CSCC63 TUT 0002
Tutorial 11

Assignment Help & Review

Prove that EQTM is decidable, recognizable, co-recognizable or neither

suppose that M is a TM, L(M) = { x | M accepts x }

EQTM = { <M, N> | L(M) = L(N) } not recognizable

reduce from HALT-c

```
P on input <M, w>:
  define M' on input <x>:
     run M on w
     accept
  define N'
      reject
  return <M', N'> # <M', N'> in EQTM if M' in HALT-c
```

EQTM-c = { <M, N> | L(M) != L(N) } not co-recognizable

reduce from HALT-c

```
P on input <M, w>:
  define M' on input <x>:
     run M on w
     accept
  define N'
      accept
  return <M', N'> # <M', N'> in EQTM-c if M' in HALT-c
```

DECIDABLE = { <M> | L(M) is decidable }
L(M) is decidable if there is a decidable TM N such that L(N) = L(M).
in the last problem we talked about given two TMs, whether we can compare if they are equal.
[given x, y, asking if x = y]
in this problem we are given a TM, asking if there is a decidable TM N such that L(M) = L(N).
[given x, is there y such that x = y]

Look at language forms to see if they are harder than languages you've seen before.

L = { <M> | L(M) = {1} } not recognizable
L-c = { <M> | L(M) != {1} } here either M rejects 1 or M accepts something other than 1. not co-recognizable.

say we have a TM that loops on all.
L(M) = empty != {1}

both can be proven with reduction from HALT-c

```
P on input <M,w>:
  define M' on input <x>:
    accept if x = 1 # if M, w in halt-c then M' accepts only 1
    run M on w
    accept # if M, w in halt, we want M' to accept more than just {1}
```

```
P on input <M,w>:
  define M' on input <x>:
    run M on w
    accept if x = 1 # we want L(M') to be = {1}
```

L = { <G> | G is a CFG such that L(G) contains a palindrome }
L-c = { <G> | G is a CFG such that L(G) contains no palindromes }
recognizable via the following recognizer:
R on input <G>:
  for w in Sigma*
    if w + w^R in G, accept

PCP to L
PCP not co-recog => L not co-recog
Say the input pcp takes in the form of (t1, b1), (t2, b2), ... (tn, bn)
where t_i is the top string of the i-th tile, b_i is the bottom string of the i-th tile

G: S -> t1 A b1^R | t2 A b2^R | ... | tn A bn^R
    A -> # | t1 A b1^R | t2 A b2^R | ... | tn A bn^R

G generates the following string: t1 t3 t5 # b5^R b3^R b1^R

if the PCP has a solution, say tile 1, 3, 5
t1 + t3 + t5 = b1 + b3 + b5, and if such a combination exists then t1 t3 t5 # b5^R b3^R b1^R is a palindrome
if the PCP has no solutions, this means that no concatenation of any t_i's will match the concatination of the
same b_i's if this is the case, then the grammar cannot generate a palindrome since the grammar contains t_is
on one end and b_i^R on the other. this grammar generates s#y, only when an instance of a PCP is a yes instance
can s = y^R.

Q5: given oracle for NAE-3SAT, solve F-NAE-3SAT in polytime.

let NAE-3SAT-ORACLE be the oracle for NAE-3SAT
F on input <phi>:

for each variable:
if setting that variable to true, still yields a phi' for which it has a remaining NAE 3SAT assignment, then in the final variable assignment, this variable can be set to true.
for each variable consider adding a set of clauses to fix variable assignment:
(x_i or a or b) AND ... AND ...

