Ex 1

Nullable S is not nullable, but B is.

First

$First(B) = First(\varepsilon) \cup First(\,(\,B\,)\,B\,) = \{\} \cup \{(\} = \{(\}$

$First(S) = First(B\,EOF) = First(B) \cup First(EOF) = \{(\} \cup \{EOF\}$

$= \{(, EOF\}$

(As B is nullable)

Follow

Follow of B:

From first rule: $\{EOF\} \subseteq Follow(B)$

From second rule: $\{)\} \subseteq Follow(B)$ and $Follow(B) \subseteq Follow(B)$

$\Rightarrow Follow(B) = \{EOF,)\}$

Follow of S:

$Follow(S) = \{\}$ as S doesn't appear on the right hand side of any rule.

Table

	(	)	EOF
S	1	/	1
B	2	1	1

(appears in First(B EOF), which is the first (and only) branch of S.

) appears in Follow(B) and the first branch of B is nullable.

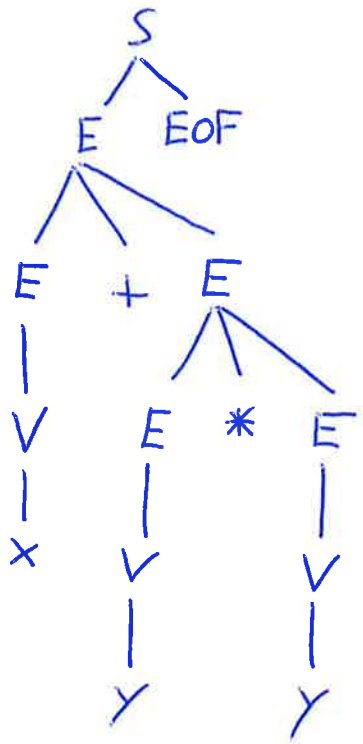
LL(1) check

As the table doesn't contain any cell with more than one applicable branch, the grammar is LL(1).

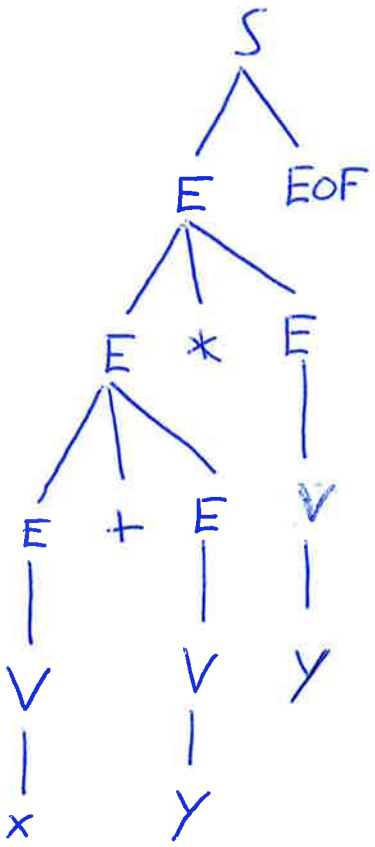
Ex 2

Input string: x+y*y EOF

First parse tree:



Second parse tree:



Ex 3 Question 1

Yes.

