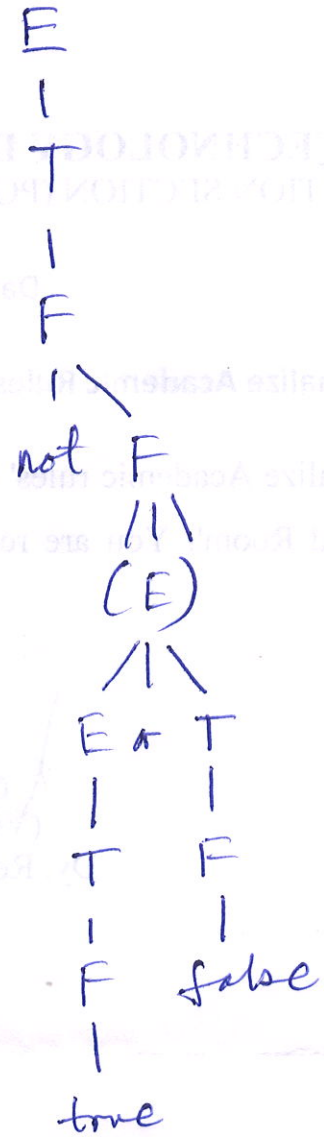
1. (a)

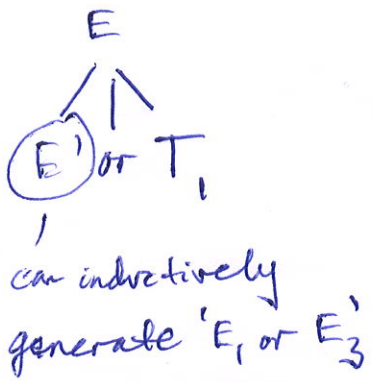


(b) Base cases : 'true' and 'false' – generated by F

Inductive cases :

(I.) E_1 or E_2

If E_2 contains 'or' : There must be a final 'or', or let $E_2 = E_3$ or T_1 (T_1 doesn't contain 'or')



If E_2 doesn't contain 'or' : Can write E_2 directly as T_1

So this takes care of occurrences of 'or'.

(II.) T_1 and T_2

Exactly the same logic as before.

(III.) not F_1

Comes directly from $F \rightarrow$ not F

(IV.) If any of the subexpressions is in parentheses: '(E_1)'

E
|
T
|
F
|
(E_1)

So these inductive rules account for all ways of constructing Boolean expressions from subexpressions.

(c) Remove left recursion:

$E \rightarrow TE'$

$E' \rightarrow$ or $TE' \mid \epsilon$

$T \rightarrow FT'$

$T' \rightarrow$ and $FT' \mid \epsilon$

$F \rightarrow$ not $F \mid (E) \mid$ true $\mid$ false

FIRST (E) = { not, (, true, false }

FIRST (E') = { or, ∈ }

FIRST (T) = { not, (, true, false }

FIRST (T') = { and, ∈ }

FIRST (F) = { not, (, true, false }

FOLLOW (E) = { $,) }

FOLLOW (E') = { $,) }

FOLLOW (T) = { or, $,) }

FOLLOW (T') = { or, $,) }

FOLLOW (F) = { and, or, $,) }

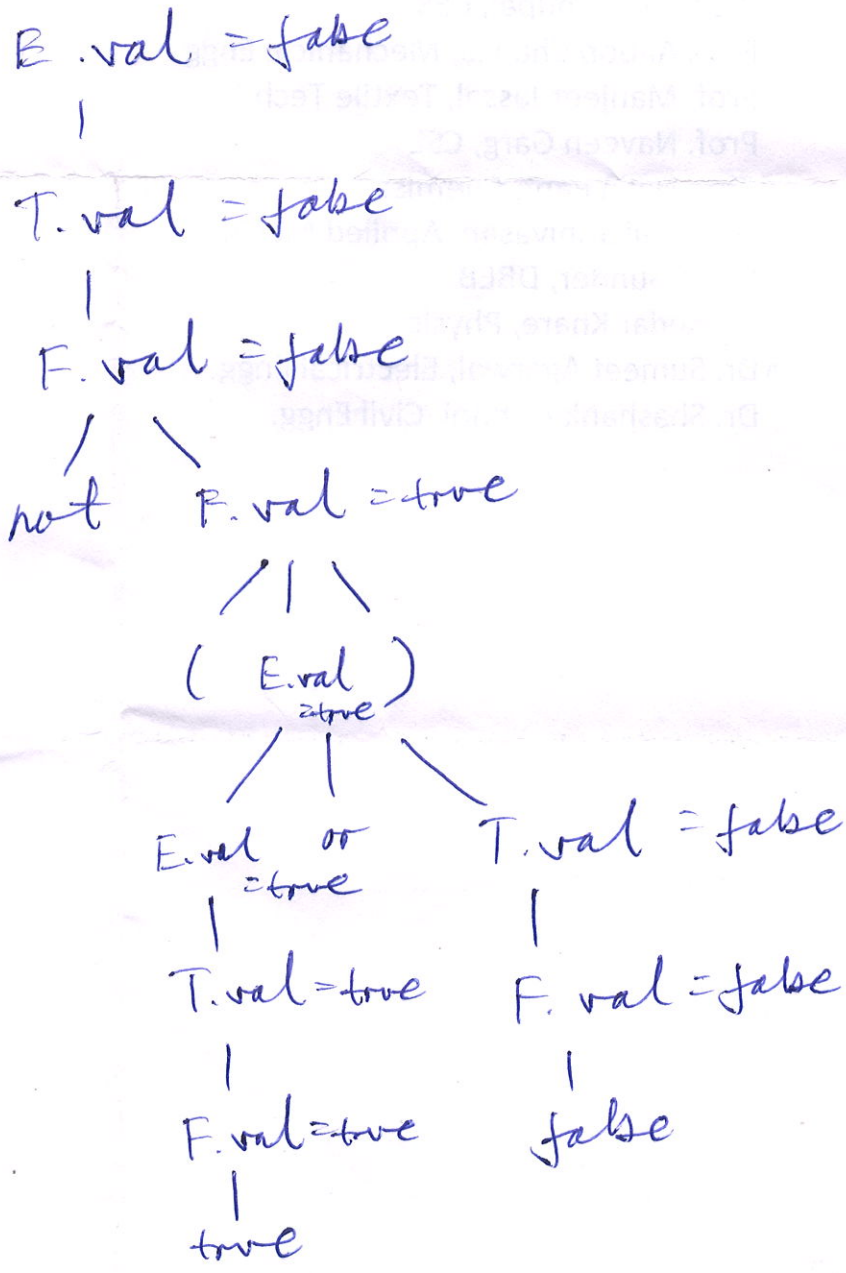
	or	and	not	(	)	true	false	$
E			E→TE'	E→TE'		E→TE'	E→TE'	
E'	E'→orTE'				E'→∈			E'→∈
T			T→FT'	T→FT'		T→FT'	T→FT'	
T'	T'→∈	T'→andFT'			T'→∈			T'→∈
F			F→notF	F→(E)		F→true	F→false	

