Principles of Programming Languages 202
Assignment 4

Ben Gindi - 205874142
Shira Segev - 208825349

- Code for parts 3 & 4 below.

Part 1: Theoretical Questions

Q1

Typint the expression: `((lambda (x1 y1) (if (> x1 y1) #t #f)) 8 3)`

Stage I: Rename bound variables:

`((lambda (x y) (if (> x y) #t #f)) 8 3)`

Stage II: Assign type variables to all sub-exps:

Expression	**Var**
`((lambda (x y) (if (> x y) #t #f)) 8 3)`	T_0
`(lambda (x y) (if (> x y) #t #f))`	T_1
`(if (> x y) #t #f)`	T_2
`(> x y)`	T_3
`>`	$T_>$
`x`	T_x
`y`	T_y
`#t`	$T_{\#t}$
`#f`	$T_{\#f}$
`8`	T_{num8}
`3`	T_{num3}

Stage III: Construct type equations.

- The equations for the sub-expressions are:

Expression	Equation
`((lambda (x y) (if (> x y) #t #f)) 8 3)`	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$
`(lambda (x y) (if (> x y) #t #f))`	$T_1 = [T_x * T_y \rightarrow T_2]$
`(if (> x y) #t #f)`	$T_2 = T_{\#t}\ and\ T_{\#t} = T_{\#f}$
`(> x y)`	$T_> = [T_x * T_y \rightarrow T_3]$

- The equations for the primitives are:

Expression	Equation
`>`	$T_> = [Number * Number \rightarrow Boolean]$
`#t`	$T_{\#t} = Boolean$
`#f`	$T_{\#f} = Boolean$
`8`	$T_{num8} = Number$
`3`	$T_{num3} = Number$

Stage IV: Solve the equations:

Equation	Substitution
$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$	{}
$T_1 = [T_x * T_y \rightarrow T_2]$	
$T_2 = T_{\#t}\ and\ T_{\#t} = T_{\#f}$	
$T_> = [T_x * T_y \rightarrow T_3]$	
$T_> = [Number * Number \rightarrow Boolean]$	
$T_{\#t} = Boolean$	
$T_{\#f} = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	

Step 1:

$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$ ∘Substitution=($T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$),a type-sub.

Substitution = Substitution ∘ ($T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$).

Equation	Substitution
$T_1 = [T_x * T_y \rightarrow T_2]$	{ $T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$ }
$T_2 = T_{\#t}$	
$T_{\#t} = T_{\#f}$	
$T_{>} = [T_x * T_y \rightarrow T_3]$	
$T_{>} = [Number * Number \rightarrow Boolean]$	
$T_{\#t} = Boolean$	
$T_{\#f} = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	

Step 2:

$T_1 = [T_x * T_y \rightarrow T_2]$ ∘Substitution=($T_1 = [T_{num8} * T_{num3} \rightarrow T_0] = [T_x * T_y \rightarrow T_2]$), not a type-sub. Substitution = Substitution ∘ (T_3= Boolean).

Equation	Substitution
$T_2 = T_{\#t}$	{ $T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$ }
$T_{\#t} = T_{\#f}$	
$T_{>} = [T_x * T_y \rightarrow T_3]$	
$T_{>} = [Number * Number \rightarrow Boolean]$	
$T_{\#t} = Boolean$	
$T_{\#f} = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	
$T_{num8} = T_x$	
$T_{num3} = T_y$	
$T_0 = T_2$	

Step 3:

$T_2 = T_{\#t}$ ∘Substitution=($T_2 = T_{\#t}$)

Substitution = Substitution ∘ $(T_2 = T_{\#t})$.

