به نام خدا

پاسخ تمرین‌های سری سوم **Programming Languages**

Concepts in Programming Languages – Mitchell

Q 4.3)

With Renaming:

(λx. λy. xy) (λx. xy) → (λx. λy. xy) (λz. zw) → (λy. (λz.zw) y) → (λy. yw)

Example: (λy. yw) a → aw (w is a free variable in the abstraction)

Without renaming:

(λx. λy. xy) (λx. xy) → (λy. (λx.xy) y) → (λy. yy)

Example: (λy. yy) a → aa (the second y is not a free variable in the abstraction and it is bounded!)

Q 4.4)

a. ((λf. λg. f (g 1)) (λx. x+4)) (λy. 3-y) → (λg. (λx. x+4) (g 1)) (λy. 3-y) →

(λx. x+4) ((λy. 3-y) 1) → (((λy. 3-y) 1) + 4) → 6

b. ((λf. λg. f (g 1)) (λx. x+4)) (λy. 3-y) → (λg. (λx. x+4) (g 1)) (λy. 3-y) →

(λx. x+4) ((λy. 3-y) 1) → (λx. x+4) 2 → 6

Q 4.6)

f ::= λg. g g

main ::= f f = (λg. g g) (λg. g g) → (λg. g g) (λg. g g) → (λg. g g) (λg. g g) → …

This reduction doesn't finish! (divergence)

Q 4.8)

a. C[[x := 1; x := x +1]](s_0) = C [[x:=x+1]] (C [[x:=1]] (s_0))= C [[x:=x+1]] (s_1) = s_2

s_0 = { (x,0) }

s_1 = { (x,1) }

s_2 = { (x,2) }

b. C[[x := 1; x := x +1]](s) = C [[x:=x+1]] (C [[x:=1]] (s))= C [[x:=x+1]] (s_1) = s_2

s = { (x,uninit) }

s_1 = { (x,1) }

s_2 = { (x,2) }

because x initialized at s , so the value of x in s isn't effective in calculation.

Q 4.9)

a. C[[x := 0; y := 0; if x = y then z := 0 else w := 1]](s_0)

= C[[y := 0; if x = y then z := 0 else w := 1]] (C[[x := 0]] (s_0))

= C[[if x = y then z := 0 else w := 1]] (C[[y := 0]](s_1))

= C[[if x = y then z := 0 else w := 1]] (s_2)

= if E[[x=y]](s2) = error or c[[z := 0]](s) = error or c[[w:=1]](s) = error then error

else c[[z := 0]](s_2) + c[[w:=1]](s_2)

= modify(s_0 , x , init) and modify(s_0 , y , init)

z and w is uninit (because both of them aren't initialized in "then" and "else" expression)

s_0 = {(x,uninit),(y, uninit),(z, uninit),(w, uninit)}

s_1 = {(x,0),(y, uninit),(z, uninit),(w, uninit)}

s_2 = {(x,0),(y,0),(z, uninit),(w, uninit)}

b. *C*[[if *x* = *y* then *z* := *y* else *z* := *w*]](*s*)

= if E[[x=y]](s) = error or c[[z := 0]](s) = error or c[[w:=1]](s) = error then error

else c[[z := 0]](s) + c[[w:=1]](s)

= modify(s , x , init) and modify(s , y , init)

z and w is uninit (because both of them aren't initialized in "then" and "else" expression) s = {(x,init) , (y,init) , (z,uninit) , (w,uninit)}

Q 4.13)

a. به دلیل وجود assignment در زبان‌های imperative و داشتن side effect نمی توان ارزیابی موازی داشت و به این دلیل که در ذات زبانهای functional است، این گونه زبانها در سیستم‌های بزرگ موازی (همروند) کاربرد دارد.

b. زبانهای functional سربار اضافه بیشتری به سیستم تحمیل می کند که در مباحث تئوری مطرح نمی شود. در مباحث تئوری به ذخیره شدن state ها در imperative اشاره می شود.

بعضی از ساختمان داده ها با زبان های functional راحت تر کار می کند در حالی‌که همان ساختمان داده با زبان imperative کارهای دیگری باید انجام شود.heavy allocation در زبانهای functional ذاتی است.

همه این موارد باعث می شود تا حجم بیشتری حافظه توسط زبانهای functional مصرف شود.

