

Assignment 4 – PPL

Segev Nagar – 312529316, Eli Avissra - 311510887

- בחלק 2 ישנו באמצעות Nested tuples ווליא Special Form

حلק 1

נכע type inference על הביטוי: $((lambda(xy)(if(xy)| |t| |f))83)$

שלב 1: לא צריך לבצע renaming ל-bound variables

שלב 2: Assign type variables for every sub expression and primitives

Expression	Variable
$((lambda(xy)(if(xy) t f))83)$	T_0
$(lambda(xy)(if(xy) t f))$	T_1
$(if(xy) t f)$	T_2
(xy)	T_3
	T
x	T_x
y	T_y
t	T_t
f	T_f
8	T_{num8}
3	T_{num3}

שלב 3: Construct type equations

Expression	Equation
$((lambda(xy)(if(xy) t f))83)$	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$
$(lambda(xy)(if(xy) t f))$	$T_1 = [T_x * T_y \rightarrow T_2]$
$(if(xy) t f)$	$T_2 = T_t$ $T_t = T_f$
(xy)	$T = [T_x * T_y \rightarrow T_3]$
T	$T = [Number * Number \rightarrow Boolean]$
T_t	$T_t = Boolean$
T_f	$T_f = Boolean$
8	$T_{num8} = Number$
3	$T_{num3} = Number$

:Solve the equations :4 ב'ש

Equation	Substitution
$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$	{}
$T_1 = [T_x * T_y \rightarrow T_2]$	
$T_2 = T_t$	
$T_t = T_f$	
$T = [T_x * T_y \rightarrow T_3]$	
$T = [Number * Number \rightarrow Boolean]$	
$T_t = Boolean$	
$T_f = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	
Equation	Substitution
$T_1 = [T_x * T_y \rightarrow T_2]$	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$
$T_2 = T_t$	
$T_t = T_f$	
$T = [T_x * T_y \rightarrow T_3]$	
$T = [Number * Number \rightarrow Boolean]$	
$T_t = Boolean$	
$T_f = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_2 = T_0$	
Equation	Substitution
$T_2 = T_t$	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0]$
$T_t = T_f$	
$T = [T_x * T_y \rightarrow T_3]$	
$T = [Number * Number \rightarrow Boolean]$	
$T_t = Boolean$	
$T_f = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_2 = T_0$	
Equation	Substitution
$T = [T_x * T_y \rightarrow T_3]$	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0] T_2 = T_t$

$T = [Number * Number \rightarrow Boolean]$	$T_2 = t_f$
$T_t = Boolean$	
$T_f = Boolean$	
$T_{num8} = Number$	
$T_{num3} = Number$	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_2 = T_0$	
Equation	Substitution
$T = [Number * Number \rightarrow Boolean]$	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0] T_2 = T_t$
$T_t = Boolean$	$T_2 = t_f$
$T_f = Boolean$	$T = [T_x * T_y \rightarrow T_3]$
$T_{num8} = Number$	
$T_{num3} = Number$	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_2 = T_0$	
Equation	Substitution
$T_t = Boolean$	$T_1 = [T_{num8} * T_{num3} \rightarrow T_0] T_2 = T_t$
$T_f = Boolean$	$T_2 = t_f$
$T_{num8} = Number$	$T = [T_x * T_y \rightarrow T_3]$
$T_{num3} = Number$	
$T_x = T_{num8}$	
$T_y = T_{num3}$	
$T_2 = T_0$	
$T_x = Number$	
$T_y = Number$	
$T_3 = Boolean$	
Equation	Substitution
$T_x = T_{num8}$	$T_1 = [Number * Number \rightarrow T_0] T_t$
$T_y = T_{num3}$	$= Boolean$
$T_2 = T_0$	$T_f = Boolean$
$T_x = Number$	$T = [T_x * T_y \rightarrow T_3]$
$T_y = Number$	$T_2 = Boolean$
$T_3 = Boolean$	$T_{num8} = Number$
	$T_{num3} = Number$
Equation	Substitution
$T_2 = T_0$	$T_1 = [Number * Number \rightarrow T_0] T_t$
$T_x = Number$	$= Boolean$
$T_y = Number$	$T_f = Boolean$
$T_3 = Boolean$	$T = [Number * Number \rightarrow T_3]$
	$T_2 = Boolean$
	$T_{num8} = Number$
	$T_{num3} = Number$
	$T_x = Number$

	$T_y = Number$
Equation	Substitution
	$T_1 = [Number * Number \rightarrow Boolean]T_t$ = Boolean $T_f = Boolean$
	$T = [Number * Number \rightarrow Boolean]$ $T_2 = Boolean$
	$T_{num8} = Number$
	$T_{num3} = Number$
	$T_x = Number$
	$T_y = Number$
	$T_0 = Boolean$
	$T_3 = Boolean$

שאלה 2

- א. כן. משום ש- x מטיפוס T_1 ו- f מקבלת T_1 כפרמטר ומחזירה T_2 , ולכן הביטוי (fx) מטיפוס T_2 . לכן הביטוי חוקי.
- ב. לא, כי f לא מקבלת שני פרמטרים, אלא היא מקבלת רק פרמטר אחד. מכאן הביטוי אינו חוקי.
- ג. כן. כי g מקבלת T_1 ומחזירה T_2 ו- x מטיפוס T_1 ולכן g מקבלת T_2 . כמו כן, f מקבלת T_2 ומחזירה T_1 ואכן g מטיפוס $T_2 \Leftarrow$ הטייפוס של הביטוי $((gx)f)$ הוא T_1 .
- ד. לא, כי f מקבלת פרמטר אחד מטיפוס T_2 ובביטוי (axf) היא מקבלת שני פרמטרים ולכן לא ניתן להסיק את הנדרש. אז הביטוי אינו חוקי.