Proof

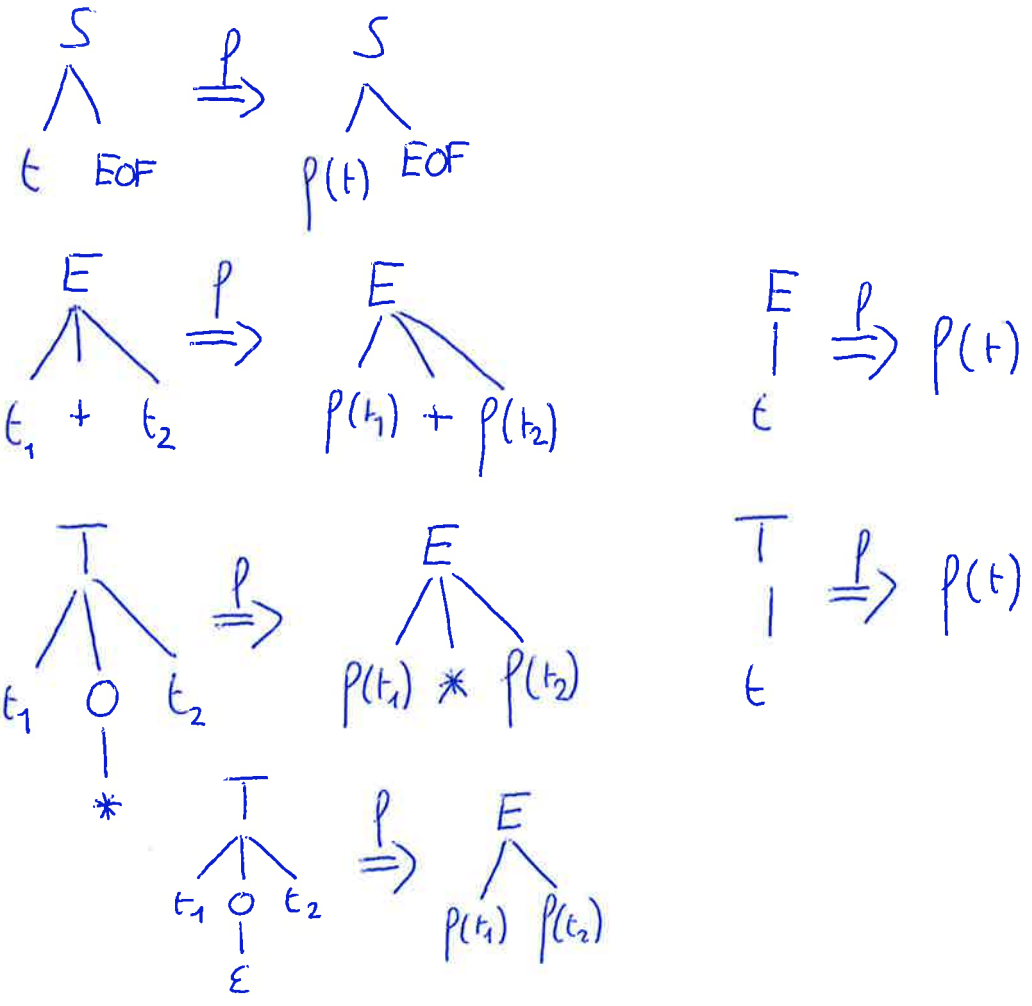
We show that the 2 grammars define the same language by showing how one can transform parse trees from one grammar to parse trees from the other grammar that yield the same word.

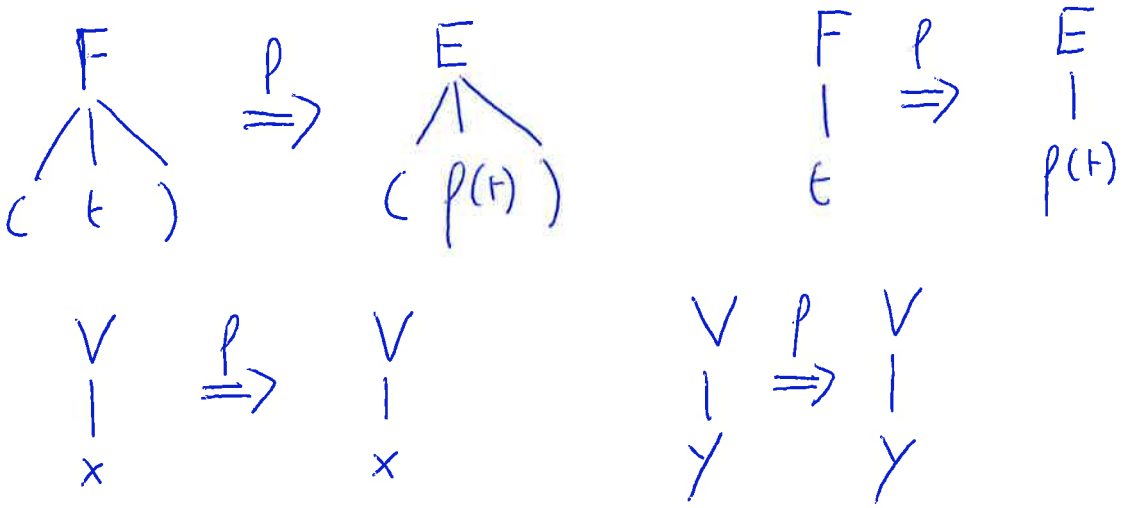
Let the grammar from Ex. 2 be G2 and the grammar from Ex. 3 be G3.