Instructions were very simple, which made hardware implementation easier

c. باتوجه به فرضیات مطرح شده ، زبانهای functional به executable های بزرگتری بعد از کامپایل نیاز دارند ولی زبانهای imperative به دلیل این که تنها قابلیت assignment و ذخیره state ها را دارند . اما در کاربردهای امروزی باتوجه به این که زبانهای imperative هم قابلیت تعریف higher-order function و هم garbage collector را دارند به executable های بزرگتری نیاز دارند.

d. این مسائل باعث شد تا در انتخاب بین functional و imperative بیشتر به imperative توجه شود و در موارد خاص (مثل محاسبات موازی) به functional روی بیاورند.

e. هنوز هم بعضی از افراد به functional عقیده دارند و اعتقاد دارند که همه برنامه ها را می توان راحت تر در functional نوشت و عده دیگری به زبان های imperative اعتقاد دارند.

Q 4.14)

۱) بعضی از زبانهای functional برای همروندی و موازی‌سازی طراحی نشدند و لذا به طور کلی این قابلیت را ندارند.

۲) ممکن است بعضی از زبانهای functional در ظاهر syntax، expression-based باشند ولی در هنگام implementation، طبق زبانی imperative شود. (از assignmentها برای ایجاد خاصیت استفاده شده)

Types and Programming Languages – Pierce

Q 5.2.2)
scc = λn . λs . λz . n s (s z)

e.g. : scc (c2) → (λn . λs . λz . n s (s z)) c2 → λs . λz . c2 s (s z) → λs . λz . (λs . λz . s (sz)) s(sz) → λs . λz . (λz . s (s z)) (s z) → λs . λz . (s (s (s z))) = c3

Q 5.2.4)
power = λx. λy. x (times y) c1

power2 = λm. λn. m n

Q 5.2.8)
nil = pair tru tru

cons = λh . λt . pair fls (pair h t)

isnil = fst

head = λh . fst (snd h)

tail = λt . snd (snd t)

OR

nil = λc. λn. n
cons = λh. λt. λc. λn. c h (t c n)
head = λl. l (λh. λt. h) fls
tail = λl. fst (l (λx. λp. pair (snd p) (cons x (snd p))) (pair nil nil))
isnil = λl. l (λh. λt. fls) tru

Q 5.2.11)
sum = λm . λn . test (isnil n) (λx . c0) (λx . (plus (head n) (m (tail n)))) c0

sumlist = fix sum

Q 5.3.6)
For beta-reduction :

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \qquad \text{(E-APP1)}$$

$$\frac{t_2 \longrightarrow t_2'}{t_1\ t_2 \longrightarrow t_1\ t_2'} \qquad \text{(E-APP2)}$$

$$(\lambda x.t_{12})\ t_2 \longrightarrow [x \mapsto t_2]t_{12} \qquad \text{(E-APPABS)}$$

It is like those rules, but in judgments, we have term t_1 and t_2, not value v_1 and v_2.
(Because it isn't necessary that is be value.)

For normal-order: (left-most and outermost)

$$\frac{na_1 \longrightarrow na_1'}{na_1\ t_2 \longrightarrow na_1'\ t_2} \qquad \text{(E-APP1)}$$

$$\frac{t_2 \longrightarrow t_2'}{nanf_1\ t_2 \longrightarrow nanf_1\ t_2'} \qquad \text{(E-APP2)}$$

$$\frac{t_1 \longrightarrow t_1'}{\lambda x.t_1 \longrightarrow \lambda x.t_1'} \qquad \text{(E-ABS)}$$

$$(\lambda x.t_{12})\ t_2 \longrightarrow [x \mapsto t_2]t_{12} \qquad \text{(E-APPABS)}$$

na ::= x | t_1 t_2 (for non-abstractions)
nf ::= λx . nf | nanf (for normal forms)
nanf ::= x | nanf nf (for non-abstraction normal form)

For lazy evaluation (or call-by-value):

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2} \qquad \text{(E-APP1)}$$

$$(\lambda x.t_{12})\ t_2 \longrightarrow [x \mapsto t_2]t_{12} \qquad \text{(E-APPABS)}$$

Because this strategy is substitution name and at last evaluate names and this strategy doesn't allow to reduction in abstraction.