EB2 -Type Inference

Exercise 1 ★

Determine, using the inference-tree method, which of the following expressions are typable. Exhibit which rules and substitutions apply in each step. In those cases in which the expression is not typable, justify your answer.

I. λz. if z then 0 else succ(0)

II. λx.λy. if x then y else succ(x)

III. λx. if isZero(x) then x else (if x then x else x)

IV. λx.λy. if x then y else succ(0)

V. if True then (λx. 0) 0 else (λx. 0) False

VI. (λf. if True then f 0 else f False) (λx. 0)

VII. λx.λy.λz. if z then y else succ(x)

VIII. fix (λx. pred(x))

In item VIII, assume that the type inference algorithm has been extended with the clause:

$$\mathbb{W}(fix) = \emptyset \triangleright fix_s : (s \to s) \to s, \text{ where } s \text{ is a fresh variable.}$$

Exercise 2 ★

Using the inference-tree method, infer the type of each of the following expressions or show that they not typable. In each step where you perform unification, show the set of equations to be unified and the substitution obtained from the process.

- $\lambda x.\ \lambda y.\ \lambda z.\ (z\ x)\ y\ z$
- $\lambda x.\ x\ (w\ (\lambda y.w\ y))$
- $(\lambda z.\lambda x.\ x\ (z\ (\lambda y.\ z)))$ True

Exercise 3

a) Extend the inference algorithm with the rules required to handle types

```
type 'a either = Left of 'a | Right of 'a
```

and

```
type 'a option = None | Some of 'a
```

b) Using these rules and the inference-tree method, type the expression:
`λx.if x then Some (Left (succ(0))) else None`

Exercise 4 ★

Suppose we extend the grammar of types and terms as follows:

$$\begin{array}{rcl} \sigma & ::= & \ldots \mid \langle \sigma \times \sigma \rangle \\ M & ::= & \ldots \mid \langle M, M \rangle \mid \pi_1(M) \mid \pi_2(M) \end{array}$$

We add the following typing rules:

$$\frac{\Gamma \rhd M : \sigma \qquad \Gamma \rhd N : \tau}{\Gamma \rhd \langle M, N \rangle : \langle \sigma \times \tau \rangle} \qquad \frac{\Gamma \rhd M : \langle \sigma \times \tau \rangle}{\Gamma \rhd \pi_1(M) : \sigma} \qquad \frac{\Gamma \rhd M : \langle \sigma \times \tau \rangle}{\Gamma \rhd \pi_2(M) : \tau}$$

Adapt the type inference algorithm for this extended language.

Exercise 5

Type the expression $(\lambda f.\langle f, 2 \rangle)\,(\lambda x.x\,1)$ using the extended type inference algorithm of exercise 4.

Exercise 6 ★

Attempt to type the following expression using the extended type inference algorithm of exercise 4.

$(\lambda f.\langle f2, fTrue \rangle)\,(\lambda x.x)$

Show exactly where it fails and why.

Exercise 7 ★

Extend the type inference algorithm so that it supports typing the *switch* of natural numbers, similar to that of C and C++. The extended language syntax is:

$$\mathtt{M} = \ldots \mid switch\ (\mathtt{M})\ \{\ case\ \underline{\mathtt{M}} :\ \mathtt{M}; case\ \underline{\mathtt{M}} :\ \mathtt{M}; \ldots; case\ \underline{\mathtt{M}} :\ \mathtt{M}\ default :\ \mathtt{M}\}$$

where each $\underline{\mathtt{M}}$ is a numeral (i.e. value of type Nat of the form 0, succ(0), succ(succ(0)), etc.).
The typing rule is:

$$\frac{\Gamma \rhd M : Nat \quad \forall i,j (1 \leq i,j \leq k \wedge i \neq j \Rightarrow n_i \neq n_j) \quad \Gamma \rhd N_i : \sigma\ ...\ \Gamma \rhd N_k : \sigma \quad \Gamma \rhd N : \sigma}{\Gamma \rhd switch\ (M)\ \{\ case\ \underline{n_1} :\ N_1\ ...\ case\ \underline{n_k} :\ N_k\ default : N\} : \sigma}$$