G3 ⇒ G2

We show how to convert parse trees of G3 into parse trees from G2 first, as it is easier.

We define a recursive function ρ that takes (sub)trees of G3 and turn them into valid (sub)trees of G2.

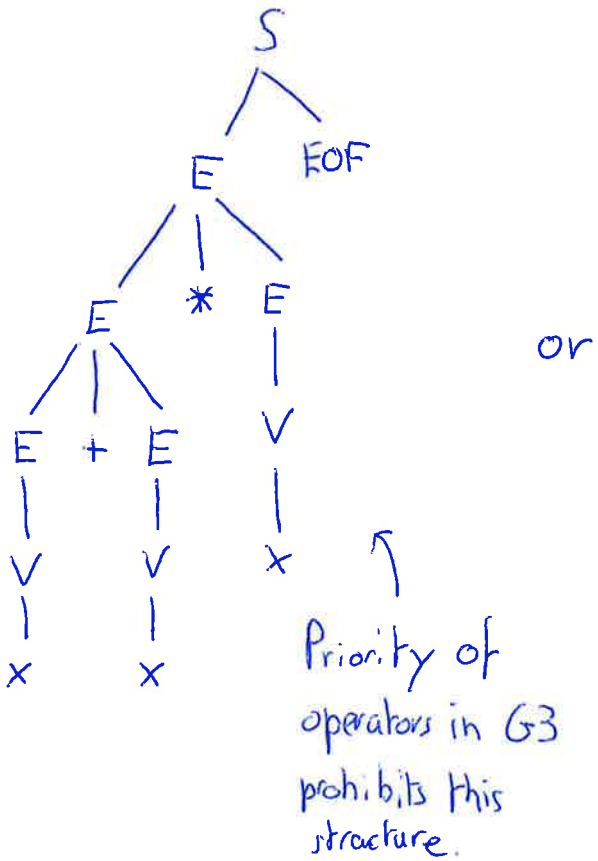




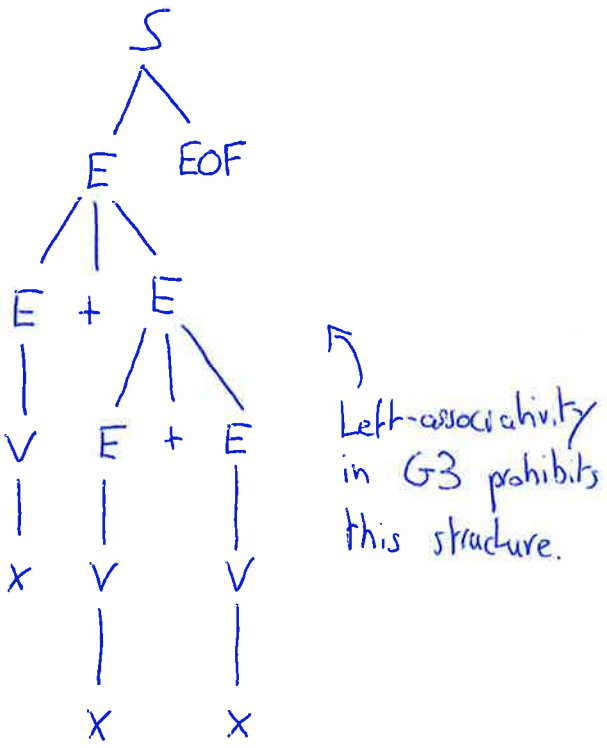
We can easily verify that, for any tree t of G3, the value $\rho(t)$ is a valid parse tree of G2. Moreover, we can easily check that t and $\rho(t)$ yield the same sequence of terminals.

