Principles of Programming languages

Assignment 4 – Theoretical part

By:
David Shmailov 311328322
Aviram Lachmani 316608819

1. Perform the typing inference algorithm for the expression:

$$((lambda\ (x_1, y_1)(if\ (>\ x_1 y_1)\#t\ \#f))8\ 3)$$

Step 1: Rename bound variables:

$$((lambda\ (x, y)(if\ (>\ x\ y)\#t\ \#f))8\ 3)$$

Step2: Assign type variables for every sub expression:

Expression	Variable
$((lambda\ (x, y)(if\ (>\ x\ y)\#t\ \#f))8\ 3)$	T0
$(lambda\ (x, y)(if\ (>\ x\ y)\#t\ \#f))$	T1
$(if\ (>\ x\ y)\ \#t\ \#f)$	T2
$(>\ x\ y)$	T3
$>$	T>
x	Tx
y	Ty
$\#t$	T#t
$\#f$	T#f
8	Tnum8
3	Tnum3

Step 3: The equations for the sub-expressions are:

Expression	Equation
$((lambda\ (x, y)(if\ (>\ x\ y)\#t\ \#f))8\ 3)$	T1=[Tnum8 *Tnum3-> T0]
$(lambda\ (x, y)(if\ (>\ x\ y)\#t\ \#f))$	T1=[Tx -> T2]
$(if\ (>\ x\ y)\ \#t\ \#f)$	T2= T#t
$(if\ (>\ x\ y)\ \#t\ \#f)$	T#t= T#f
$(>\ x\ y)$	T> =[Tx * Ty -> T3]
$>$	T>= [Number * Number-> Boolean]
$\#t$	T#t= Boolean
$\#f$	T#f= Boolean
8	Tnum8= Number
3	Tnum3= Number

Step 4: Then Solve the equations

Equation	Substitution
T1=[Tnum8 *Tnum3-> T0]	
T1=[Tx -> T2]	
T2= T#t	
T#t= T#f	
T> =[Tx * Ty -> T3]	
T>= [Number * Number-> Boolean]	
T#t= Boolean	
T#f= Boolean	
Tnum8= Number	
Tnum3= Number	

After a few step we get:

Equation	Substitution
	T1=[Number * Number -> Boolean]
	T2= Boolean
	T#t= Boolean
	T#f= Boolean
	T> =[Number * Number -> Boolean]
	Tnum8= Number
	Tnum3= Number
	Tx= Number
	Ty= Number
	T0= Boolean

And we will return the answer T0= Boolean.

2. a. assuming that this is the typing statement: {f:[T1->T2], x: T1} |- (f x): T2 (there seemed to be a typo of this sort (f x)}: T2) then under the environment that x is of T1, and f receives a T1 and returns T2, then f(x) is of type T2, so the statement is True.
   If this was not a typo, then the statement is false since there is no expression.

   b. again, assuming that there is a typo and the correct typing statement is:
   {f:[T1->T2],g: [T2->T3], x: T2}|- (f g x): T3, then g(x) receives x of type T2 and outputs T3, but f receives T1, so it cannot receive g as an input, so the typing statement is False.

c. {f:[T2->T1],g: [T1->T2], x: T1}|- (f (g x)): T1, this statement is True,
because it goes $x: T1 \rightarrow g: T2 \rightarrow f: T1$, g can receive T1 as input and f receives T2 as input so everything is legal. So the final type is T1, makes the statement true.

d. {f:[T2->Number],, x: Number}|- (f x x): Number , this statement is false. Because we do not know anything about T2, but we activate f on (number*number) and we cannot confirm T2 is of this type. We also cannot confirm T2 is of type number, so this statement is False.

3. The type of cons in scheme is: $(T1 * T2) \rightarrow T3$, since it receives two arguments of T1 and T2, and returns a list of these arguments, so it also returns a vector type of T1*T2 but in the form of a list which we consider T3. The list type is generic because it depends on T1 and T2.

   The type of car in scheme is: T1 -> T2, when T1 is a list of type (T3*T4*T5*…Tn) and T2 is the first element of the list so T2=T3.

   The type of cdr in scheme is: T1 -> T2, when T1 is a list of type (T3*T4*T5*…Tn) and T2 is a list of type (T4*T5*…Tn).

4. The type of the following function: $(Define\ f\ (lambda\ (x)\ (values\ x\ x\ x)))$

$$\boldsymbol{T0} = [\boldsymbol{Tx} -> [\boldsymbol{Tx} * \boldsymbol{Tx} * \boldsymbol{Tx}]]$$

5. a. the MGU here is T1=T2

   b. any MGU here will work , because there is no generic property here the statement Number=Number will always be true.

   c. [T1*[T1->T2]->Number] , [[T3->Number]*[T4->Number]->Number]
   the MGU here is the following:
   T1 = [T3 -> Number] because of the left side argument of the main function
   T4 = T1 - because of the right side argument of the main function and nested the left side argument of that function. So now we can deduce that:
   T4 = [T3 -> Number]
   T2 = Number
   Substituting we got the following type statement:
   [[T3->Number]*[[T3->Number]->Number]->Number] ,
   [[T3->Number]*[[T3 -> Number]->Number]->Number]

   d. [T1->T1] , [T1->[Number->Number]] has the following MGU:
   T1 = T1,
   T1 = [Number->Number]
   Substituting we get:
   [[Number->Number] -> [Number->Number]],
   [[Number->Number]-> [Number->Number]]

Part 2 q.3

- $(define\ f\ (lambda\ (x)\ (values\ x\ (+\ x\ 1))))$

$$(define\ f$$

$$(lambda\ ([x:\ Number]): [Number\ *\ Number]$$

$$(values\ x\ (+\ x\ 1))))$$

- $(define\ g\ (lambda\ (x)\ (values\ \text{“}x\text{”}\ x)))$

$(define\ g$

$$(lambda\ ([x: Tx]): [String * Tx]$$

$$(values\ \text{“}x\text{”}\ x)))$$

Part 4 q.b

The advantages of the Promise object versus the callbacks are:

- Easier to understand.
- Working with synchronous operations that need to notify only once (usually completion or error).
- Coordinating or managing multiple asynchronous operations at the same time.
- Handling errors from nested or deeply nested asynchronous operations at the highest level without going into the nested operations
- Chaining asynchronous operations (such as do these two async operations, examine the result) without the need of complex syntax and code.
- Managing a mix of asynchronous and synchronous operations