Lecture 21: Simply-Typed Lambda Calculus

CS4400 Programming Languages

Syntax extensions:

- Note that abstractions need to specify the type of the bound variable – there is no way for the type-checker to guess it (at this stage)

```
data Expr = ...
          | Lam Variable Type Expr
          | App Expr Expr

data Type = ...
          | TyArrow Type Type
```

`TyArrow`:

- The new *type constructor*, `TyArrow`, represents a function type:

  `TyArrow TyInt TyBool` is the type a function that takes an integer (`TyInt`) and returns a boolean (`TyBool`). In Haskell (also in some other languages and in type theory), this is written `Integer -> Bool`

  `TyArrow (TyArrow TyInt TyBool) (TyArrow TyInt TyBool)` corresponds to `(Integer -> Bool) -> (Integer -> Bool)`, that is, the type of a function that takes a function from integers to booleans and returns a function from integers to booleans.

  > Due to currying, we normally understand this as a function that takes a function from integers to booleans, then an integer and returns a boolean. Note that this also means that the arrow -> is *right-associative* and the above Haskell type can be equivalently written as `(Integer -> Bool) -> Integer -> Bool`. Also note, that this is opposite of how application associates, which is to the left.

Note on associativity:

> Function *type* – RIGHT: `t1 -> t2 -> t3 -> t4` is the same as `t1 -> (t2 -> t3 -> t3)` is the same as `t1 -> (t2 -> (t3 -> t4))`
>
> Function *application* – LEFT: `f a b c` is the same as `(f a) b c` is the same as `((f a) b) c`

Rules

```
 add x t1 tenv |- e : t2
------------------------------------
 tenv |- Lam x t1 e : TyArrow t1 t2
```

```
 tenv |- e1 : TyArrow t2 t1    e2 : t2'   t2 == t2'
----------------------------------------------------
             tenv |- App e1 e2 : t1
```

The fixpoint operator:

- No fixpoint combinator (e.g., Y or Z) can be type-checked in STLC, so it has to be added as a primitive operation

```
data Expr = ...
          | Fix Expr
```

```
 tenv |- e : TyArrow t t'   t == t'
------------------------------------
          tenv |- Fix e : t
```