G2 ⇒ G3

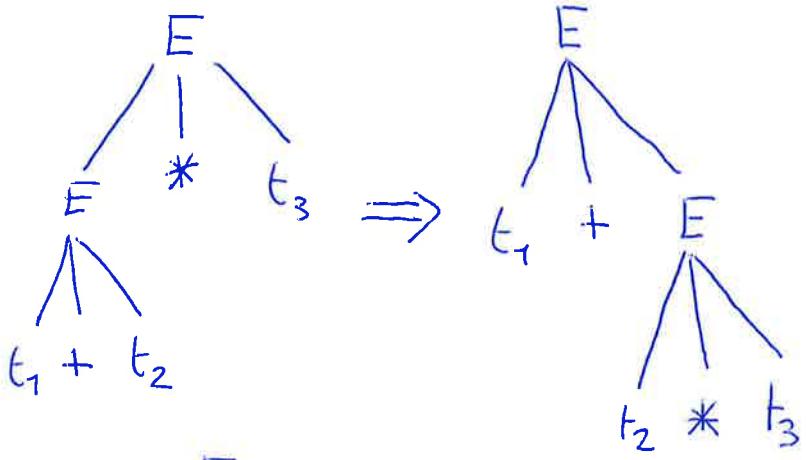
The second direction is trickier. Some parse trees of G2 are not straightforwardly translatable into parse trees of G3. For instance:



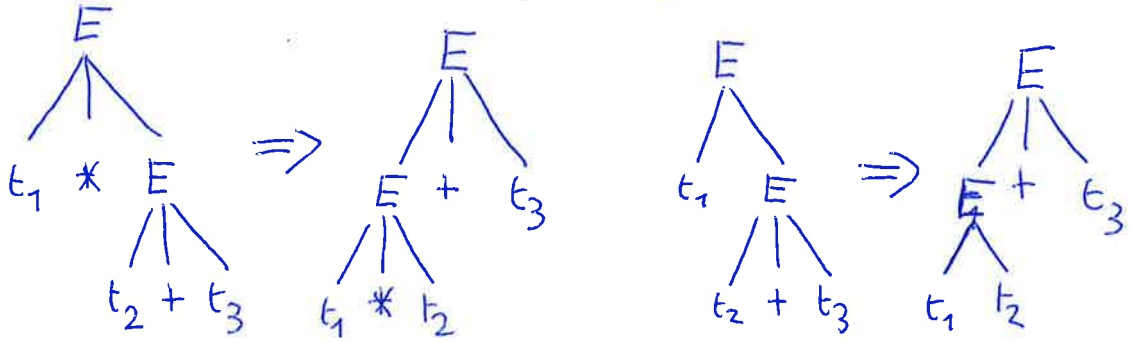
or

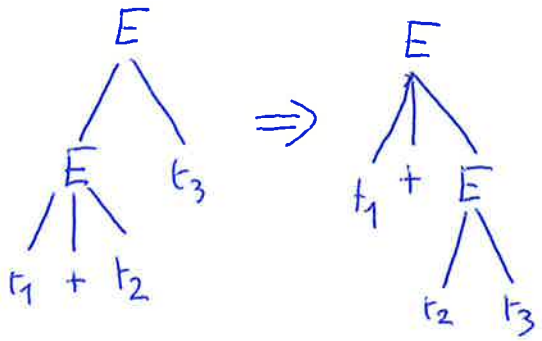


As a first step, we will apply transformations onto problematic trees and subtrees until the structure of the parse tree is compatible with G3. The transformation preserve the yield.

