# EOPL Pset 6

Alireza Habibzadeh `99109393`

## 1

### الف

به نظر من در این روش دو فایده وجود دارد.

یکی این که کار برنامه‌نویس ساده‌تر می‌شود و فکر کردنش سطح بالاتر و دورتر از مفاهیم ماشین پس زبان سطح بالاتر می‌شود و کد کوتاه‌تر. به قولی وقتی اشیا روی میز را نام می‌برید به type آن‌ها فکر نمی‌کنید. تنها برای مثال می‌گویید این یک لیوان است و اگر لازم شد اطلاعات بیشتری می‌دهید یا خود فرد آن‌ها را استنتاج می‌کند.

> *"...I mean, types are there to make the compiler writer's jobs easier. Types are not, I mean.. Heck, do you have an ontology of types for just the objects in this table? No. So types are there because compiler writers are human and they're limited in what they can do."*
>
> Peter Wang (CEO and Co-founder of Anaconda; creator of pyscript_dev, PyData, Bokeh, Datashader)

فایده‌ی دوم این که به نظر من type-checking تغییرات کد را در آینده دشوارتر می‌کند. روش type-inference هرچند dynamic با تعریف زبان‌های dynamic نیست، اما به یک زبان dynamic نزدیک‌تر است چرا که تنها با تغییر کوچک در روندی که type نهایی توسط مفسر infer شده (مثلا عوض کردن type خروجی یک تابع) می‌توان کارکرد یک برنامه را کلی‌تر کرد یا از کاربردی به کاربرد دیگر کوچ داد. (مثلا فرایندی از جنس $int \to int$ که شباهت‌هایی به فرایند دیگری از جنس $int \to bool$ دارد در صورتی که به روش type-checking پیاده شده باشد نیاز به تغییر بسیاری از تایپ‌ها دارد اما در روش type-inference چند تغییر جزیی کار را انجام می‌دهد.

### ب

با این که هر دو زبان strongly-typed محسوب می‌شوند دلیل این حرف این است که واقعا در Java هرچند به قیمت performance پایین‌تر type-checking بیشتری وجود دارد.

```
// Java is more strongly typed than C

public class MyClass {
    public static void main(String args[]) {
      float value  = 6.98;  // compile-time error
      double value2 = 6.98;  // OK
      float value3  = 6.98f; // also OK
    }
}
```

```
/MyClass.java:3: error: incompatible types: possible lossy conversion from double to
float
      float value = 6.98;  // compile-time error
```

```
                    ^
1 error
```

در کد بالا عدد 6.98 ابتدا به صورت یک double تولید شده و سپس کامپایلر سعی دارد آن را به یک متغییر از جنس float assign کند. از آنجایی که این تبدیل ممکن است بی‌نقص نباشد و مقداری داده دور ریخته شود کامپایلر جاوا مگر این که برنامه‌نویس صراحتا تبدیل تایپ را بیان کند اجازه‌ی این کار را نمی‌دهد.

یا مثال‌های جالب دیگری که پیدا کردم در نسخه‌های اولیه‌ی C و C++ در function callها هیچ type-checkingای روی پارامترها انجام نمی‌شده. در واقع این دو زبان پارامترها را مشتی داده می‌دیدند که می‌شود عملیات‌های هر تایپی را روی هر تایپی دیگری اجرا کرد که هر چند می‌توان با این امکان کارهای خلاقانه کرد (Fast Inverse Square Root — A Quake III Algorithm YouTube Wikipedia) ممکن است باعث ایجاد خطاهای زیادی هم بشود.

ولی هنوز هم توابعی که تعداد متغییری ورودی می‌گیرند مثل `scanf` و `printf` به طور کامل ورودی‌هایشان type-check نمی‌شود.

مثال آخر هم در cast کردن رفرنس‌ها است. در C می‌توان یک `int*` را به یک `char*` کست کرد و زبان واقعا مموری آن نقطه را re-interpret شده می‌بیند ولی در Java تنها وقتی می‌توانید رفرنس یک Object را به String کست کنید که آن Object در درجه‌ی اول واقعا String باشد وگرنه با خطا مواجه می‌شوید. در واقع C تغییر typeهای implicit بیشتری در اختیار برنامه‌نویس می‌گذارد.

به طور کلی زمانی می‌توان گفت یک زبان strongly-typed است که در آن زبان یک سخت‌گیری حداقلی در چک شدن، اظهار توسط برنامه‌نویس و تبدیل typeها وجود دارد. به عبارتی هر لحظه که برنامه‌نویس عبارتی را می‌نویسد باید حواسش به تایپ تک‌تک عناصر و تبدیل شدن آن‌ها باشد و نمی‌تواند کارها را به مفسر زبان واگذار کند.

البته اینجا مقایسه بین دو زبانی هست که هر دو strongly-typed محسوب می‌شوند.

