PL Final solution

1 a) [Chap05, p27]

Pro

Type safe: Type errors are all detected at compile time

Efficiency: No type checking code at run time

Con

Inflexible: Variables have fixed types, e.g. fn x => x x is illegal in ML.

b) [Chap10, p3]

Subprograms cannot be nested.

All variables are static (thus, recursion isn't allowed).

c) [Chap05, p22]

They are not always equivalent. For examples

let val x=2 in x end;

= (fn x=>x) 2;

But,

let val f=fn x=> x in (f 3,f true) end;

≠ (fn f=>(f 2,f true)) (fn x=>x)

Because, the formal is typable, but the latter isn't.

d) [Chap09,pp16,19]

Context-independent overloading, Context-dependent overloading

Parameterized polymorphism, Inclusion polymorphism

e) [Chap09,p38]

A piece of code to implement delayed evaluation is called a thunk. More precisely, a by-name or -need actual parameter is passed as a parameter- less function, called a *thunk*.

[Chap12,p25]

A piece of code used to implement virtual function call is called a *thunk*. More precisely, the virtual table contains code pointers and offsets for virtual functions.

f) It outputs 2 and stops, because Perl adopts dynamic scoping for labels. Thus, the returned label **`z`** isn't the label contained in **`A`**. Instead, it is the label **`z`** found in the caller of **`B`**.

2 a) actual parameters have no side effects

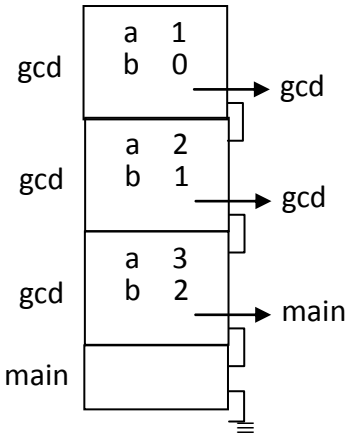
b) formal parameters aren't modified

c) call-by-name doesn't change the bindings of formal and actual parameters

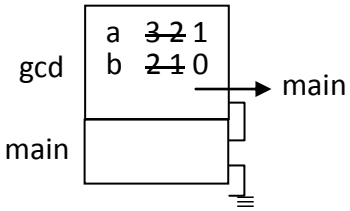
d) call-by-reference doesn't introduce aliasing

e) the function returns normally

3 a)



b)



c)

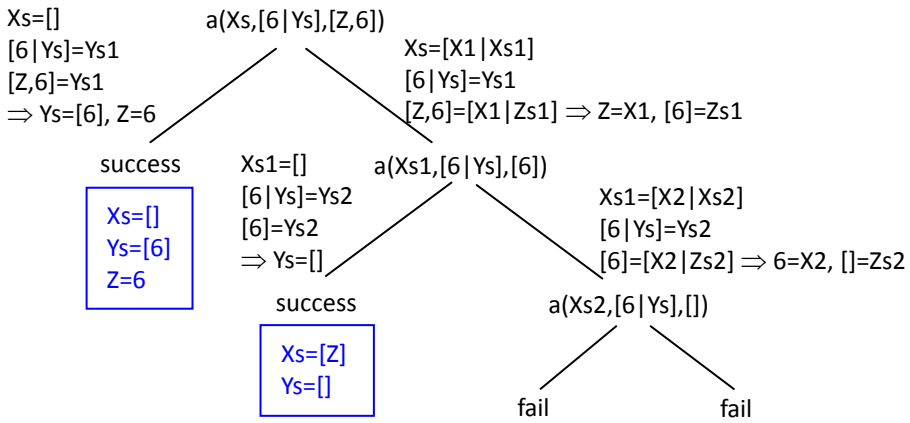
```
int gcd(int a,int b)
{
entry:
    if (b==0) return a;
    else { int c=a; a=b; b=c%b; goto entry; }
}
```

