Part 1

1. ((lambda (x1 y1) (if (> x1 y1) #t #f)) 8 3)

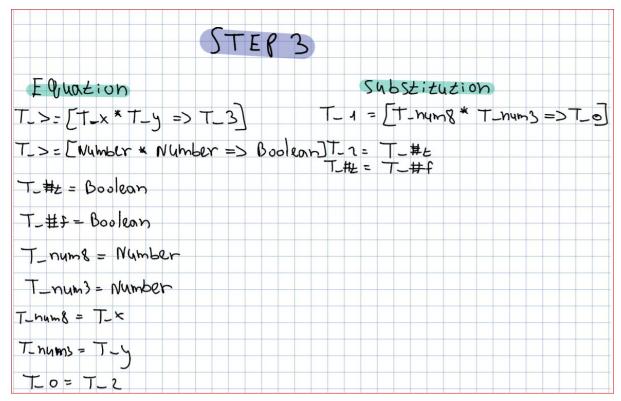
• Rename: ((lambda (x y) (if (> x y) #t #f)) 8 3)

•

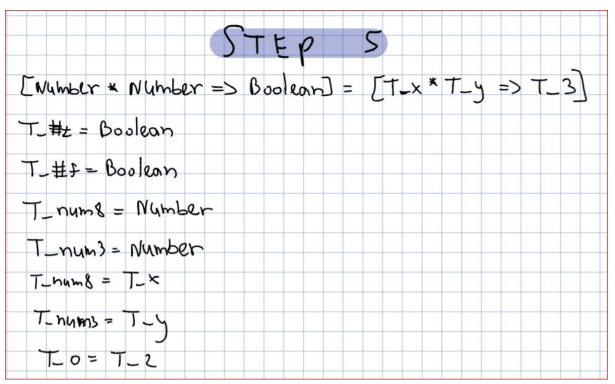
Expression	Var
((lambda (x y) (if (> x y) #t #f)) 8 3)	T_0
(lambda (x y) (if (> x y) #t #f))	T_1
(if (> x y) #t #f)	T_2
(> x y)	T_3
>	T_>
х	T_x
у	T_y
#t	T_#t
#f	T_#f
8	T_num8
3	T_num3

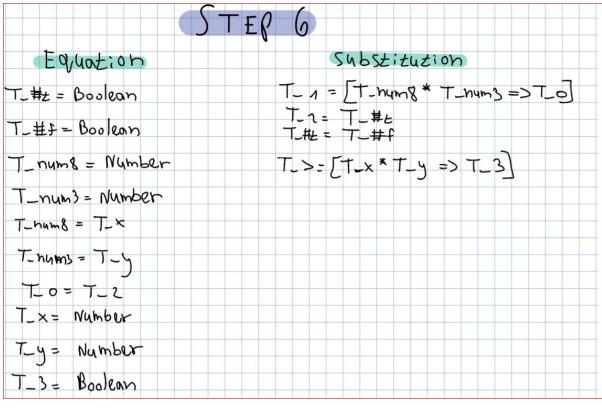
Expression	Equation
((lambda (x y) (if (> x y) #t #f)) 8 3)	T_1 = [T_num8 * T_num3 => T_0]
(lambda (x y) (if (> x y) #t #f))	T_1 = [T_x * T_y => T_2]
(if (> x y) #t #f)	T_2 = T_#t T_2 = T_#f
(> x y)	T_> = [T_x * T_y => T_3]
>	T_> = [Number * Number => Boolean]
#t	T_#t = Boolean
#f	T_#f = Boolean
8	T_num8 = Number
3	T_num3 = Number

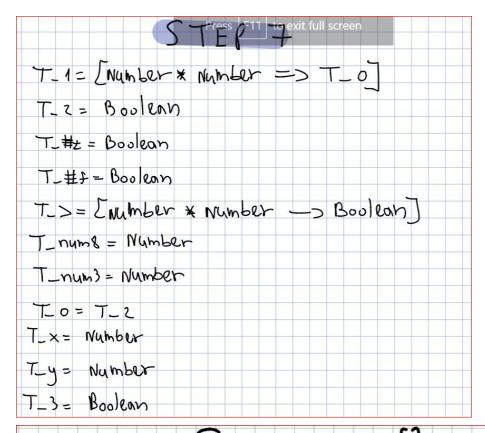
step 4:



STEPU	
E Quazion	Substitution
T_>= [Number * Number => Boolean]	T_1 = [T-hym8 * T_hym3 => T_0]
T_# = Boolean	T.推□ T_#f
T_#f = Boolean	T.>=[t-x*T-y=>T-3]
T_num8 = Number	
T_num3 = Number T_num8 = T_x	
T-nums = T-y	
T-0= T-2	







TEP
T-1= [Number * Number => Boolean]
T-z = Boolean
T_#z = Boolean
T_#f=Boolean
T_>= [Number * Number _> Boolean]
J_num8 = Number
T_num3 = Number
To= Boolean
T_x= Number
Ty = Nymber
T_3 = Boolean

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

הסקת הטיפוס נכונה כי האילוץ היחיד שלנו הוא שהפרמטר ש-f מקבלת יהיה מטיפוס T1 ואכן x מטיפוס T1.

