Implementation of Unification in Haskell

Terms are coded in Haskell as elements of

```
data Term = Var String | Fun String [Term] | MVar String
            deriving Eq
```

A substitution is an element of

```
    type Subst = [(String,Term)]
```

These can be applied to terms and/or composed via

```
    apply :: Subst -> Term -> Term
    apply [] t = t
    apply ((x,u):s) t = apply s (subst u x t)

    (@@) :: Subst -> Subst -> Subst
    s1 @@ s2 = [(x, apply s2 t) | (x,t)<- s1] ++ s2
```

Unifying terms

```
unify :: Term -> Term -> Maybe Subst
unify (MVar x) (MVar y) = Just [(x, MVar y)]

unify (MVar x) t2 = if x ‘elem‘ mvars t2
                    then Nothing
                    else Just [(x,t2)]

unify t1 (MVar y) = if y ‘elem‘ mvars t1
                    then Nothing
                    else Just [(y,t1)]

unify (Fun f ts) (Fun g ss) = if f==g
                              then listUnify ts ss
                              else Nothing
```

Unifying lists

```
listUnify :: [Term] -> [Term] -> Maybe Subst

listUnify []     []     = Just []
listUnify []     (r:rs) = Nothing
listUnify (t:ts) []     = Nothing

listUnify (t:ts) (r:rs)
   = case unify t r of
       Nothing -> Nothing
       Just u1 -> case listUnify (map (apply u1) ts)
                                 (map (apply u1) rs) of
                    Nothing -> Nothing
                    Just u2 -> Just (u1 @@ u2)
```