Purpose

Unification is a core component of many programming language related algorithms. It is important to be able to solve unification problems by hand, as well as to be able to specify to the computer how to solve such a problem.

Your objectives:

- Explain the syntax and usage of $\phi$ as a substitution operator.
- Identify the proper situations for each of the four unification rules and the results.
- Explain the necessity of the occurs-check.
- Implement the unification rules in Haskell.

Part 1 --- $\phi$ Day

Time estimate: 10 minutes.

For the following table, let $\phi = \{ x \mapsto 10, y \mapsto 2 \}$

<table>
<thead>
<tr>
<th>Formula</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi({(x, y)})$</td>
<td>${(10, 2)}$</td>
</tr>
<tr>
<td>$\phi({(a, x), (y, z)})$</td>
<td>${(a, 10), (2, z)}$</td>
</tr>
<tr>
<td>$\phi[x \mapsto z]({(x, y)})$</td>
<td>${(z, 2)}$</td>
</tr>
<tr>
<td>$\phi[z \mapsto 5]({(a, x), (x, z)})$</td>
<td>${(a, 10), (10, 5)}$</td>
</tr>
<tr>
<td>$\phi[z \mapsto 5][y \mapsto 20]({(a, x), (y, z)})$</td>
<td>${(a, 10), (20, 5)}$</td>
</tr>
</tbody>
</table>

Problem 1) As a team, describe the behavior of $\phi$.

- If there is a mapping $x \mapsto y$ in $\phi$, how many times will $x$ be replaced in $\phi$'s argument? **Every occurrence of $x$ will be replaced.**
- If there is a variable $x$ that has no mapping in $\phi$, what happens to the occurrences of $x$ in $\phi$'s argument? **Nothing, it is returned as is.**
- If there is a mapping $x \mapsto y$ in $\phi$, and we call the function $\phi[x \mapsto z]$, on a term $x$, which mapping wins? **The right-most one, $x \mapsto z$.**
**Problem 2)** Now, solve these formulas. Let $\phi = \{ x \mapsto a, y \mapsto b \}$

<table>
<thead>
<tr>
<th>Formula</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi({(x, y)})$</td>
<td>${(a, b)}$</td>
</tr>
<tr>
<td>$\phi({(a, x), (y, z)})$</td>
<td>${(a, a), (b, z)}$</td>
</tr>
<tr>
<td>$\phi[x \mapsto z]({(x, y)})$</td>
<td>${(z, b)}$</td>
</tr>
<tr>
<td>$\phi[z \mapsto x]({(a, x), (y, z)})$</td>
<td>${(a, a), (b, x)}$</td>
</tr>
<tr>
<td>$\phi[z \mapsto x][y \mapsto c]({(a, x), (y, z)})$</td>
<td></td>
</tr>
</tbody>
</table>

Note that only one substitution is made. If $\phi = \{ x \mapsto y, y \mapsto z \}$, then we get $\phi(x) = y$, not $z$. 
Part 2 --- The Rules

Time estimate: 10 minutes

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 \( s \) and \( t \) are identical 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 \( \phi[s \mapsto \phi(t)] \).

### Problem 3

The Eliminate rule rewrites \( \phi \) to \( \phi[s \mapsto \phi(t)] \). Why can't we just rewrite to \( \phi[s \mapsto t] \) instead?

If \( t \) has variables in it, we will want those substituted as well.

### Problem 4

In Haskell, function calls like \( \text{zipWith} \ xx \ yy \) will truncate the longer of \( xx \) and \( yy \) if they are not the same size. The decompose rule doesn’t do this. Why not?

The goal of \( \text{zipWith} \) is to combine data; the goal of unification is to check for compatibility.

### Problem 5

Solve the following unification problem, in the order specified above. Label the rule you use for each step.

\[
\text{unify}(\{ f(\alpha) = f(x), g(\alpha) = g(\beta), h(\gamma, x) = h(\beta, \alpha) \})
\]

\[
= \text{unify}(\{ \alpha = x, g(\alpha) = g(\beta), h(\gamma, x) = h(\beta, \alpha) \})
\]

\[
= \{ \alpha = x \} \cup \text{unify}(\{ g(x) = g(\beta), h(\gamma, x) = h(\beta, x) \})
\]

\[
= \{ \alpha = x \} \cup \text{unify}(\{ x = \beta, h(\gamma, x) = h(\beta, x) \})
\]

\[
= \{ \alpha = x \} \cup \text{unify}(\{ \beta = x, h(\gamma, x) = h(\beta, x) \})
\]

\[
= \{ \alpha = x, \beta = x \} \cup \text{unify}(\{ h(\gamma, x) = h(x, x) \})
\]

\[
= \{ \alpha = x, \beta = x \} \cup \text{unify}(\{ \gamma = x, x = x \})
\]

\[
= \{ \alpha = x, \beta = x, \gamma = x \} \cup \text{unify}(\{ x = x \})
\]

\[
= \{ \alpha = x, \beta = x, \gamma = x \}
\]
Part 3 --- It Never Occurred to Me

**Problem 6)** What happens when we try to solve this?

\[
\text{unify}\{f(\alpha) = f(f(\alpha))\}
\]

We get an infinitely large solution.

**Problem 7)** Consider this HASKELL code. What is its type?

```haskell
\text{foo} \ a = [\text{foo} \ a]
```

We get an infinite type error.
Part 4 --- Show me the Code

This will be put in your repository.