Formal Description

Given a constraint set $C$, we define $\text{unify}(C)$ as...

- If $C$ is empty, return the identity solution. $\phi(s) = s$
- Otherwise, let $(s, t) \in C$ and $C' = C \setminus \{(s, t)\}$.

  Delete  If $s = t$ then $\text{unify}(C')$
  Orient  If $t$ is a variable and $s$ is not, $\text{unify}((\{t, s\}) \cup C')$.
  Decompose  If $P$ is a constructor, $s = P(s_1, \ldots, s_n)$ and $t = P(t_1, \ldots, t_n)$ then $\text{unify}(C' \cup \{(s_1, t_1), \ldots, (s_n, t_n)\})$
  Eliminate  If $s$ is a variable, and $s$ does not occur in $t$, substitute $s$ with $t$ in $C'$ to get $C''$. Then let $\phi = \text{unify}(C'')$ and return $\phi[s \mapsto \phi(t)]$.

Examples and Problems

Use the formal description above to unify the following problems. Always select the first pair from the constraints.

$$\{g(\alpha, a) = g(b, \beta), \ h(\gamma, \gamma) = h(f(\alpha), \gamma)\}$$

$$\{f(\alpha, \alpha) = f(\gamma, x), \ h(\beta, g(\gamma)) = h(y, \delta)\}$$

$$\{f(\alpha) = f(x), \ g(\alpha) = g(\beta), \ h(\gamma, x) = h(\beta, \alpha)\}$$
First, review this code with another student. What does it do? How does it work? Write `occurs`, `phi`, and `unify`.

```haskell
module Main where

import Lib

import qualified Data.HashMap.Strict as H
import Data.Maybe (fromJust)
import Data.List (intersperse)

data Entity = Var String
  | Object String [Entity]
deriving (Eq)

instance Show Entity where
  show (Var s) = s
  show (Object s []) = s
  show (Object f xx) = concat $ f $ "," $ (map show xx) ++ ["]"

isVar (Var _) = True
isVar _ = False

-- Environment functions

type Env = H.HashMap String Entity

initial :: Env
initial = H.empty

add :: String -> Entity -> Env -> Env
add x y b = H.insert x y b

contains :: String -> Env -> Bool
contains x b = H.member x b

-- Functions you get to write

phi :: Env -> Entity -> Entity
phi env (Var s) = undefined
phi env (Object s xx) = undefined

occurs :: String -> Entity -> Boolean
occurs = undefined

unify :: [(Entity,Entity)] -> Env
unify [] = initial
unify ((s,t):c') = undefined
```