No multiple entries, so LL(1).

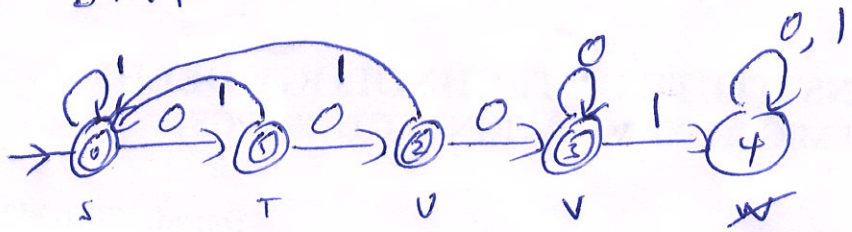
(d) Not ambiguous : Ambiguous grammars can never be predictively parsable, as there will be at least one instance for which multiple parses are valid.

(e)

$E \rightarrow E_1 \text{ or } T$	$E.val := E_1.val \mid\mid T.val$
$E \rightarrow T$	$E.val := T.val$
$T \rightarrow T_1 \text{ and } F$	$T.val := T_1.val \;\&\&\; F.val$
$T \rightarrow F$	$T.val := F.val$
$F \rightarrow \text{not } F_1$	$F.val := !\, F_1.val$
$F \rightarrow (E)$	$F.val := E.val$
$F \rightarrow \text{true}$	$F.val := true$
$F \rightarrow \text{false}$	$F.val := false$



2. (a) DFA:



CFG:

$S \to 1S \mid 0T \mid \epsilon$

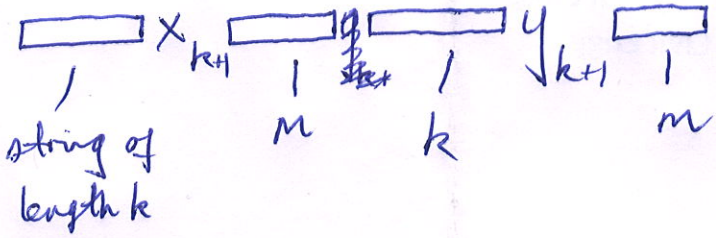
$T \to 0U \mid 1S \mid \epsilon$

$U \to 0V \mid 1S \mid \epsilon$

$V \to 0V \mid \epsilon$

Accepted by DFA, hence regular.

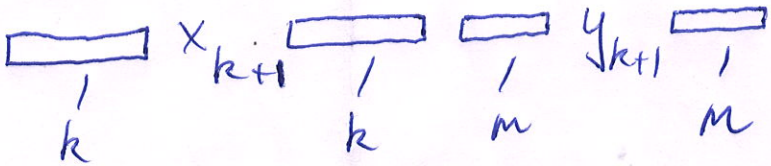
(b) There must be at least one mismatched position. So in general, a string looks like:



where $x_{k+1} \neq y_{k+1}$

k, m are natural numbers (including 0)

Since the strings in the boxes can be anything, we can also break down the above as:



Possible pairwise values of x_{k+1}, y_{k+1}:

x_{k+1}	y_{k+1}
a	b
a	c
b	a
b	c
c	a
c	b

This motivates the following CFG:

$$S \to AB \mid AC \mid BA \mid BC \mid CA \mid CB$$

$$A \to XAX \mid a$$

$$B \to XBX \mid b$$

$$C \to XCX \mid c$$

$$X \to a \mid b \mid c$$

This language is not regular, as the length of the first half has to be known to be able to match the second half, and this can potentially be infinite: so finite memory cannot be sufficient.