Equation	Substitution
$T_{\#t} = T_{\#f}$	{ $T_1 := [T_{num8} * T_{num3} \rightarrow T_0]$, $T_2 := T_{\#f}$ }
$T_{>} = [T_x * T_y \rightarrow T_3]$	
$T_{>} = [Number * Number \rightarrow Boolean]$	
$T_{\#t} = Boolean$	
$T_{\#f} = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	
$T_{num8} = T_x$	
$T_{num3} = T_y$	
$T_0 = T_2$	

Step 4:

$T_{\#t} = T_{\#f}$ ∘Substitution=($T_{\#t} = T_{\#f}$)

Substitution = Substitution ∘ $(T_{\#t} = T_{\#f})$.

Equation	Substitution
$T_{>} = [T_x * T_y \rightarrow T_3]$	{ $T_1 := [T_{num8} * T_{num3} \rightarrow T_0]$, $T_2 := T_{\#f}$ $T_{\#t} := T_{\#f}$ }
$T_{>} = [Number * Number \rightarrow Boolean]$	
$T_{\#t} = Boolean$	
$T_{\#f} = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	
$T_{num8} = T_x$	
$T_{num3} = T_y$	
$T_0 = T_2$	

Step 5:

$T_> = [T_x * T_y \to T_3]$ ∘Substitution=($T_> = [T_x * T_y \to T_3]$),

Substitution = Substitution ∘ ($T_> = [T_x * T_y \to T_3]$).

Equation	Substitution
$T_> = [Number * Number \to Boolean]$	{ $T_1 = [T_{num8} * T_{num3} \to T_0]$, $T_2 := T_{\#f}$, $T_{\#t} := T_{\#f}$, $T_> = [T_x * T_y \to T_3]$ }
$T_{\#t} = Boolean$	
$T_{\#f} = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	
$T_{num8} = T_x$	
$T_{num3} = T_y$	
$T_0 = T_2$	

Step 6:

$T_> = [Number * Number \to Boolean]$ ∘Substitution=($[T_x * T_y \to T_3] = [Number * Number \to Boolean]$), not a type-sub.

Substitution = Substitution ∘ ($T_> = [T_x * T_y \to T_3]$).

Equation	Substitution
$T_{\#t} = Boolean$	{ $T_1 = [T_{num8} * T_{num3} \to T_0]$, $T_2 := T_{\#f}$, $T_{\#t} := T_{\#f}$, $T_> = [T_x * T_y \to T_3]$ }
$T_{\#f} = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	
$T_{num8} = T_x$	
$T_{num3} = T_y$	
$T_0 = T_2$	
$T_x = Number$	
$T_y = Number$	
$T_3 =$ Boolean	

Step 7:

$T_{\#t} = Boolean$ ∘Substitution=($T_{\#f} = Boolean$) ,a type-sub.

Substitution = Substitution ∘ ($T_{\#f} = Boolean$).

Equation	Substitution
$T_{\#f} = Boolean$	{ $T_1 = [T_{num8} * T_{num3} \to T_0]$, $T_2 := Boolean$, $T_{\#t} := Boolean$, $T_{>} = [T_x * T_y \to T_3]$, $T_{\#t} = Boolean$ }
$T_{num8} = Number$	
$T_{num3} = Number$	
$T_{num8} = T_x$	
$T_{num3} = T_y$	
$T_0 = T_2$	
$T_x = Number$	
$T_y = Number$	
$T_3 =$ Boolean	

Step 8:

$T_{\#f} = Boolean$ ∘Substitution=($Boolean = Boolean$), true.

Substitution = Substitution ∘ ($Boolean = Boolean$).

Equation	Substitution
$T_{num8} = Number$	{ $T_1 = [T_{num8} * T_{num3} \to T_0]$, $T_2 := Boolean$, $T_{\#t} := Boolean$, $T_{>} = [T_x * T_y \to T_3]$, $T_{\#t} = Boolean$ }
$T_{num3} = Number$	
$T_{num8} = T_x$	
$T_{num3} = T_y$	
$T_0 = T_2$	
$T_x = Number$	
$T_y = Number$	
$T_3 =$ Boolean	

Step 9:

$T_{num8} = Number$ ∘Substitution=($T_{num8} = Number$), a type-sub.

