Principles of Programming Languages - Homework 8

Abhi Agarwal

1 Problem 1

(a)

(i): The expression is well typed. It will evaluate to a type number.

(ii): The expression is not well typed. You cannot add a Number to a Boolean. This happens on Line 2. This is dictated by the TypeArith rule.

(iii): The expression is well typed. It will evaluate to a type number => number.

(iv): The expression is not well typed. Line 4 doesn't follow the TypeEqual rule. Line 4 tries to test equality of two functions.

(v): The expression is not well typed. The anonymous function does not define a return type. This is dictated by the TypeFunctionRec rule. However, if x > y then this will return a Number, if x < y then it will never return (infinite loop).

(b)

(i)
t_1: number => number
t_2: number => number
t_3: number
f: (number => number) => (number => number)

(ii) Kind of like the opposite of (i). Since t_3 can be anything, which implies that t_1 can be anything, which implies that t_2 can be anything.
$t_1 : t_3$: Any
$t_2 : t_3$: Any
t_3: Any
Let's take an example of t_3 being Boolean. Then f: (boolean => boolean) => (boolean => boolean). This is a universal rule to this program.

(iii) There exist no concrete types for the mission parameter types such that the given program is well-typed according to the rules. For it to be well-typed t_2 has to be both a Boolean and a Number as it needs to satisfy both f(g)(true) and f(h)(1). This isn't possible so it isn't possible for it to be well-typed.