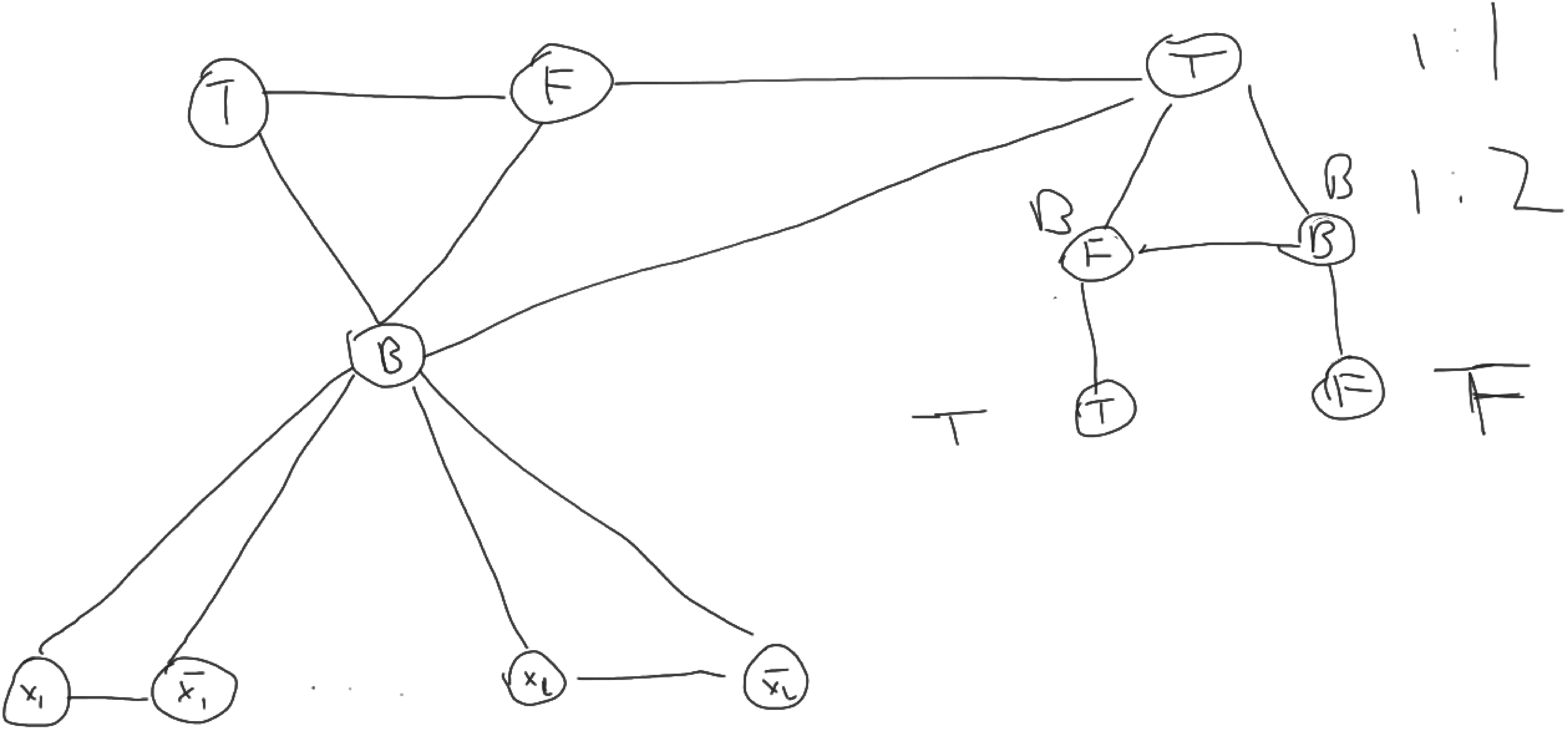
Input: 3CNF phi
Output: a satisfying truth assignment if one exists, otherwise null
F-3CNF is an oracle for 3SAT