Substitution = Substitution ∘ ($T_{num8} = Number$).

Equation	Substitution
$T_{num3} = Number$	{ $T_1 := [T_{num8} * T_{num3} \rightarrow T_0]$, $T_2 := Boolean$, $T_{\#t} := Boolean$, $T_{>} = [T_x * T_y \rightarrow T_3]$, $T_{\#t} = Boolean$, $T_{num8} := Number$ }
$T_{num8} = T_x$	
$T_{num3} = T_y$	
$T_0 = T_2$	
$T_x = Number$	
$T_y = Number$	
$T_3 =$ Boolean	

Step 10:

$T_{num3} = Number$ ∘Substitution=($T_{num3} = Number$), a type-sub.

Substitution = Substitution ∘ ($T_{num3} = Number$).

Equation	Substitution
$T_{num8} = T_x$	{ $T_1 := [T_{num8} * T_{num3} \rightarrow T_0]$, $T_2 := Boolean$, $T_{\#t} := Boolean$, $T_{>} = [T_x * T_y \rightarrow T_3]$, $T_{\#t} = Boolean$, $T_{num8} := Number$, $T_{num3} = Number$ }
$T_{num3} = T_y$	
$T_0 = T_2$	
$T_x = Number$	
$T_y = Number$	
$T_3 =$ Boolean	

Step 11:

$T_{num8} = T_x$ ∘Substitution=($T_x = Number$), a type-sub.

Substitution = Substitution ∘ ($T_x = Number$).

Equation	Substitution
$T_{num3} = T_y$	{ $T_1 := [T_{num8} * T_{num3} \to T_0]$, $T_2 := Boolean$, $T_{\#t} := Boolean$, $T_> = [T_x * T_y \to T_3]$, $T_{\#t} = Boolean$, $T_{num8} := Number$, $T_{num3} := Number$, $T_{num8} := T_x$ }
$T_0 = T_2$	
$T_x = Number$	
$T_y = Number$	
$T_3 = Boolean$	

Step 12:

$T_{num3} = T_y$ ∘Substitution=($T_y = Numberr$), a type-sub.

Substitution = Substitution ∘ ($T_y = Number$).

Equation	Substitution
$T_0 = T_2$	{ $T_1 := [T_{num8} * T_{num3} \to T_0]$, $T_2 := Boolean$, $T_{\#t} := Boolean$, $T_> := [T_x * T_y \to T_3]$, $T_{\#t} := Boolean$, $T_{num8} := Number$, $T_{num3} := Number$, $T_{num8} := T_x$, $T_{num3} := T_y$ }
$T_x = Number$	
$T_y = Number$	
$T_3 = Boolean$	

Step 13:

$T_0 = T_2$ ∘Substitution=($T_0 = Boolean$), a type-sub.

Substitution = Substitution ∘ ($T_0 = T_2$).

Equation	Substitution
$T_x = Number$	{ $T_1 := [T_{num8} * T_{num3} \to T_0]$, $T_2 := Boolean$, $T_{\#t} := Boolean$, $T_> := [T_x * T_y \to T_3]$, $T_{\#t} := Boolean$, $T_{num8} := Number$, $T_{num3} := Number$, $T_{num8} := T_x$, $T_{num3} := T_y$, $T_0 := Boolean$ }
$T_y = Number$	
$T_3 = Boolean$	

Step 14:

$T_x = Number$ ○Substitution=($Number = Number$), true.

Equation	Substitution
$T_y = Number$	{ $T_1 := [T_{num8} * T_{num3} \to T_0]$, $T_2 := Boolean$, $T_{\#t} := Boolean$, $T_{>} := [T_x * T_y \to T_3]$, $T_{\#t} := Boolean$, $T_{num8} := Number$, $T_{num3} := Number$, $T_{num8} := T_x$, $T_{num3} := T_y$, $T_0 := Boolean$, $T_x := Number$ }
$T_3 = Boolean$	

Step 15:

$T_y = Number$ ○Substitution=($Number = Number$), true.