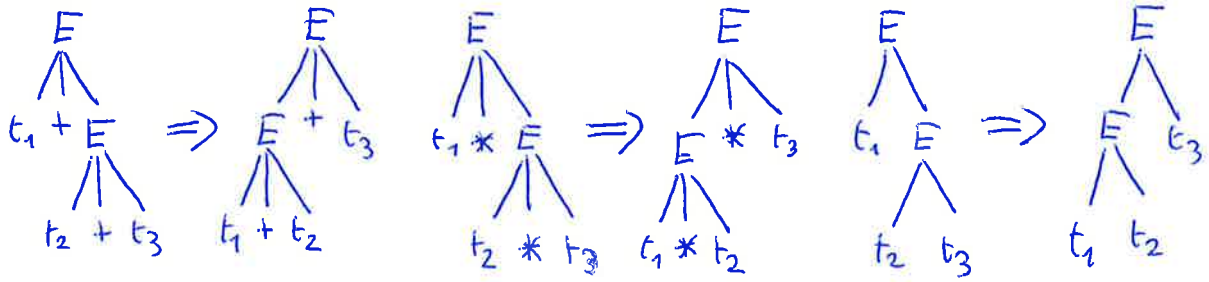


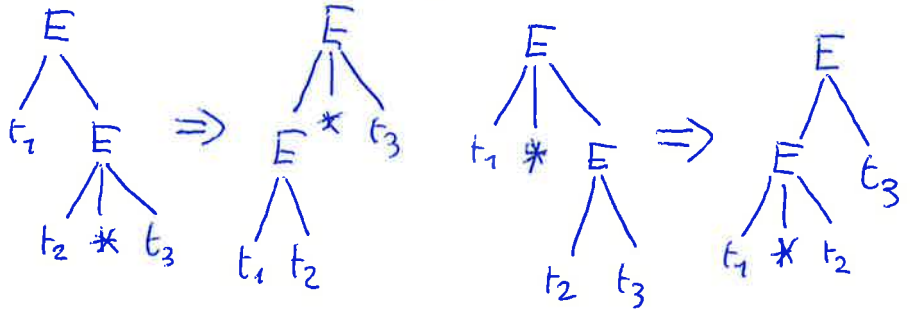
" Fixing priority between * and + "





Those 4 transformations are fixing problems with respect to priority of operators.

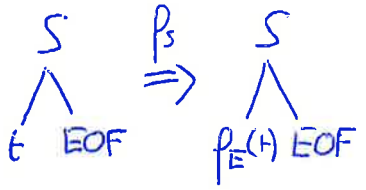




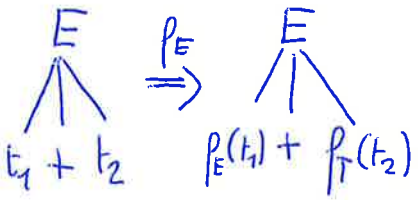
Those 5 transformations are fixing problems with respect to associativity.

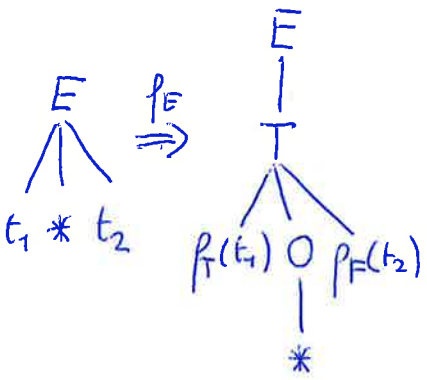
We apply the previously defined transformations iteratively on the parse trees and their subtrees until we reach a fixpoint. Ideally, we would also prove that the procedure always terminates.

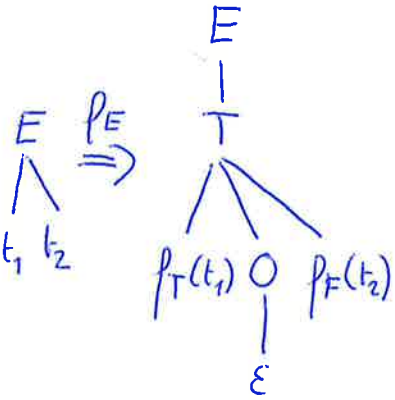
Next, we can apply the recursive function ρ_S (defined in terms of other recursive functions ρ_E, ρ_T, ρ_F and ρ_V) onto (fixed) parse trees t of G2. The resulting value $\rho(t)$ will be a parse tree of G3 that yields the same value as t.