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

ו- X ו g - אותו g אינה נכונה כי f אמורה לקבל פרמטר יחיד אולם מקבלת שני פרמטרים f (לא אותו באי). בנאי).

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

הסקת הטיפוס נכונה כי g מקבלת פרמטר מטיפוס T1 ומחזירה פרמטר מטיפוס x, ואכן x מטיפוס g הסקת הטיפוס נכונה כי g מקבלת פרמטר מטיפוס T1. פונקציה f מקבלת את g כפרמטר ואכן כפי שאמרנו T2 מחזירה פרמטר מטיפוס T2 התואם לפרמטר ש-f מקבלת.

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

x - ו- x אולם מקבלת שני פרמטרים - x ו- x הסקת הטיפוס אינה נכונה שכן f אמורה לקבל פרמטר יחיד (מטיפוס (T2) אולם מקבלת שני פרמטרים (לא אותו בנאי).

3

a. cons:[T1 * T2 -> Pair(T1 . T2)]

b. car:[Pair(T1 . T2) -> T1]

c. cdr:[Pair(T1 . T2) -> T2]

4

(Define f (lambda (x) (values x x x))) f: $[T1 \rightarrow [T1 * T1 * T1]]$

5

- a. $\{T1 = T2\}$
- b. The MGU will be empty: {}
- c. {T1 = T4 = [T3 -> Number], T2 = Number, N = Number}
- d. {T1 = [Number -> Number]}

Part 2

values is implemented as special form.

```
2.3 (define f (lambda (x) (values x (+ x 1)))) f: [number ->[number * number ]] (define g (lambda (x) (values "x" x))) g: [ T1 ->[string * T1 ] ]
```

Part 4

b. Benefits of the promise interface compared to the callback interface:

- The type of functions returning Promises is more informative and similar to the simple types of synchronous versions.
- 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.

(took from the class material)

Part 3 code:

```
export function* braid <T,S,R> (gen1 : Generator <T,S,R> ,gen2
: Generator<T,S,R> ) : Generator<T,void,R> {
  let moshe = gen1.next();
  let moshe2 = gen2.next();
  while(!moshe.done || !moshe2.done) {
    if(!moshe.done) {
      yield moshe.value;
      moshe=gen1.next();
    }
  if(!moshe2.done) {
      yield moshe2.value;
      moshe2=gen2.next();
    }
}
```

```
export function* biased <T,S,R> (gen1 : Generator <T,S,R>
,gen2 : Generator<T,S,R> ) : Generator<T,void,R> {
    let moshe = gen1.next();
   let moshe2 = gen2.next();
   while(!moshe.done || !moshe2.done) {
        if(!moshe.done) {
            yield moshe.value;
            moshe=gen1.next();
        }
        if(!moshe.done) {
            yield moshe.value;
            moshe=gen1.next();
        }
        if(!moshe2.done) {
            yield moshe2.value;
            moshe2=gen2.next();
        }
```

Part 4 Code:

```
export function f (x : number) : Promise<number> {
    return new Promise<number>((resolve, reject) => {
        (x===0)? reject(new Error ("dived by zero")) :
    resolve(1/x);
    });
}
```

```
export function g (x : number) : Promise<number> {
    return new Promise<number>((resolve, reject) =>
resolve(x*x));
export function h (x: number) : Promise<number> {
    return new Promise<number> ((resolve, reject) => g(x).then(
(num) => f(num)).then((num) => {resolve (num)}).catch(() =>
reject(new Error("Something wrong happened D-:"))));
export const slower =<T> (promises: Promise<any>[]):
Promise<[number,T]> =>{
    return new Promise((resolve, reject) => {
        let done = promises.length;
        promises.forEach((val, index) => {
            Promise.resolve(val).then(res => {
                done --;
                if (done == 0) {
                    resolve([index,res]);
            }).catch(err => reject(err));
        });
    });
```