Equation	Substitution
$T_3 = Boolean$	{ $T_1 := [T_{num8} * T_{num3} \to T_0]$, $T_2 := Boolean$, $T_{\#t} := Boolean$, $T_{>} := [T_x * T_y \to T_3]$, $T_{\#t} := Boolean$, $T_{num8} := Number$, $T_{num3} := Number$, $T_{num8} := T_x$, $T_{num3} := T_y$, $T_0 := Boolean$, $T_x := Number$, $T_y := Number$ }

Final Step (16):

($T_3 = Boolean$) ○ Substitution = ($T_3 = Boolean$), type-sub.

Substitution = Substitution ○ ($T_3 = Boolean$).

Equation	Substitution
	{ $T_1 := [Number \ast Number \rightarrow Boolean]$, $T_2 := Boolean$, $T_{\#t} := Boolean$, $T_{>} := [Number \ast Number \rightarrow Boolean]$, $T_{\#t} := Boolean$, $T_{num8} := Number$, $T_{num3} := Number$, $T_x := Number$, $T_y := Number$, $T_0 := Boolean$, $T_3 := Boolean$ }

$T_1 := [Number \ast Number \rightarrow Boolean]$

Q2

a. `{f:[T1->T2], x: T1} |- (f x)}: T2` ⇒ **true**

Explanation: The function `f` receives argument of type `T1`, and returns `T2`. And indeed, the argument `x` is defined as `T1`, and the return value of `(f x)` is `T2.`

b. `{f:[T1->T2],g: [T2->T3]}, x: T2}|- (f g x): T3` ⇒ **false**

Explanation: Number of arguments is illegal (`f` receives one argument of type `T1`, but it is called with **two** arguments (`g x`)).

c. `{f:[T2->T1],g: [T1->T2], x: T1}|- (f (g x)): T1` ⇒ **true**

Explanation:

- The function `f` receives argument of type `T2`, and returns `T1`.
- The function `g` receives argument of type `T1`, and returns `T2`.
- The argument `x` is defined as `T1`.
- In the expression `(g x)`, the return value is `T2`.
- Then, `f` indeed receives `T2`, and returns `T1`.

d. `{f:[T2->Number], x: Number}|- (f x x): Number` ⇒ **false**

Explanation: number of arguments is illegal (`f` receives `Number` as an argument, but it is called with **two** numbers (`x x`)).

Q3

a. cons- `Type:` $[\, T_1 * T_2 \rightarrow Pair(T_1, T_2) \,]$

b. car- `Type:` $[\, Pair(T_1, T_2) \rightarrow T_1 \,]$

c. cdr- `Type:` $[\, Pair(T_1, T_2) \rightarrow T_2 \,]$

Q4

```
(Define f
    (lambda (x)
    (values x x x)))
```

The type of function f is: `f (x: T): [T * T * T]`

Q5

a. T_1, T_2

MGU: $T_1 = T_2$

b. $Number, Number$

MGU: already a Number

c. $[\,T_1 * [\,T_1 \rightarrow T_2\,] \rightarrow Number\,]$, $[\,[\,T_3 \rightarrow Number\,] * [\,T_4 \rightarrow Number\,] \rightarrow N\,]$

MGU: $T_1 = [\,T_3 \rightarrow Number\,]$, $T_2 = Number$, $T_4 = [\,T_3 \rightarrow Number\,]$

d. $[\,T_1 \rightarrow T_1\,]$, $[\,T_1 \rightarrow [\,Number \rightarrow Number\,]$

MGU: $T_1 = [\,Number \rightarrow Number\,]$

Part 2: Type Checking

- We implemented values as a special form.

