Homework 11

CS3490: Programming Languages

Due: Wednesday 2021-11-16, 10:00PM

1. Typing in $\lambda_{\to}$

Recall the rules of the simply typed lambda calculus, $\lambda_{\to}$.

Syntax.

$$\begin{array}{rcl} \mathbb{T} & ::= & o \mid \mathbb{T} \to \mathbb{T} \\ \Lambda & ::= & \mathbb{V} \mid \Lambda\Lambda \mid \lambda\mathbb{V}.\Lambda \end{array}$$

Typing.

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x : A}\text{ Var} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\text{ Abs}$$

$$\frac{\Gamma \vdash s : A \to B \qquad \Gamma \vdash t : A}{\Gamma \vdash st : B}\text{ App}$$

Reduction.

$$(\lambda x.s)t \quad = \quad [t/x]s$$

For each term t below, determine whether or not t is typable in the empty context. That is, determine whether, for some type $A \in \mathbb{T}$, one can derive $\emptyset \vdash t : A$.

If such a type exists, give the full derivation of $\emptyset \vdash t : A$ using the typing rules.

If no such A exists, explain why.

1. $\lambda x.xy$
2. $\lambda x.x(\lambda yz.z)$
3. $\lambda x.xx$
4. $\lambda nfz.nf(fz)$

2. Programming in λT

Refer to the definition of the type system λT provided in the handout.

NOTATION. For $n \geq 0$, let $\underline{n} \in \Lambda$ denote the term

$$\underline{n} = S^n(0) = \underbrace{S(S(\cdots S(0)))}_{n\ S}$$

2.1. Exponential

2.1.1.

Find a term $\mathsf{exp} \in \Lambda$ that behaves like the exponential.

$$\mathsf{exp}\,\underline{n}\,\underline{m} = \underline{n^m}$$

You may employ the definitions of addition and multiplication done in class.

Hint. First program the exponential function in Haskell, using `recNat`.

(You should call it `expt`, since `exp` is already taken.)

Once done, translate it into the syntax of λT.

```
data Nat = Zero | Succ Nat
  deriving Show

recNat :: a -> (Nat -> a -> a) -> Nat -> a
recNat z f Zero     = z
recNat z f (Succ n) = f n (recNat z f n)

add :: Nat -> Nat -> Nat
add x y = recNat y (const Succ) x

mul :: Nat -> Nat -> Nat
mul x y = recNat Zero (const (add y)) x
```

To help debug your program, you can use the following functions to translate between `Nat` and `Integer`.

```
nat2int :: Nat -> Integer
nat2int Zero     = 0
nat2int (Succ n) = 1 + nat2int n

int2nat :: Integer -> Nat
int2nat x | x <= 0 = Zero
int2nat x          = Succ (int2nat (x-1))

testfun :: (Nat -> Nat -> Nat) -> Integer -> Integer -> Integer
testfun f x y = nat2int (f (int2nat x) (int2nat y))

testAdd = testfun add 5 8
```

2.1.2.

Confirm that your term works correctly by using the reduction rules of λT to reduce the following term until no more reductions are possible:

$$\mathsf{exp}\, S(S(S(0)))\, S(0)$$

You should get $\underline{3} = S(S(S(0)))$ as the answer.
Write out the trace of the reduction above.

2.1.3.

Confirm that your term has the correct type by using the typing rules to derive the following type judgment:

$$\emptyset \vdash \mathsf{exp} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$$

Write out the full derivation tree for this judgment.

2.2. Factorial

Repeat the previous steps for the factorial function.

2.2.1.

Find a term $\mathsf{fact} \in \Lambda$ which behaves like factorial:

$$\mathsf{fact}\, \underline{n} = \underline{n!} = \underline{1 \cdot 2 \cdot \cdots \cdot (n-1) \cdot n}$$

2.2.2.

Confirm that your term works correctly by using the reduction rules to reduce the following term until no more reductions are possible:

$$\mathsf{fact}\, S(S(S(0)))$$

You should get $\underline{6} = S(S(S(S(S(S(0))))))$ as the answer.
Write out the trace of the reduction above.

2.2.3.

Confirm that your term has the correct type by using the typing rules to derive the following type judgment:

$$\emptyset \vdash \mathsf{fact} : \mathbb{N} \to \mathbb{N}$$

Write out the full derivation tree for this judgment.