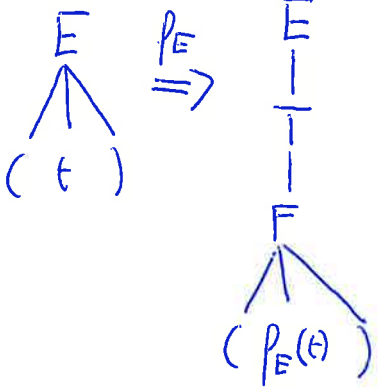


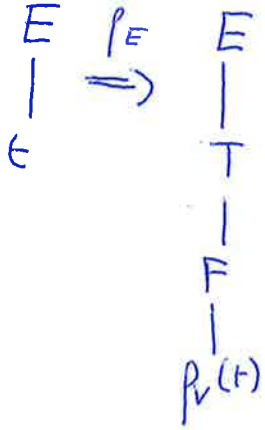
ρ_S converts top level trees



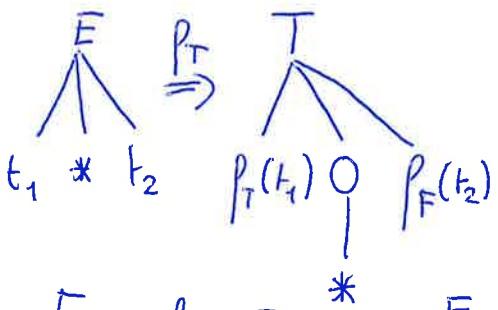


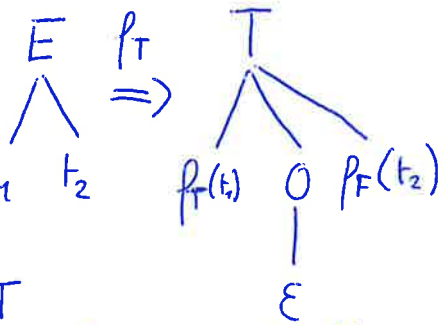


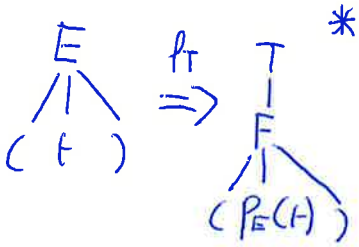


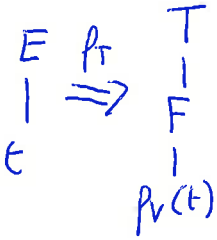


ρ_E converts all subtrees rooted at E into subtrees rooted at E (in G3).

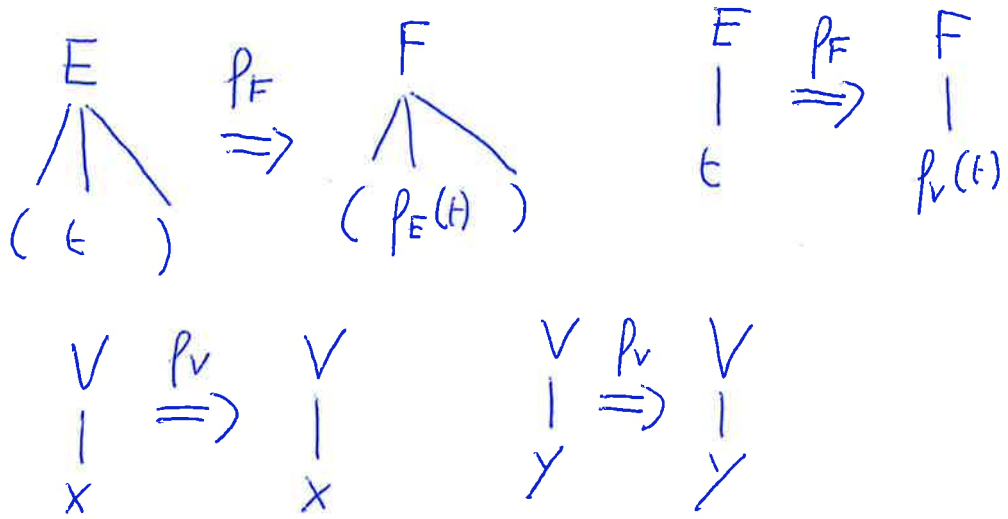








ρ_T converts all subtrees rooted at E, minus the case +, into subtrees rooted at T.



