Substitution, `equiv_types`

CS 421

Revision 1.0

1 Change Log

1.0 Initial Release.

2 Objectives

Your objective for this assignment is to understand the details of the substitution.

3 Datatypes for Type Inference

Below is some of the code available for your use in the `Common` module. This module includes the following data types to represent the types of PicoML, which you should recognize from MP3:

```
type typeVar = int

type monoTy = TyVar of typeVar | TyConst of (string * monoTy list)
```

You can use `string_of_monoTy` in `Common` to convert your types into a readable concrete syntax for types.

4 Substitutions

In MP3, one of the things we returned was a substitution. Our substitutions have the type `(typeVar * monoTy) list`. The first component of a pair is the index (or "name") of a type variable. The second is the type that should be substituted for that type variable.

In this ML, you were to implement a function `subst_fun` that will take a substitution and return a substitution *function*, a function that takes a type variable as input and returns the replacement type as given by the substitution. (Recall that we are using the the type `int` for type variables, which we give the synonym `typeVar`.) When creating such a function from a substitution (i.e., a list of pairs as described above), if a given type variable does not have an entry in the list, the identity substitution is assumed for that type variable (i.e. the variable is substituted with itself). For instance, the substitution

```
# let phi = [(5, mk_fun_ty bool_ty (TyVar(2)))];;
val phi : (int * monoTy) list =
  [(5, TyConst ("->", [TyConst ("bool", []); TyVar 2]))]
```

is considered to represent the subsitution function

$$\phi(\tau_i) = \begin{cases} \texttt{bool} \rightarrow \tau_2 & \text{if } i = 5 \\ \tau_i & \text{otherwise} \end{cases}$$

Throughout this ML you may assume that substitutions we work on are always well-structured: there are no two pairs in a substitution list with the same index.

We can also *lift* a substitution to operate on types. A substitution ϕ, when lifted, replaces all the type variables occurring in its input type with the corresponding types. In this ML you were to implement a function `monoTy_lift_subst` for lifting substitutions to generic `monoTy`s.

```
# let rec monoTy_lift_subst sigma = ...
val monoTy_lift_subst : (typeVar * monoTy) list -> monoTy -> monoTy = <fun>
# let lifted_sub = monoTy_lift_subst phi;;
val lifted_sub : monoTy -> monoTy = <fun>
# lifted_sub (TyConst ("->", [TyVar 1; TyVar 5]));;
- : monoTy =
TyConst ("->", [TyVar 1; TyConst ("->", [TyConst ("bool", []); TyVar 2])])
```

1. **Extra Credit (10 pts)** Two types τ_1 and τ_2 are equivalent if there exist two substitutions ϕ_1, ϕ_2 such that $\phi_1(\tau_1) = \tau_2$ and $\phi_2(\tau_2) = \tau_1$. Write a function `equiv_types : monoTy -> monoTy -> bool` to indicate whether the two input type expressions are equivalent.

   Hint: find τ_3 such that τ_1 is equivalent to τ_3 and τ_2 is also equivalent to τ_3 by reducing τ_1 and τ_2 to a canonical form.

   ```
   # let equiv_types ty1 ty2 = ...
   val equiv_types : monoTy -> monoTy -> bool = <fun>
   # equiv_types
       (TyConst ("->", [TyVar 4; TyConst ("->", [TyVar 3; TyVar 4])]))
       (TyConst ("->", [TyVar 3; TyConst ("->", [TyVar 4; TyVar 3])]));;
   - : bool = true
   # equiv_types
       (TyConst ("->", [TyVar 4; TyConst ("->", [TyVar 3; TyVar 4])]))
       (TyConst ("->", [TyVar 4; TyConst ("->", [TyVar 3; TyVar 2])]));;
   - : bool = false
   ```