

else if $(L.val = '\&')$

then $S_2.val$

else $S_2.val \cdot L.val$;

$S.flg = (!S_2.flg \wedge L.val = '\&')$;

$S \to \varepsilon$ $\quad S.val = ''$; $\quad S.flg = false$;

$L \to \&$ $\quad L.val = '\&'$

$\vdots$ $\quad \vdots$

$L \to C$ $\quad L.val = 'C'$

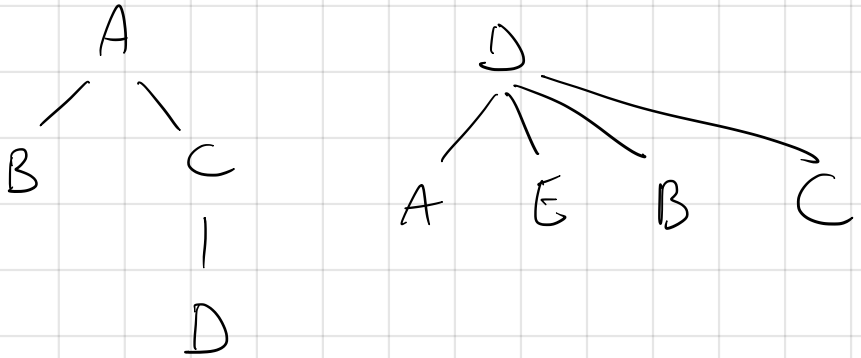
* of the productions and such an SDT can be implemented during BOTTOM-UP parsing

The SDD is S-attributed and the grammar is LR(1). Thus, the corresponding STD is the SDD in which all the actions are put on the right side *

EX Consider the Language of the forests of labelled trees represented as parenthesized lists of trees seperated by commas.

Every tree that is not just a leaf is represented by a pair of elements, seperated by comma and enclosed in spare brackets. The first of these elements is the label of the root and the second is the forest of the trees that are the children of the root.

Let $\Sigma = \{A, B, C, D, E\}$ the alphabet of the labels.



([A, (B, [C, (D)])], [D, (A, E, B, C))

1) Give a grammar for the Language
2) Give an attributed grammar suitable for TOP-DOWN Parsing that calculates for each non-terminal node in the parse tree, the syntax tree derived from that node using the following operations:
   1) MkT: Label x List(Tree) → Tree
   2) MkE: → List(Tree) (creates the empty list)
   3) Add: Tree x List(Tree) → List(Tree)