We can easily check that, for any parse tree t of G_2 and its "fixed" parse tree t' of G2, $\rho(t')$ is a valid parse tree of G3, and ϵ and $\rho(t')$ yield the same value.

Note that it is very important that the trees are fixed, as some functions, such as ρ_T or ρ_F, are not defined on some cases. The transformations make sure those cases do not appear.

#

Question 2

No, the grammar is not ambiguous.

We can check this by inspecting the languages defined by the branches of the different rules and showing that they are disjoint for any specific rule.

For instance, in $E \to E+T \mid T$, we know that all words of $E+T$ must have a $+$ outside of parentheses, while it is impossible for words of T to have such a $+$.

Question 3

Only O is nullable.

First

First(V) = {x, y}

First(F) = {(, x, y}

First(O) = {*}

First(T)

We have that

First(T) ⊆ First(T) by first branch

First(F) ⊆ First(T) by second.

⇒ First(T) = {(, x, y}

First(E)

We have that

First(E) ⊆ First(E) by first branch

First(T) ⊆ First(E) by second branch

⇒ First(E) = {(, x, y}

First(S) = First(E) = {(, x, y}

Follow

Follow(S) = {}

Follow(E)

{EOF} ⊆ Follow(E) by first rule

{+} ⊆ Follow(E) by second rule

{)} ⊆ Follow(E) by fifth rule

⇒ Follow(E) = {EOF, +,)}

Follow(T)

Follow(E) ⊆ Follow(T) by second rule

First(O) ⊆ Follow(T) by third rule

First(F) ⊆ Follow(T) by third rule (O nullable).

⇒ Follow(T) = {EOF, +,), *, (, x, y}

Follow (O) = First (F) = { (, x, y }

Follow (F)

Follow (T) ⊆ Follow (F) by third rule

⇒ Follow (F) = { EOF, +,), *, (, x, y }

Follow (V)

Follow (F) ⊆ Follow (V) by fifth rule.

⇒ Follow (V) = { EOF, +,), *, (, x, y }

The grammar is not LL(1).

Proof

Take rule $E \rightarrow E+T \mid T$

We have that First (E+T) = First (E) = { (, x, y }

and First (T) = { (, x, y }.

As the two sets are not disjoint, the grammar can not be LL(1)

Ex 4

Question 1

I suggest the following grammar:

$S \to E \ EOF$
$E \to TE'$
$E' \to +E \mid \varepsilon$
$T \to FT'$
$T' \to *T \mid T \mid \varepsilon$
$F \to (E) \mid V$
$V \to x \mid y$

You can get towards this grammar by eliminating left recursion. Once we compute the parsing table, the fact that the grammar is LL(1) will be clearer.

Question 2

Only E' and T' are nullable.

First

$First(V) = \{x, y\}$ $\quad$ $First(F) = \{(, x, y\}$

$First(T) = First(F) = \{(, x, y\}$ $\quad$ $First(T') = \{*\} \cup First(T) = \{*, (, x, y\}$

$First(E) = First(T) = \{(, x, y\}$ $\quad$ $First(E') = \{+\}$

$First(S) = First(E) = \{(, x, y\}$

Follow

Follow (S) = { }

Follow (E), Follow (E')

{EOF} ⊆ Follow (E) by first rule

Follow (E') ⊆ Follow (E) by third rule

Follow (E) ⊆ Follow (E') by second rule.

{) } ⊆ Follow (E) by sixth rule.

⇒ Follow (E) = Follow (E') = { EOF,) }

Follow (T), Follow (T')

First (E') ⊆ Follow (T) by rule 2.

Follow (E) ⊆ Follow (T) by rule 2 (E' is nullable).

Follow (T) ⊆ Follow (T') by rule 4.

Follow (T') ⊆ Follow (T) by rule 5.

