1.

a. terminals: 0, 1, #, €

non-terminals: S, A, B

b. S => AB

=> 0A0B

=> 01A10B

=> 010A010B

=> 010#B010B

=> 010#010B

=> 010#0101B

=> 010#0101

c. S => AB

=> A1B

=> A1

=> 0A01

=> 01A101

=> 010A0101

=> 010#B0101

=> 010#0101

d.

S

/ \

/ \

/ \

/ \

/ \

A B

/|\ |\

/ | \ | \

/ | \ | \

/ | \ | \

/ | \ | \

0 A 0 1 B

/|\ |

/ | \ |

/ | \ |

/ | \ |

1 A 1 €

/|\

/ | \

/ | \

0 A 0

|\

| \

# B

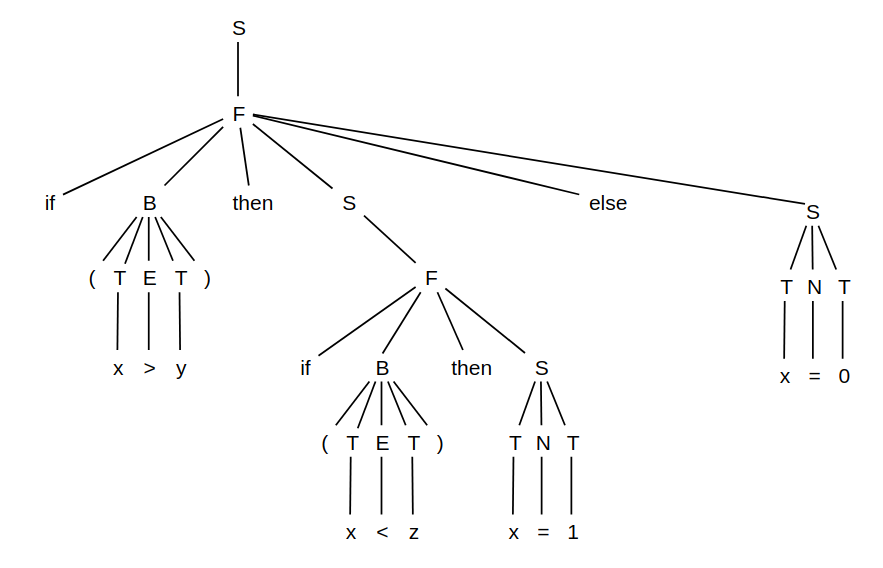
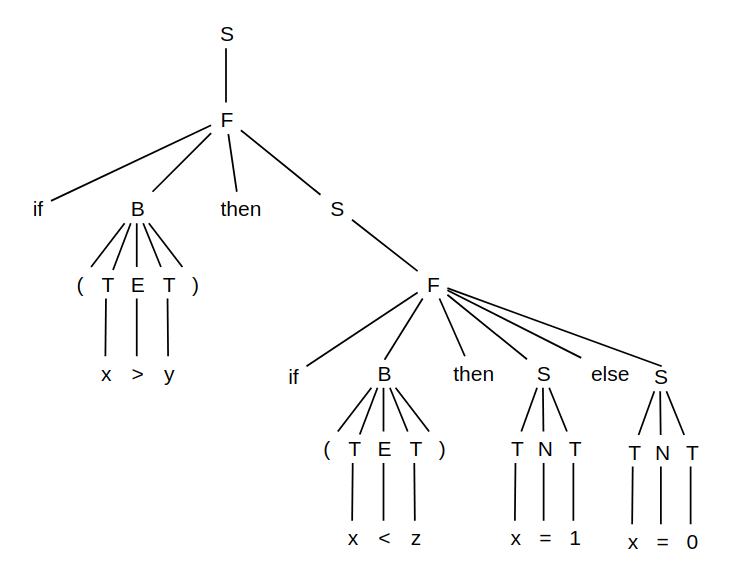
|

€

e. 0#1

1#0

2. a.



b.

old: S → F | T N T

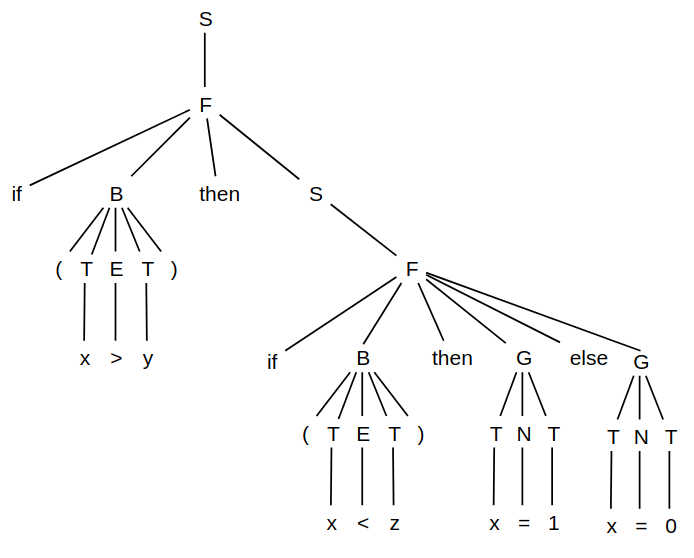
F → if B then S | if B then S else S

new: S → F

G → T N T

F → if B then S | if B then G else G

c.



d. The problem is “S → F | T N T” grammar, by separating this grammar into G and F will solve problem.

3.

a. not: right to left

and: left to right

or: left to right

xor: left to right

b. not > and > xor > or

A xor B = A and not B or not A and B. So “xor“ is not higher precedence than “and” and not lower than “or”.

c.

i. 0 xor 1 and not 0 or 0 xor not 1

ii. E => E or T

=>\* E and T or F xor T

=>\* T and not T or 0 xor not T

=>\* F xor T and not F or 0 xor not F

=>\* 0 xor 1 and not 0 or 0 xor not 1

4.

a. terminals: 0, 1, 23, |, (, ), \*, +

non-terminals: S, A, B

start symbol: S

production rules: S → A | B | €

A → 0S | 1S | 2S | 3S

B → (S | )S | \*S | |S | +S

b. terminals: [, ]

non-terminals: S, A

start symbol: S

production rules: S → A | €

A → [S | ]S

c. terminals: a, b, c

non-terminals: S

start symbol: S

production rules: S → aS | bS | cS | €