Q2.3

```
a. (define f
        (lambda (x: Number): [Number * Number]
            (values x (+ x 1))))
```

The type of function f is: `f (x: Number): [Number * Number]`

```
b. (define g
        (lambda (x: T): [string * T]
            (values “x” x)))
```

The type of function f is: `f (x: T): [string * T]`

Part 4: Promises

Q1.b

The benefits of the promise interface compared to the callback interface are:

- **Code** written in promise interface is **more readable**
- The **type** of functions returning Promises is **more informative** and similar to the simple types of synchronous versions
- **Composition** is simplified by **chaining** (we can chain sequences of asynchronous calls in a chain of `.then()` calls).
- We can aggregate **error handling** in a **single handler** for a chain of calls, in a way similar to exception handling.

Part 3: Generators

Q1

```
/*
Purpose: given two generators, the method returns a generator
    that combines both generators by:
    interleaving their values
Signature: braid(gen1, gen2)
Type: [Generator * Generator -> Generator]
*/
export function* braid(gen1: Generator, gen2: Generator) {
    var ir1 , ir2;

    while(1) {
        ir1 = gen1.next();
        ir2 = gen2.next();

        if (!ir1.done){
            yield ir1.value;
        }
        if (!ir2.done){
            yield ir2.value;
        }
        if(ir1.done && ir2.done) {
            break;
        }
    }
}
```

Q2

```
/*
Purpose: given two generators, the method returns a generator
    that combines both generators by:
    taking two elements from gen1 and one from the gen2.
Signature: biased(gen1, gen2)
Type: [Generator * Generator -> Generator]
*/
export function* biased(gen1: Generator, gen2: Generator) {
    var ir1, ir2;

    while (1) {
        ir1 = gen1.next();
        ir2 = gen2.next();
        if (!ir1.done){
            yield ir1.value;
            ir1 = gen1.next();
            if (!ir1.done){
                yield ir1.value;
            }
        }
        if (!ir2.done){
            yield ir2.value;
        }
        if (ir1.done && ir2.done) {
            break;
        }
    }
}
```