## 2

### a

| Expression | Type Variable |
|---|---|
| `p` | $t_p$ |
| `let p = zero?(1) in if p then 88 else 99` | $t_0$ |
| `zero?(1)` | $t_1$ |
| `if p then 88 else 99` | $t_2$ |

| Expression | Equations |
|---|---|
| `let p = zero?(1) in if p then 88 else 99` | $t_p = t_1$ |
| | $t_0 = t_2$ |
| `zero?(1)` | $t_1 = bool$ |
| | $int = int$ |
| `if p then 88 else 99` | $t_p = bool$ |
| | $t_2 = int$ |
| | $t_2 = int$ |

| Equations | Substitution |
| --- | --- |
| $t_p = t_1$ | |
| $t_0 = t_2$ | |
| $t_1 = bool$ | |
| $int = int$ | |
| $t_p = bool$ | |
| $t_2 = int$ | |
| $t_2 = int$ | |

| Equations | Substitution |
| --- | --- |
| | $t_p = t_1$ |
| $t_0 = t_2$ | |
| $t_1 = bool$ | |
| $int = int$ | |
| $t_p = bool$ | |
| $t_2 = int$ | |
| $t_2 = int$ | |

| Equations | Substitution |
| --- | --- |
| | $t_p = bool$ |
| $t_0 = t_2$ | |
| | $t_1 = bool$ |
| $int = int$ | |
| $t_p = bool$ | |
| $t_2 = int$ | |
| $t_2 = int$ | |

معادلات tautology و تکراری را حذف می‌کنیم:

| Equations | Substitution |
| --- | --- |
| | $t_p = bool$ |
| $t_0 = t_2$ | |
| | $t_1 = bool$ |
| $t_2 = int$ | |

| Equations | Substitution |
|---|---|
| | $t_p = bool$ |
| | $t_0 = t_2$ |
| | $t_1 = bool$ |
| $t_2 = int$ | |

| Equations | Substitution |
|---|---|
| | $t_p = bool$ |
| | $t_0 = int$ |
| | $t_1 = bool$ |
| | $t_2 = int$ |

عبارت از نظر type صحیح است و جنس آن int است.

## b

| Expression | Type Variable |
|---|---|
| `f` | $t_f$ |
| `let f = proc (z) z in proc (x) -((f x), 1)` | $t_0$ |
| `z` | $t_z$ |
| `proc (z) z` | $t_1$ |
| `x` | $t_x$ |
| `proc (x) -((f x), 1)` | $t_2$ |
| `-((f x), 1)` | $t_3$ |
| `(f x)` | $t_4$ |

| Expression | Equations |
|---|---|
| `let f = proc (z) z in proc (x) -((f x), 1)` | $t_f = t_1$ |
| | $t_0 = t_2$ |
| `proc (z) z` | $t_1 = t_z \to t_z$ |
| `proc (x) -((f x), 1)` | $t_2 = t_x \to t_3$ |
| `-((f x), 1)` | $t_3 = int$ |
| | $t_4 = int$ |
| | $int = int$ |
| `(f x)` | $t_f = t_x \to t_4$ |

| Equations | Substitution |
| --- | --- |
| $t_f = t_1$ | |
| $t_0 = t_2$ | |
| $t_1 = t_z \to t_z$ | |
| $t_2 = t_x \to t_3$ | |
| $t_3 = int$ | |
| $t_4 = int$ | |
| $int = int$ | |
| $t_f = t_x \to t_4$ | |

حذف tautology:

| Equations | Substitution |
| --- | --- |
| $t_f = t_1$ | |
| $t_0 = t_2$ | |
| $t_1 = t_z \to t_z$ | |
| $t_2 = t_x \to t_3$ | |
| $t_3 = int$ | |
| $t_4 = int$ | |
| $t_f = t_x \to t_4$ | |

| Equations | Substitution |
| --- | --- |
| | $t_f = t_1$ |
| $t_0 = t_2$ | |
| $t_1 = t_z \to t_z$ | |
| $t_2 = t_x \to t_3$ | |
| $t_3 = int$ | |
| $t_4 = int$ | |
| $t_f = t_x \to t_4$ | |

| Equations | Substitution |
| --- | --- |
| | $t_f = t_1$ |
| | $t_0 = t_2$ |
| $t_1 = t_z \to t_z$ | |
| $t_2 = t_x \to t_3$ | |

| Equations | Substitution |
| --- | --- |
| $t_3 = int$ | |
| $t_4 = int$ | |
| $t_f = t_x \to t_4$ | |

| Equations | Substitution |
| --- | --- |
| | $t_f = t_z \to t_z$ |
| | $t_0 = t_2$ |
| | $t_1 = t_z \to t_z$ |
| $t_2 = t_x \to t_3$ | |
| $t_3 = int$ | |
| $t_4 = int$ | |
| $t_f = t_x \to t_4$ | |

| Equations | Substitution |
| --- | --- |
| | $t_f = t_z \to t_z$ |
| | $t_0 = t_x \to t_3$ |
| | $t_1 = t_z \to t_z$ |
| | $t_2 = t_x \to t_3$ |
| $t_3 = int$ | |
| $t_4 = int$ | |
| $t_f = t_x \to t_4$ | |