⇒ Follow (T) = Follow (T') = { +, EOF,) }

Follow (F)

First (T') ⊆ Follow (F) by rule 4.

Follow (T) ⊆ Follow (F) by rule 4 (T' is nullable).

⇒ Follow (F) = { *, (, x, y, +, EOF,) }

Follow (V) = Follow (F) = { *, (, x, y, +, EOF,) }

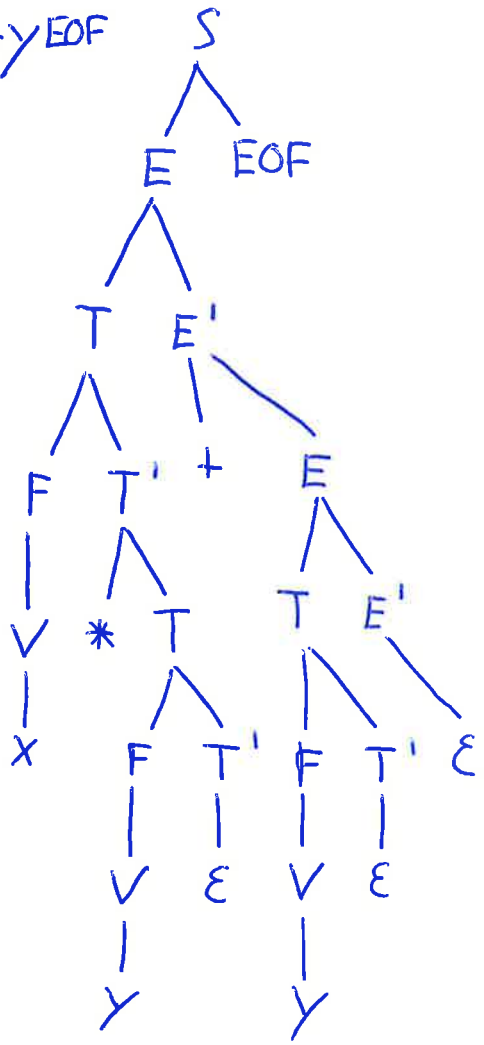
Question 3

	+	*	(	)	x	y	EOF
S			1		1	1	
E			1		1	1	
E'	1			2			2
T			1		1	1	
T'	3	1	2	3	2	2	3
F			1		2	2	
V					1	2	

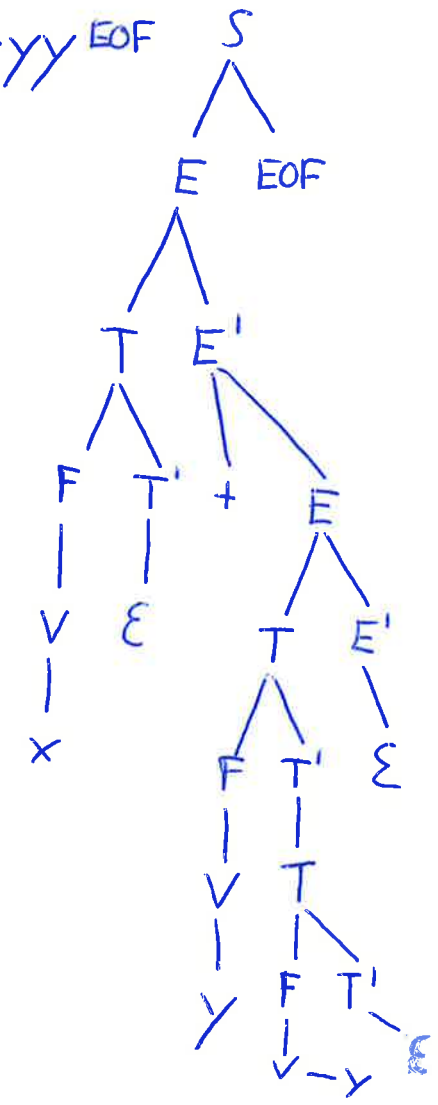
Question 4

You will obtain the following parse trees:

x*y+yEOF



x+yy EOF



(x+y)×EOF