Part 4: Promises

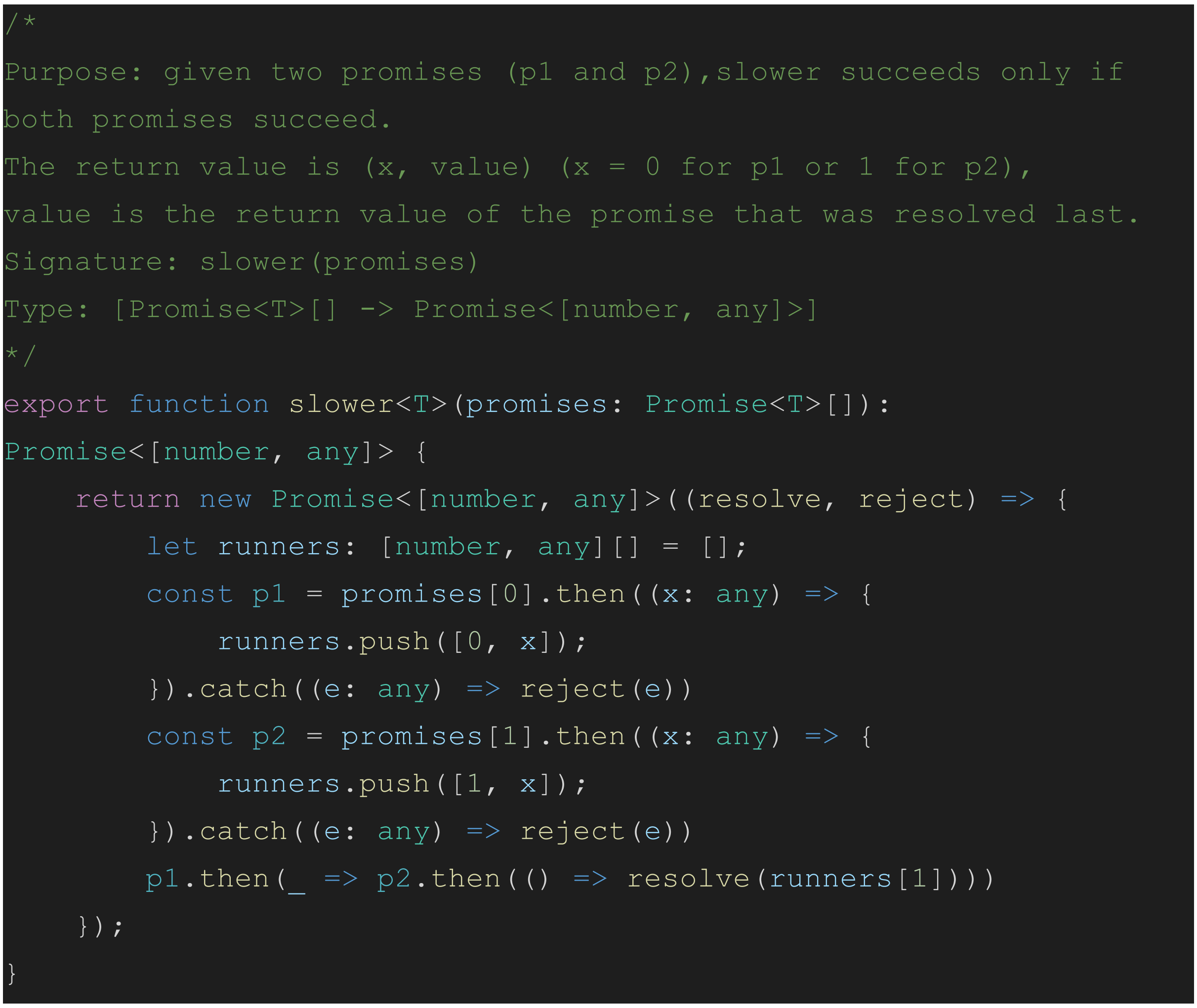
Q1.a

```
export function f(x: number): Promise<number> {
    return new Promise<number>((resolve, reject) =>{
        if (x != 0){
            resolve(1 / x);
        }
        else{
            reject(new Error("can't divide by zero"));
        }
    });
}

export function g(x: number): Promise<number> {
    return new Promise<number>((resolve, reject) => {
        if (x != null) {
            resolve(x * x);
        }
        else {
            reject(new Error("can't multiply nulls"));
        }
    });
}

/*
Purpose: composition of 2 functions f(g(x)).
Signature: h(x)
Type: [number -> Promise<number>]
*/
export function h(x: number): Promise<number> {
    return new Promise<number> (async (resolve, reject) => {
        g(x).then((gRes: number) => {
            f(gRes).then((fRes: number) => {
                resolve(fRes);
            }).catch((e: any) => reject(e));
        }).catch((e: any) => reject(e));
    });
}
```

Q2



```
/*
Purpose: given two promises (p1 and p2),slower succeeds only if
both promises succeed.
The return value is (x, value) (x = 0 for p1 or 1 for p2),
value is the return value of the promise that was resolved last.
Signature: slower(promises)
Type: [Promise<T>[] -> Promise<[number, any]>]
*/
export function slower<T>(promises: Promise<T>[]):
Promise<[number, any]> {
    return new Promise<[number, any]>((resolve, reject) => {
        let runners: [number, any][] = [];
        const p1 = promises[0].then((x: any) => {
            runners.push([0, x]);
        }).catch((e: any) => reject(e))
        const p2 = promises[1].then((x: any) => {
            runners.push([1, x]);
        }).catch((e: any) => reject(e))
        p1.then(_ => p2.then(() => resolve(runners[1])))
    });
}
```