Concepts in programming languages

4.3

$$(\lambda x.\lambda y.xy)(\lambda x.xy) \xrightarrow{rename\ bound\ variables} (\lambda k.\lambda m.km)(\lambda x.xy) \rightarrow \lambda m.(\lambda x.xy)m \rightarrow \lambda m.\text{my}$$

If we were to forget to rename bound variables:
$(\lambda x.\lambda y.xy)(\lambda x.xy) \rightarrow \lambda y.(\lambda x.xy)y \rightarrow \lambda y.\text{yy}$

The second reduction is not correct because the variable y is free in $\lambda x.xy$ but becomes bound in $\lambda y.\text{yy}$.

4.4

a)

$$\big((\lambda f.\lambda g.\,f(g\ 1))(\ \lambda x.x+4)\big)\,(\lambda y.3-y) \rightarrow \big(\lambda g.(\ \lambda x.x+4)(g\ 1)\big)(\lambda y.3-y)$$
$$\rightarrow (\ \lambda x.x+4)\big((\lambda y.3-y)\ 1\big) \rightarrow \big((\lambda y.3-y)\ 1\big)+4\big) \rightarrow (3-1)+4=6$$

b)

$$\big((\lambda f.\lambda g.\,f(g\ 1))(\ \lambda x.x+4)\big)\,(\lambda y.3-y) \rightarrow \big(\lambda g.(\ \lambda x.x+4)(g\ 1)\big)(\lambda y.3-y)$$
$$\rightarrow (\ \lambda x.x+4)\big((\lambda y.3-y)\ 1\big) \rightarrow (\ \lambda x.x+4)(3-\ 1) \rightarrow 2+4=6$$

4.6

$f = \lambda f.\,ff$

$f(f)$: $(\lambda f.\,ff)(\lambda g.\,gg) \rightarrow (\lambda g.\,gg)(\lambda g.\,gg) \rightarrow (\lambda g.\,gg)(\lambda g.\,gg)$

این تابع به یک term نمی رسد و دائما به $(\lambda g.\,gg)(\lambda g.\,gg)$ تبدیل می شود.

4.8

a)

$$C\big[[x := 1; x := x+1;]\big](s0) = c[[x := x+1]](c[[x := 1]])$$
$$= c\big[[x := x+1]\big]\big(\text{modify}(s0, x, 1)\big) \xrightarrow{s1=modify(s0,x,1),s1(x)=1} \text{modify}(s1, x, E[[x$$
$$+ 1]](s1)) = \text{modify}(s1, x, 2)$$

b)

$$C\big[[x := 2;]\big](s0) = modify(s, x, 2) \rightarrow C\big[[x := 1; x := x+1;]\big](s0) = C\big[[x := 2;]\big](s0)$$

4.9

a)

$$C[[x := 0;\ y := 0;\ \text{if } x = y \text{ then } z := 0 \text{ else } w := 1\ ]](s0)$$
$$= C[[\ y := 0;\ \text{if } x = y \text{ then } z := 0 \text{ else } w := 1\ ]](C[[x := 0;]](s0))$$
$$\xrightarrow{C[[x:=0;]](s0)=modify(s0,x,0)=s1} C[[\ y := 0;\ \text{if } x = y \text{ then } z := 0 \text{ else } w := 1\ ]](s1)$$
$$= C[[\ \text{if } x = y \text{ then } z := 0 \text{ else } w := 1\ ]](C[[y := 0;]](s1))$$
$$\xrightarrow{C[[y:=0;]](s1)=modify(s1,y,0)=s2} C[[\ \text{if } x = y \text{ then } z := 0 \text{ else } w := 1\ ]](s2)$$
$$= if\ E[[\ x == y]](s2)\ then\ C[[z := 0]](s2)\ else\ C[[w := 1]](s2)$$
$$= if\ E[[\ x == y]](s2)\ then\ modify(s2, z, 0) else\ modify(s2, w, 1)$$
$$= if\ E[[\ 0 == 0]](s2)\ then\ modify(s2, z, 0) else\ modify(s2, w, 1)$$
$$= modify(s2, z, 0)$$

b)

$$C[[\ \text{if } x = y \text{ then } z := \text{y else } z := \text{w}]](s0)$$
$$= if\ E[[\ x == y]](s0)\ then\ C[[z := y]](s0)\ else\ C[[z := w]](s0)$$
$$= if\ E[[\ x == y]](s0)\ then\ modify(s0, z, y) else\ modify(s0, z, w)$$

4.13

a)