שאלה 3

*cons: $[T_1 * T_2 \rightarrow Pair(T_1 * T_2)]$.a*
*car: $[Pair(T_1 * T_2) \rightarrow T_1]$.b*
*cdr: $[Pair(T_1 * T_2) \rightarrow T_2]$.c*

שאלה 4

$f: [T \rightarrow T * T * T]$

שאלה 5

- $MGUis\{T_1 = T_2\}$.a
 $MGUis\{\}$.b
 $MGUis\{T_1 = [T_3 \rightarrow Number], T_4 = [T_3 \rightarrow Number], T_2 = Number\}$.c
 $MGUis\{T_1 = [Number \rightarrow Number]\}$.d

חלון 2

שאלה 3

```

(define f : [number -> number * number]
  (lambda (x : number) : number * number
    (values x (+ x 1)))

(define g : [T -> string * T]
  (lambda (x : T) : string * T
    (values "x" x)))
  
```

חלון 3

שאלה 1ב

- הרכבה של פונקציות אסינכרוניות במשתק promise יותר קל יותר להבנה וזכה ביתרונות פשוטות על פני המשתק callback.
- הטיפוס של פונקציות אסינכרוניות promise דומה יותר בדרך שבה נגידר ונמשט טיפוס של פונקציות מסונכרנות, מאשר במשתק callback.
- טיפול בשגיאות במשתק promise ניתן לבצע במקום אחד. לעומת זאת, במשתק promise נוצרה הצורך בהרבה התנאיות if-else שבויל להתמודד עם שגיאות, מה שיכל ליצור קוד מבלבל.

generator.ts

```

function* braid(generator1: Generator, generator2: Generator): Generator {
  let v1 = generator1.next();
  let v2 = generator2.next();

  while (!v1.done && !v2.done) {
    yield v1.value;
  }
}
  
```

```

yield v2.value;
v1 = generator1.next();
v2 = generator2.next();
if (v1.done && !v2.done) {
    while (!v2.done) {
        yield v2.value
        v2 = generator2.next();
    }
} else if (v2.done && !v1.done) {
    while (!v1.done) {
        yield v1.value
        v1 = generator1.next();
    }
}
}

function* biased(generator1: Generator, generator2: Generator): Generator
{
    let v1 = generator1.next();
    let v2 = generator2.next();
    while (!v1.done && !v2.done) {
        yield v1.value;
        v1 = generator1.next();
        yield v1.value;
        v1 = generator1.next();
        yield v2.value
        v2 = generator2.next();
        if (v1.done && !v2.done) {
            while (!v2.done) {
                yield v2.value
                v2 = generator2.next();
            }
        } else if (v2.done && !v1.done) {
            while (!v1.done) {
                yield v1.value
                v1 = generator1.next();
            }
        }
    }
}

function* gen1() { yield 3; yield 6; yield 9; yield 12; }
function* gen2() { yield 8; yield 10; } for (let n of take(4, biased(gen1(),
gen2())))) { console.log(n); }

```

```
function* take(n: number, generator: Generator) {
  for (let x of generator) {
    if (n <= 0)
      return; n--;
    yield x;
  }
}
```

promises.ts

```
import { promises, resolve } from "dns";
import { reject } from "ramda";

function f(x: number): Promise<number> {
  return new Promise<number>((resolve, reject) => {
    try {
      resolve(1 / x);
    }

    catch (e) {
      reject(e)
    }
  });
}

function q(x: number): Promise<number> {
  return new Promise<number>((reslove, reject) => {
    try {
      reslove(x * x);
    }

    catch (e) {
      reject(e)
    }
  });
}

function h(x: number): Promise<number> {
  return new Promise<number>(async (reslove, reject) => {
    try {
      reslove(await f(await q(x)));
    }

    catch (e) {
```

```
        reject(e)
    }
});
```

```
}
```

```
function slower<T1, T2>(promise1: Promise<T1>, promise2: Promise<T2>): Promise<[T1 | T2, number]> {
    return new Promise<[T1 | T2, number]>((resolve, reject) => {
        let arr: [T1 | T2, number][] = [];
        const p1 = promise1.then((x: T1) => {
            arr.push([x, 0])
        })
            .catch(e => reject(e))
        const p2 = promise2.then((x: T2) => {
            arr.push([x, 1])
        })
            .catch(e => reject(e))
        p1.then(_ => p2.then(() => resolve(arr[1])))
    });
}

const exmpro = new Promise((resolve, reject) => setTimeout(resolve, 20, "one"))
const exmpro2 = new Promise((resolve, reject) => setTimeout(resolve, 50, "two"))

slower(exmpro, exmpro2).then(value => console.log(value))
```