(S) → (T F S.forest

F → , T F F.forest

F →) F.forest

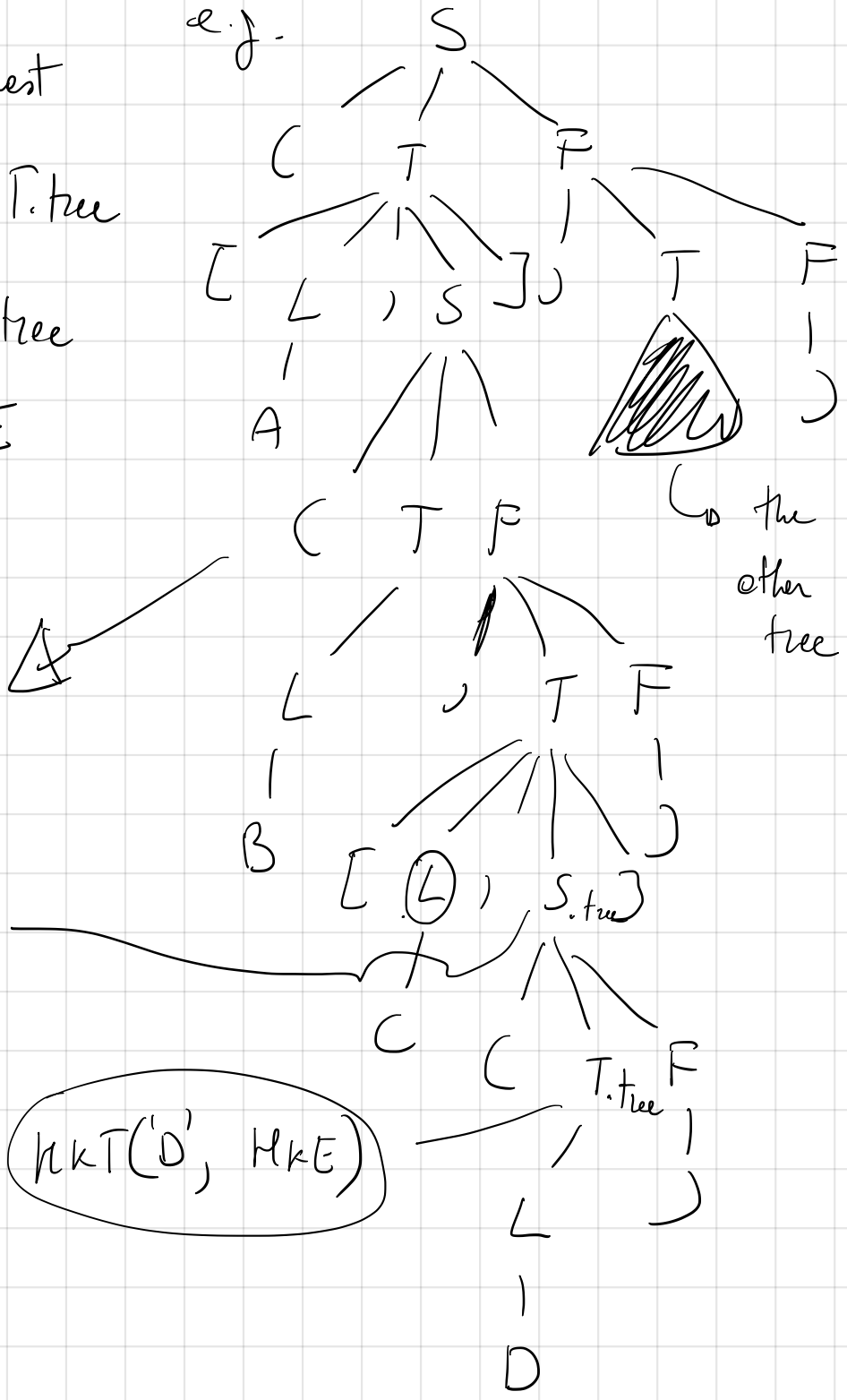
(T) → [L , S] T.tree

(T) → L T.tree

L → A | B | C | D | E

L.lab

The grammar is LL(2)
(to prove for exercise)



Add(T.tree, MKE)

MKT('D', MKE)

$L \to A \mid \dots \mid E$ $\quad L.lab = 'A' \mid \dots \mid 'E'$

$T \to L$ $\quad T.tree = MkT(L.lab, MkE());$

$T \to [L, S]$ $\quad T.tree = MkT(L.lab, S.forest);$

$F \to)$ $\quad F.forest = MkE();$

$F \to , T F_1$ $\quad F.forest = Add(T.tree, F_1.forest);$

$S \to (T F$ $\quad S.forest = Add(T.tree, F.forest);$

---

4) Give an attribute grammar suitable for TOP-DOWN parsing that calculates as an attribute of the start symbol the depth of the forest, that is the maximum height of all the trees in the forest

S, F, T attribute depth Synthesized

$L \to A \mid \dots \mid E$

$T \to L$ $\quad T.depth = 0;$

$T \to [L, S]$ $\quad T.depth = S.depth + 1;$

$F \to)$ $\quad F.depth = -1;$

$F \to , T F_1$ $\quad F.depth = max(T.depth, F_2.depth);$

$S \to (T F$ $\quad S.depth = max(T.depth, F.depth);$