

Sample Solution for Homework 8

Problem 1 Type Checking and Type Inference (16 Points)

- (a) Use the type inference rules of the type system to determine for each of the following expressions whether the expression is well-typed. If an expression is not well-typed, explain why. If it is well-typed, give the inferred type. You do not need to show the individual inference steps.

(i)

```
1 const x = 2;
2 const y = x + 1;
3 x * y
```

The program is well-typed. The inferred type is **Num**.

(ii)

```
1 const x = true;
2 const y = x + 1;
3 x * y
```

The program is not well-typed. The $+$ operator on line 2 expects two expressions of type **Num** as arguments. However, the inferred type of x is **Bool**.

(iii)

```
1 const f = function f(x: Num): Num => Num (
2     function (y: Num) (x == y ? 1 : y * f(x)(y + 1))
3 );
4 f(3)
```

The program is well-typed. The inferred type is **Num** \Rightarrow **Num**.

(iv)

```
1 const f = function f(x: Num): Num => Num (
2     function (y: Num) (x == y ? 1 : y * f(x)(y + 1))
3 );
4 f(3) == f(3)
```

The program is not well-typed. The problem is that line 3 checks equality between the expression $f(3)$ and itself. However, the inferred type for $f(3)$ is **Num** \Rightarrow **Num** and the type system does not allow expressions of function types to be compared using the equality operator.

(v)

```
1 const f = function f(x: Num) (
2     function (y: Num) (x == y ? 1 : y * f(x)(y + 1))
3 );
4 f(3)(1)
```

The program is not well-typed. The problem is that line 1 declares a recursive function f . However, the function expression is missing the annotation of the return type. The type system does not support recursive function expressions without return type annotations.

- (b) For each of the following programs, find concrete types for the missing parameter type annotations τ_1 , τ_2 , and τ_3 such that the given program is well-typed according to the typing rules. If no such types exist, explain why. If you can find type annotations that make the program well-typed, what is the inferred type of f for your annotations? Are your chosen types the only annotations that work? If not, give at least one other choice of annotations that also makes the program well-typed.

(i)

```

1 const f = function(x:  $\tau_1$ )
2   (function(y:  $\tau_2$ ) (function(z:  $\tau_3$ ) (x(y(z)))));
3 const g = function(x: Num) (x + 1);
4 const h = function(x: Num) (x * 2);
5 f(h)(g)(3)

```

The program is well-typed for the following type annotations:

$$\begin{aligned}\tau_1 &= \mathbf{Num} \Rightarrow \mathbf{Num} \\ \tau_2 &= \mathbf{Num} \Rightarrow \mathbf{Num} \\ \tau_3 &= \mathbf{Num}\end{aligned}$$

These are the only possible type annotations for which the program is well-typed. The inferred type of f is

$$\tau_1 \Rightarrow \tau_2 \Rightarrow \tau_3 \Rightarrow \mathbf{Num}$$

(ii)

```

1 const f = function(x:  $\tau_1$ )
2   (function(y:  $\tau_2$ ) (function(z:  $\tau_3$ ) (x(y(z)))));
3 const g = function(x:  $\tau_3$ ) (x);
4 f(g)(g)

```

The program is well-typed for the following type annotations:

$$\begin{aligned}\tau_1 &= \mathbf{Num} \Rightarrow \mathbf{Num} \\ \tau_2 &= \mathbf{Num} \Rightarrow \mathbf{Num} \\ \tau_3 &= \mathbf{Num}\end{aligned}$$

Another possible annotation is:

$$\begin{aligned}\tau_1 &= \mathbf{Bool} \Rightarrow \mathbf{Bool} \\ \tau_2 &= \mathbf{Bool} \Rightarrow \mathbf{Bool} \\ \tau_3 &= \mathbf{Bool}\end{aligned}$$

In general, any annotation that satisfies the following equalities would work:

$$\tau_1 = \tau_2 = (\tau_3 \Rightarrow \tau_3)$$

The inferred type of `f` is as in (i) for the respective values of τ_1 , τ_2 , and τ_3 .

(iii)

```
1 const f = function(x:  $\tau_1$ ) (function(y:  $\tau_2$ ) (x(y)));  
2 const g = function(x: Bool) (x ? 1 : 0);  
3 const h = function(x: Num) (x + x);  
4 f(g)(true) + f(h)(1)
```

There exists no type annotation for which this program is well-typed. The problem is that calling `f` on `g` and `h` forces $\tau_1 = \mathbf{Num} \Rightarrow \mathbf{Num}$ and $\tau_1 = \mathbf{Bool} \Rightarrow \mathbf{Num}$. We thus must satisfy $\mathbf{Bool} = \mathbf{Num}$ which is impossible. To make the program well-typed we need a more expressive type system that supports type parameterization.