Session 6: HW4 Quick Start Guidance

hw4_basic

- First of all, try to compile. Comment out the incomplete functions in `unify.ml` and the code snippet under "Main entry point" in `repl.ml`. Insert a `print_string "Hello World!"` into `repl.ml`. Then follow the original hw4 instructions to compile and run, by issuing the following command:

```
make
./repl.o

# if everything is working well, you should see a "Hello World!" printed on the
screen.
```

- Then you can test different functions by calling them separately.
  - To construct the original lambda expression, you can either use the `parse` function provided in `repl.ml`, or construct them using the defined types `Fun` / `App` / `Var`:

```
(* represent a lambda expression of type expr *)
let myexpr0 = Fun ("x", Var "x");;
println_string (to_string myexpr0);;

let myexpr1 = App (Fun ("x", Var "x"), Var "y");;
println_string (to_string myexpr1);;

let myexpr2 = App (Fun ("x", Var "x"), Fun ("y", Var "y"));;
println_string (to_string myexpr2);;
```

  - Then the `annotate` function will assign type variables for the nodes. Before that, we can also use the defined types `AFun` / `AApp` / `AVar` to construct such annotated lambda expressions:

```
(* represent an annotated lambda expression of type aexpr *)
let myaexpr0 = AFun (
  "x",
  AVar ("x", TVar "a"),
  Arrow(TVar "a", TVar "a")
);;
println_string (aexpr_to_string myaexpr0);;
```

```
let myaexpr1 = AApp (
  AFun (
    "x",
    AVar ("x", TVar "a"),
    Arrow (TVar "a", TVar "a")
  ),
  AVar ("y", TVar "b"),
  TVar "c"
);;
println_string (aexpr_to_string myaexpr1);;

let myaexpr2 = AApp (
  AFun (
    "x",
    AVar ("x", TVar "a"),
    Arrow (TVar "a", TVar "a")
  ),
  AFun (
    "y",
    AVar ("y", TVar "b"),
    Arrow (TVar "b", TVar "b")
  ),
  TVar "c"
);;
println_string (aexpr_to_string myaexpr2);;
```

- So we can get the same expression by calling the `annotate` function:

```
(* see how the annotate function works *)
Infer.reset_type_vars();;
let annotate_myexpr0 = Infer.annotate myexpr0;;
println_string (aexpr_to_string annotate_myexpr0);;

Infer.reset_type_vars();;
let annotate_myexpr1 = Infer.annotate myexpr1;;
println_string (aexpr_to_string annotate_myexpr1);;

Infer.reset_type_vars();;
let annotate_myexpr2 = Infer.annotate myexpr2;;
println_string (aexpr_to_string annotate_myexpr2);;
```

- Notice that there's no concrete type (e.g., `int` / `number` / `char` /etc.) in our homework. So every annotated type is a type variable.
- So we can call the `collect` function to see the constraints we needed:

```
(* see how the collect function works *)
let collect_myaexpr0 = Infer.collect [annotate_myexpr0] [];;
println_string "collected constraints 0:";;
println_collect collect_myaexpr0;;

let collect_myaexpr1 = Infer.collect [annotate_myexpr1] [];;
println_string "collected constraints 1:";;
println_collect collect_myaexpr1;;

let collect_myaexpr2 = Infer.collect [annotate_myexpr2] [];;
println_string "collected constraints 2:";;
println_collect collect_myaexpr2;;
```

- The `collect` function generates a constraint list of type `(typ * typ) list`, which will be the direct input to the `unify` function. The `unity` function will generate a `substitution` "function" of type `(id * typ) list`, whose elements of type `(id * typ)` define substitution rules to apply. For example, `("x", Arrow(TVar "y", TVar "z"))` indicates replacing the type variable `x` with the term `Arrow(TVar "y", TVar "z")`.
- And eventually based on the test cases (see hw4 instructions), you are supposed to output the type of root node (if you represent the lambda expression with an abstract syntax tree):

```
let test1 = "fun x -> x";;
(* fun x -> x : 'a -> 'a *)

let test2 = "fun x -> fun y -> fun z -> x z (y z)";;
(* ('c -> 'e -> 'd) -> ('c -> 'e) -> 'c -> 'd *)

let test3 = "fun x -> fun y -> x";;
(* 'a -> 'b -> 'a *)

let test4 = "fun f -> fun g -> fun x -> f (g x)";;
(* ('e -> 'd) -> ('c -> 'e) -> 'c -> 'd *)

let test5 = "fun f -> (fun x -> f x x) (fun y -> f y y)";;
(* This contains circular dependence. *)
```

hw4_bonus

This is similar to hw4_basic.

Notes

- The above testing code is already included in the latest version of `repl.ml`. Check it out.
- hw4 instruction page: https://github.com/fredfeng/CS162/blob/master/homework/hw4/hw4.md

- More notes on hw4: Piazza@147