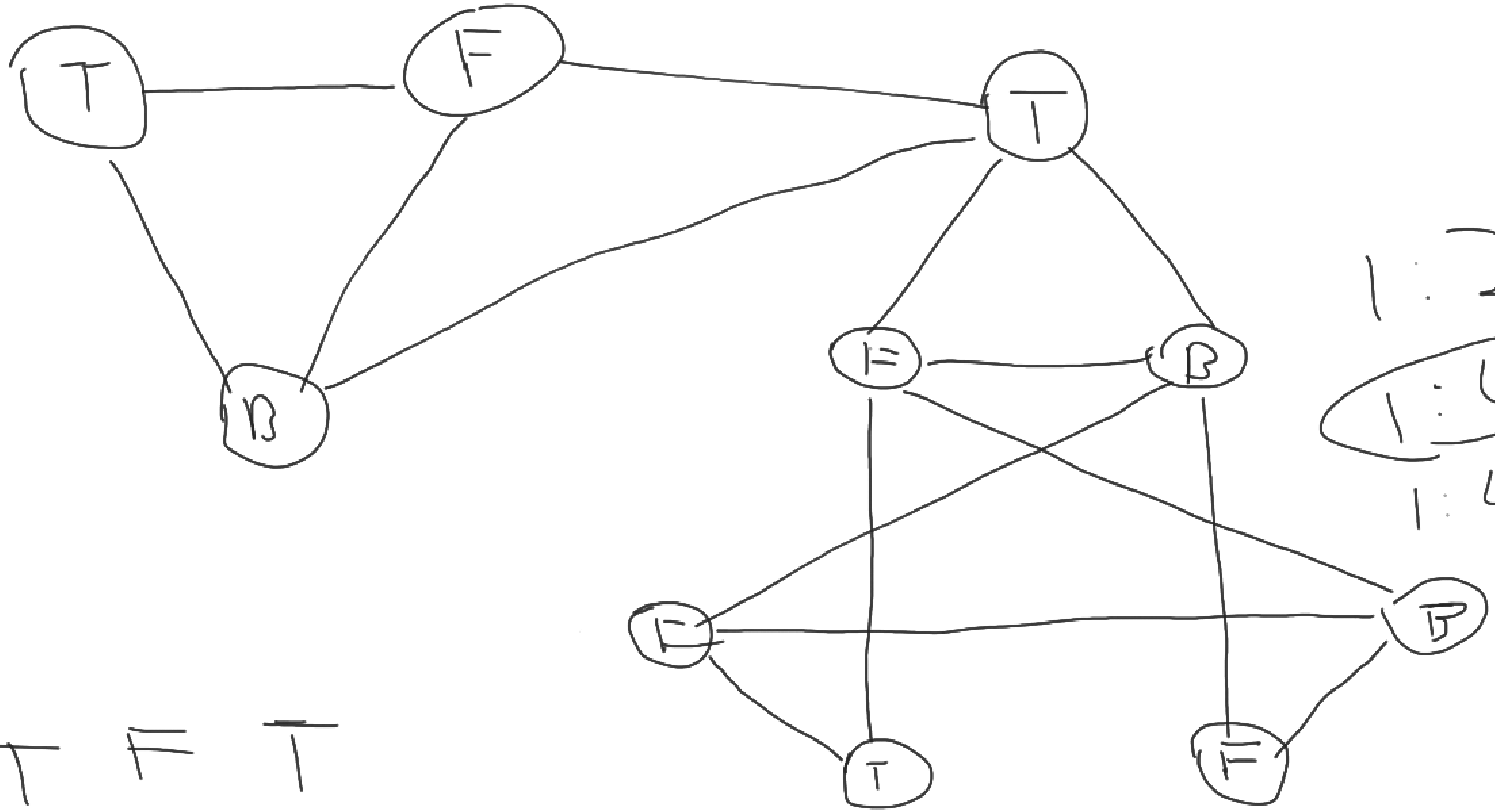
```
M on input <phi>:
  if F-3CNF(phi) returns false, return null

  let phi' = phi, t = [], # where t[i] = the truth assignment of x_i
  for each variable x_i in phi':
    let phi'' = phi' AND (x_i or x_i or x_i)
    run F-3CNF(phi'')
    if returns true, phi' = phi'' and t[i] = true
    otherwise phi' = phi'' AND (not x_i or not x_i or not x_i) and t[i] = false
return t
```

Show that #3COL is #P complete by finding a parsimonious reduction from #3SAT

we have 3 colors, {T, F, B}





1 : 2

1 : 4

1 : 4 · 3!

T F T