Unification

In this exercise we implement the basic unification algorithm for Prolog terms. Recall that unification is a procedure that, given two terms tells you what terms to substitute for the variables in the terms, such that the two terms become equal. Such a substitution is called a *unifier*.

For example, a unifier of the terms `f(g(X))` and `f(Y)` is substitutes `g(X)` for `Y`.

Prolog Terms First we need to introduce a data type to represent Prolog terms. Terms are either functor terms of the form `f(t1,...,tn)`, i.e. a functor `f` and zero or more arguments, or a variable `X`. To simplify things, we assume that variables are ordered, and only refer to them by their rank (an `Int`).

```
type VarId = Int
data Term
  = F String [Term]   -- f(t1,...,tn)
  | Var VarId         -- Xn
  deriving (Eq,Ord)
```

`Show` instance for `Term` which prints the corresponding Prolog term has been defined for you. Define a function `occurs n t` which returns true if and only if `Var n` occurs in the term `t`. This function will be useful later on.
For example,

```
> F "f" []
f
> Var 0
X0
> F "f" [F "g" [], F "h" [Var 1]]
f(g,h(X1))
> F "f2" [Var 2]
f2(X2)
```

Substitutions As mentioned previously, the result of a unification of two terms is a unifier, a substitution, such that when it is applied to the two the terms they become equal. We will represent a substitution in Haskell as a list of pairs of a `VarId` and a `Term`.

Define a function `applySubst` which applies a substitution to a term. Remember that a substitution leaves the variables for which it is not defined unchanged, and a substitution applied to a functor term is that same functor term, but with the substitution applied to its arguments.
For example,

```
> applySubst [(0,F "g" [])] (Var 0)
g
> applySubst [(0,F "g" [])] (Var 1)
X1
> applySubst [(0,F "g" [Var 1])] (F "f" [Var 0])
```

```
f(g(X1))
> applySubst [(0,F "g" [Var 1])] (F "f" [F "g" [Var 1]])
f(g(X1))
> applySubst [(0,F "g" []),(1,Var 2)] (F "f" [F "g" [Var 1], Var 0])
f(g(X2),g)
```

Also write a function `conc` that concatenates two substitutions. This is not just the list concatenation, you must *also* apply the substitution on the left to every term in the substitution on the right.
For example,

```
> conc [(0,F "f" [Var 1])] [(1,F "g" [Var 0]),(2,[Var 1])]
[(0,f(X1)),(1,g(f(X1))),(2,X1)]
> conc [] [(1, F "g" [Var 2])]
[(1,g(X2))]
```

Unification Suppose we have a list of equations between terms, we want to obtain a substitution that unifies *all* those equations. This is achieved by the following algorithm:

- If the list is empty, return the empty substitution.
- If the first equation is of the form `X = X`, where `X` is a variable, try to unify the remaining equations.
- If the first equation is of the form `X = t` where `X` is a variable *and* `X` does not occur in `t`, then
  1. Substitute `t` for all occurences of `X` in the remaining equations.
  2. Try to unify the remaining equations.
  3. Concatenate the substitution obtained in the previous step with the substitution `[(X,t)]`.
- If the first equation is of the form `t = X` where `X` is a variable, then swap the position of `X` and `t` and try to unify the equations again.
- If the first equation is of the form `f(t1,...,tn) = g(u1,...,um)`, where $\texttt{f} = \texttt{g}$ *and* $n = m$, then
  1. Add all equations of the form `t1 = u1`, ..., `tn` = `un` to the list of equations.
  2. Try to unify this new set of equations.
- Otherwise, we fail to find a unifier.

In Haskell, we represent the list of equations as a list of pairs of terms. Because the unification algorithm can fail, the result is a `Maybe Substitution`.

Complete the function `unify` which implements the algorithm above.

To unify two terms, we must simply pass the appropriate list to `unify`:

```
unify1 :: Term -> Term -> Maybe Substitution
unify1 t1 t2 = unify [(t1,t2)]
```

Some examples:

```
> unify1 (Var 0) (F "f" [])
Just [(0,f)]
> unify1 (F "f" []) (Var 0)
Just [(0,f)]
> unify1 (F "f" [F "g" [Var 1]]) (F "f" [Var 0])
Just [(0,g(X1))]
> unify1 (Var 0, F "f" []) (Var 0)
Nothing
> unify1 (F "g" [], F "f" []) (Var 0)
Nothing
> unify1 (F "f" [Var 0, Var1], F "f" [Var 0]) (Var 0)
Nothing
> unify [(F "f" [Var 1],Var 0),(F "g" [Var 1],F "g" [F "a" []])]
Just [(1,a),(0,f(a))]
```