| Equations | Substitution |
| --- | --- |
| | $t_f = t_z \to t_z$ |
| | $t_0 = t_x \to int$ |
| | $t_1 = t_z \to t_z$ |
| | $t_2 = t_x \to int$ |
| | $t_3 = int$ |
| | $t_4 = int$ |
| $t_f = t_x \to t_4$ | |

| Equations | Substitution |
| --- | --- |
| | $t_f = t_z \to t_z$ |

| Equations | Substitution |
|---|---|
| | $t_0 = t_x \to int$ |
| | $t_1 = t_z \to t_z$ |
| | $t_2 = t_x \to int$ |
| | $t_3 = int$ |
| | $t_4 = int$ |
| | $t_f = t_x \to int$ |

از دو معادله‌ی $t_f$:

$$t_f = (t_x \to int) = (t_z \to t_z)$$

$$\Rightarrow \quad t_z = int, \quad t_x = t_z = int, \quad t_f = (int \to int)$$

| Equations | Substitution |
|---|---|
| | $t_f = int \to int$ |
| | $t_0 = int \to int$ |
| | $t_1 = int \to int$ |
| | $t_2 = int \to int$ |
| | $t_3 = int$ |
| | $t_4 = int$ |
| | $t_z = int$ |
| | $t_x = int$ |

■ تمام

## 3

```
                (type-of e1 tenv) = t
                (type-of e2 tenv) = t
                                  .
                                  .
                                  .
                (type-of en tenv) = t
-----------------------------------------------------
 (type-of (list-exp e1 (e2 ... en)) tenv) = listof t


       (type-of e1 tenv) = t
       (type-of e2 tenv) = listof t
-------------------------------------------
 (type-of (cons-exp (e1, e2)) tenv) = listof t


    (type-of e1 tenv) = listof t
```

```
  ------------------------------------
   (type-of (car-exp (e1)) tenv) = t


         (type-of e1 tenv) = listof t
  ----------------------------------------
   (type-of (cdr-exp (e1)) tenv) = listof t


     (type-of e1 tenv) = listof t
  ---------------------------------------
   (type-of (null-exp (e1)) tenv) = bool



  ------------------------------------------
  (type-of (emptylist-exp t) tenv) = listof t
```

```
#lang eopl

(define type-of
  (λ (exp tenv)
    (cases expression exp
      ; .
      ; .
      ; .
      [list-exp (e1 exps) (let ([ty1 (type-of e1 tenv)])
                            (if (null? exps)
                                (list-type ty1)
                                (let ([first (car exps)])
                                  (check-equal-type! (type-of first tenv) ty1 first)
                                  (type-of (list-exp first (cdr exps)) tenv)))))]
      [cons-exp (e1 e2) (let ([ty1 (type-of e1 tenv)]
                              [ty2 (type-of e2 tenv)])
                              (check-equal-type! (list-type ty1) ty2 e2)
                          ty2)]
      [null-exp (e1) (let ([ty1 (type-of e1 tenv)])
                       (cases type ty1
                         [list-type (ty2) (bool-type)]
                         [else (eopl:error 'type-of "Not a list: ~s" e1)]))]
      [emptylist-exp (ty) (list-type ty)]
      [car-exp (e1) (let ([ty1 (type-of e1 tenv)])
                      (cases type ty1
                        [list-type (ty2) ty2]
                        [else (eopl:error 'type-of "Not a list: ~s" e1)]))]
      [cdr-exp (e1) (let ([ty1 (type-of e1 tenv)])
                        (cases type ty1
                          [list-type (ty2) ty1]
                          [else (eopl:error 'type-of "Not a list: ~s" e1)])))])))
```

### ج

چون در تعریف زبان ما از لیست هر لیست چه خالی چه غیرخالی یک تایپ پایه دارد.

اگر قرار است لیست خالی هم یک لیست حساب کنیم باید یک تایپ پایه داشته باشد مگر این که تغییراتی در زبان انجام دهیم.

در واقع زبان ما تمایزی از جنس type بین لیست خالی و لیست غیرخالی قائل نیست. البته منطقی هم هست چرا که ممکن است خالی بودن یا پر بودن یک لیست در حین اجرا مشخص شود و در صورتی که بخواهیم لیست خالی بدون تایپ داشته باشیم دیگر نمی‌توانیم type-checking را با سخت‌گیری فعلی انجام دهیم.

ما برای هر عبارت و expression یک تایپ مشخص می‌خواهیم و توابع ما هم نمی‌توانند خروجی‌هایی با تایپ‌های مختلف داشته باشند اگر بخواهیم لیست خالی را به عنوان یک تایپ جدید معرفی کنیم آن موقع مثلا تابع `cons` و `cdr` تایپ مشخصی ندارند و به مشکل می‌خوریم.