Q1

```haskell
data Var = U | V | W | X | Y | Z deriving (Eq, Show)
data Fun = A | B | C | D | E | F | G | H | I | J deriving (Eq, Show)
data Gtree = Gnode Fun [Gtree] | Leafv Var deriving (Eq, Show)
type Term = Gtree


inTree :: Var -> Term -> Bool
inTree a (Leafv b) = a == b
inTree var (Gnode fun xs) = any (inTree var) xs

singlefdouble :: [[a]] -> [a]
singlefdouble = foldl (++) []

unify ::(Term, Term) -> [(Var, Term)]

unify (Leafv a, lf@(Leafv v2)) = (a, lf):[]
unify (Leafv a, gn@(Gnode fun xs)) =
   if a `inTree` gn
   then error "Infinite MGU"
   else (a, gn):[]

unify (gn@(Gnode fun xs), lf@(Leafv v1)) = unify (lf, gn)
unify (Gnode fun1 xs1, Gnode fun2 xs2)
 | fun1 /= fun2 = error "MGU not possible"
 | (length xs1) /= (length xs2) = error "MGU not possible"
 | otherwise = reverse (singlefdouble (map unify (zip xs1 xs2)))
```

Q2

a) **YES,** protocol satisfies mutual exclusion.
Proof by contradiction: Assume mutual exclusion is not satisfied

Observation from protocol code that the following happens before holding any relationships, also assuming that the thread A is the last one thread which writes to turn without any problem

$$write_A(turn=A) \rightarrow write_A(turn=B) \rightarrow \text{Critical Section A},$$
$$write_B(turn=B) \rightarrow write_A(turn=A) \rightarrow \text{Critical Section B}$$

As by assumption thread A is the last thread to write to turn we get:

$$write_B(turn=B) \rightarrow write_A(turn=A)$$

From above we can conclude that A does not complete its outer waiting loop which is contrary to our assumption hence our assumption was wrong , it should satisfy mutual exclusion.

b) No, protocol is not starvation free

Starvation ultimately leads to deadlock, so if it is not deadlock free, it is not protocol free for which we provide a counterexample in the next part.

c) No, protocol is not deadlock free

EG: Sequence leading to deadlock:

$$write_A(turn=A) \rightarrow read_A(busy=false) \rightarrow write_B(turn=B)$$