خیر – هیچ یک ازویژگی های ذاتی زبان Imperative دلیل اصلی محبوبیت آنها نیست. به عنوان مثال به دلیل این که هسته ی سیستم عامل های کنونی با استفاده از زبان های Imperative نوشته شده اند به همین دلیل توسعه دهندگان با استفاده از همین زبان ها سایر برنامه های کاربردی را تولید می کنند. به عنوان مثال محبوبیت زبان C با وجود مشکلات زیادی که ایجاد کرده است (buffer overflow و ...) بیشتر به خاطر Unix است. علاوه بر این، زبان هایی garbage collected هستند در صورت عدم نیاز برنامه به این ویژگی سربار دارند.

b)

زبان های Imperative به دلیل این که در زمان اجرا به منابع کمتری نیاز دارند برای سیستم هایی که منابع محدودی دارند مناسب تر هستند.

c)

زبان های functional به دلیل این که در زمان اجرا garbage collection انجام می دهند فایل های اجرایی آن ها حجم بیش تری دارند.

d)

همانطور که پیش تر اشاره شد زبان های Imperative به دلیل عدم پشتیبانی از garbage collection و با توجه به عدم استفاده بهینه از حافظه مشکلاتی ایجاد می کردند. از طرف دیگر اضافه کردن الگوریتم های garbage collection به حافظه ی بیشتری نیاز داشت.

e)

امروزه با افزایش حجم حافظه ها و پیشرفت سخت افزار دیگر نگرانی های قبلی وجود ندارد ولی هنوز استفاده بهینه از منابع سیستم مطرح است.

4.14

1- اگر تعداد ترد ها (سطح موازی سازی) زیاد باشد سربار زیادی دارد (زمان context switch خیلی زیاد می شود و به صرفه نیست)

2- زبان های functional در استفاده از ساختار داده هایی مانند آرایه های یک بعدی بهینه نیستند و در صورتی که برنامه از این ساختار داده ها زیاد استفاده کند عملکرد خوبی نخواهد داشت.

Types and programming languages

5.2.2

$succ = \lambda n.\lambda s.\lambda z\ s(n\ s\ z)$

$plus\ 1\ n = \lambda n.\lambda s.\lambda z\ (\lambda s.\lambda z\ (s\ z))\ s\ (n\ s\ z) = \lambda n.\lambda s.\lambda z\ s(n\ s\ z) = succ\ n$

5.2.4

$Pow = \lambda m.\lambda n.\lambda m\ (times\ n)\ c_1$

در این تابع m بار times n انجام می شود. اولین بار در c_1 ضرب می شود و در مراحل بعدی در نتیجه ی مرحله قبلی.

5.2.8

$nil = \lambda c.\lambda n.\ n$

$list = \lambda c.\lambda n\ c\ x_1\ (c\ x_2(c\ x_3(\dots(c\ x_k\ n)))$

$isnil = \lambda m.\ m\ (\lambda x\ fls)\ tru$

$cons = \lambda h.\lambda m.\lambda c.\lambda n\ c\ h\ (m\ c\ n)$

در head اگر به جای c ، true قرار بگیرد مولفه ی اول لیست را بر میگرداند ولی وقتی لیست خالی است fls برمی گردد.

$Head = \lambda m.\ m\ tru\ fls$

$tail = \lambda m.\ fst\ (m\ (\lambda x.\lambda p.\ pair\ (snd\ p)\ (cons\ x\ (snd\ p)))(pair\ nil\ nil))$

5.2.11

$F = \lambda f.\lambda l\ (\text{if}(\text{isnil l})\ then\ (\lambda x.\, c0)\ else\ \left(\lambda x.\left(\text{plus}(\text{head l})\left(f(\text{tail l})\right)\right)\right) c0$

Fixed point تابع F همان تابع sumlist است.

$sumlist = \lambda F.\,(\lambda x.\,F(xx))(\lambda x.\,F(xx))$

5.3.6

Full beta reduction

$$\frac{t_1 \rightarrow t_1'}{t_1 t_2 \rightarrow t_1' t_2}$$

$$\frac{t_2 \rightarrow t_2'}{t_1 t_2 \rightarrow t_1 t_2'}$$

$$(\lambda x.\, t_{12}) t_2 \rightarrow [x \rightarrow t_2] t_{12}$$

Normal-order

$$\frac{na_1 \rightarrow na_1'}{na_1 t_2 \rightarrow na_1' t_2}$$

$$\frac{t_2 \rightarrow t_2'}{nanf_1\ t_2 \rightarrow nanf_1\ t_2'}$$

$$\frac{t_1 \rightarrow t_1'}{\lambda x.\, t_1 \rightarrow \lambda x.\, t_1'}$$

$$(\lambda x.\, t_{12}) t_2 \rightarrow [x \rightarrow t_2] t_{12}$$

Lazy evaluation

$$\frac{t_1 \rightarrow t_1'}{t_1 t_2 \rightarrow t_1' t_2}$$

$$(\lambda x.\, t_{12}) t_2 \rightarrow [x \rightarrow t_2] t_{12}$$