For example, an expression such as:

$$\lambda x.\ switch\ (x)\ \{\ case\ \underline{0} :\ True; default :\ False\}$$

should have type $Nat \rightarrow Bool$. But the expression:

$$switch\ (\underline{3})\ \{\ case\ \underline{1} :\ 1; case\ \underline{2} :\ 2; default :\ False\}$$

has not type, since not all branches have the same type. Finally, the expression:

$$switch\ (\underline{3})\ \{\ case\ \underline{1} :\ 1; case\ \underline{2} :\ 2; case\ \underline{1} :\ 3; default :\ 0\}$$

doesn't have a type either since there are repeated branches.

Exercise 8 ★

Extend the type inference algorithm to support binary trees. The extended syntax of types and terms is:

$$\begin{array}{rcl} \sigma & ::= & ...|tree_\sigma \\ M & ::= & ...|Empty_\sigma|Node_\sigma(M,N,O)\,|\,case\ M\ of\{nil : M; node(x,y,z) : M\} \end{array}$$

The new typing rules are::

$$\frac{}{\Gamma \rhd Empty_\sigma : tree_\sigma}$$

$$\frac{\Gamma \rhd M : tree_\sigma \quad \Gamma \rhd O : tree_\sigma \quad \Gamma \rhd N : \sigma}{\Gamma \rhd Node_\sigma(M,N,O) : tree_\sigma}$$

$$\frac{\Gamma \rhd M : tree_\tau \quad \Gamma \rhd P : \sigma \quad \Gamma, x : \tau, y : tree_\tau, z : tree_\tau \rhd Q : \sigma}{\Gamma \rhd caseMof\{empty : P; node(x,y,z) : Q\} : \sigma}$$

Note: the *Erase* function is extended as follows:

$$\begin{array}{rcl} Erase(Empty_\sigma) & = & Empty \\ Erase(Node_\sigma(M,N,O)) & = & Node(Erase(M), Erase(N), Erase(O)) \end{array}$$

Recall that the input to the type inference algorithm is an type erased term such as:

$$(\lambda x.Node(Empty, 5, Node(Empty, x, Empty)))\ 5$$

Exercise 9

This exercise asks you to extend type inference for dealing with *binary operators*. These operators are similar to functions except that they always take two arguments and they are infix.:

$$M\ ::=\ ...\ |\ \varphi x : \sigma\ y : \tau.M\ |\ \langle M\ N\ O\rangle \qquad \sigma\ ::=\ ...\ |\ \mathrm{Op}(\sigma, \tau \to \upsilon)$$

Here φ is the operator constructor thart binds variables x ("left" parameter) and y ("right" parameter) and $\langle M\ N\ O\rangle$ is the application of operator N to parameters M and O. $\mathrm{Op}(\sigma, \tau \to \upsilon)$ is the type of an operator whose left argument type is σ, the right argument type is τ and the result type is υ.

The typing rules:

$$\frac{\Gamma, x{:}\,\sigma,\ y{:}\,\tau \rhd M{:}\,\upsilon}{\Gamma \rhd \varphi x : \sigma\ y : \tau.M : \mathrm{Op}(\sigma, \tau \to \upsilon)}$$

$$\frac{\Gamma \rhd M : \sigma \quad \Gamma \rhd N : \mathrm{Op}(\sigma, \tau \to \upsilon) \quad \Gamma \rhd O : \tau}{\Gamma \rhd \langle M\ N\ O\rangle : \upsilon}$$

I. Extend the type inference algorithm to this extended language.

II. Apply your extended algorithm to: $\langle(\lambda x.\mathrm{succ}(x))\ (\varphi xy.xy)\ 0\rangle$