CSE 114A lecture 19: recap and review!

What did we do?

- programs that operate on programs. in particular:
  - ★ - interpreters
  - ★ - type inferencers
    - Rust
    - C++
    - Haskell
    - ...
  - more generally, static analysis of programs
  - fuzzers / debuggers

ASTs!

what to take away from this class?

- understand that there are many flavors of programs that operate on programs e.g. interpreters, type inferencers.
- understand that it's possible to statically analyze programs to learn a lot about what they do (without running them!) (type inference is an example)
- understand when it's useful to be able to statically analyze programs instead of running them.

Exam review

questions 17–20 from Spring '22

17) $a \to b$, $Int \to c$

$[(c, b), (Int, a)]$

$[(a, Int), (b, Bool \to Bool), (c, Bool \to Bool)]$

Remember:
- a substitution maps type variables to types
- unification is coming up with a substitution that can be applied to two types to make them equal.

18)

$Int \to (Int \to a)$ $\quad$ $Int \to (Int \to Int)$

$b \to c$ $\quad$ $b \to (Int \to Int)$

$[(b, Int \to Int)\ (c, Int)]$ ✗

$[(b, Int)\ (c, Int \to a)]$ ✓

$[(a, Int)\ (c, Int \to Int)]$ ✗

19)

$Int \to a$

$b \to String$

$[(b, Int), (a, String)]$

20) a $\quad$ $b \to Bool \to a$

$b \to Bool \to a$ $\quad$ $b \to Bool \to (b \to Bool \to a)$

↑ since a occurs here, this type can't unify with a.

$[(a, b \to Bool \to a)]$ ✗

These types don't unify. (We cannot come up with a substitution that would make them equal.)

$a \to Int$

$Int \to a$

$[(a, Int)]$

16 a) lookupInEnv id env

16 b) What do we do with a closure when evaluating a function application

$(\backslash cId \to cExpr)\ e_2 \Rightarrow_b$ cExpr but with e_2 plugged in for cId

Remember the beta rule!

```
let v2 = eval env e2 in
    eval (extendEnv cId v2 cEnv) cExpr
```

env in the solutions doc.

↑ all this is the same as what was in the solutions but written in a different way.

13) a)

```
let a = 1 in
let b = 2 in
let f = \x y -> x + y + a + b in
let a = 3 in
let b = 4 in
f a b
```

in the closure for f we have an environment [(a,1), (b,2)]

f a b

f 3 4

(\x y -> x + y + 1 + 2) 3 4

3 + 4 + 1 + 2

10

13 b)

```
let a = 1 in
let b = 2 in
let f = \x y -> x + y + a + b in
let a = 3 in
let b = 4 in
f a b
```

under dynamic scope, oh, no, no environment was saved!

f a b

f 3 4

(\x y -> x + y + 3 + 4) 3 4

3 + 4 + 3 + 4

14

Another problem:

```
let a = 1 in
let f = \x y -> x + y + a + b in
let a = 3 in
let b = 4 in
f a b
```

under static scope: error

under dynamic scope: 14

21)

g) must be T-App since $(\backslash x \to x)\ 3$ is an application. (also, T-App is the only one with two premises.)

e) We know it's an arrow type, $T_1 \to T_2$, and other constraints let us figure out that both T_1 and T_2 are Int.