4 See HW4 a)

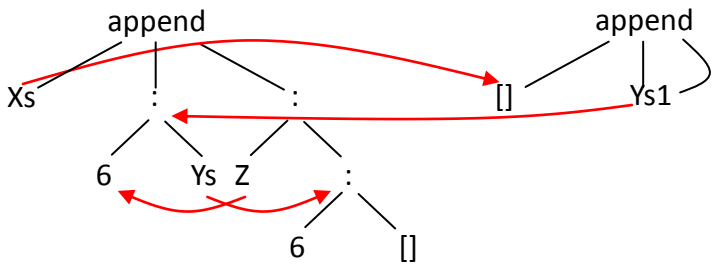
5 a) For simplicity, let's rename append as a

a([],Ys,Ys).

a([X|Xs],Ys,[X|Zs]) :- a(Xs,Ys,Zs).

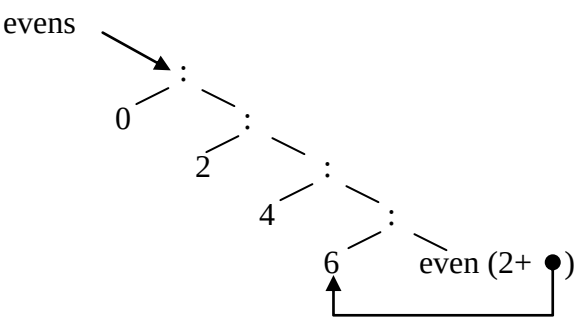


5 b)

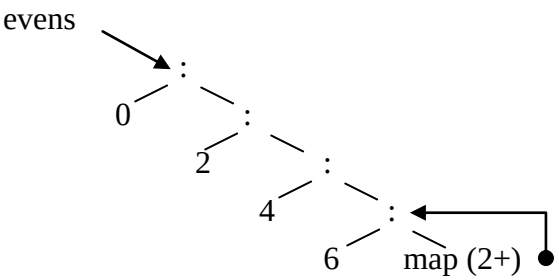


6 a) With these definitions, the element $2x$ is generated by a multiplication. The definition of part b) take the advantage of already-generated element $2(x-1)$ to generate $2x$ by an addition $2+2(x-1)$.

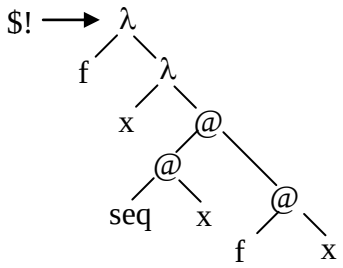
b)
```
evens = [ x | x <- even 0] where even n = n : even (2+n)
```



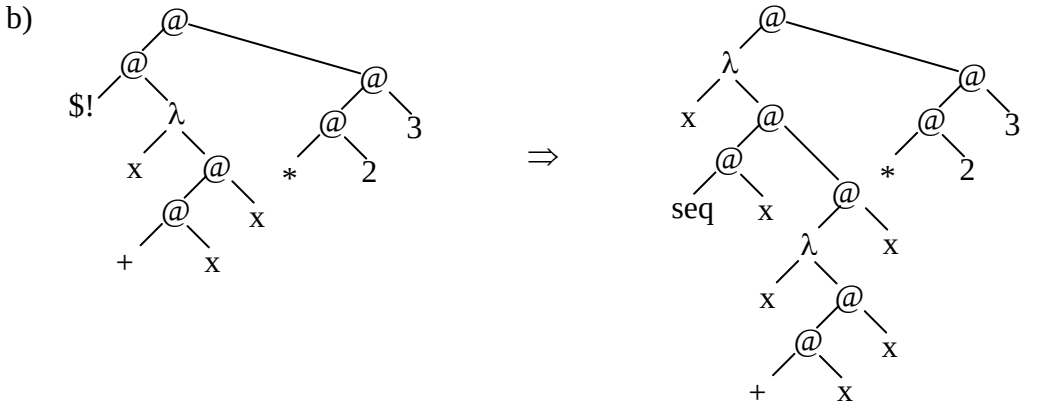
c)
```
evens = 0 : [2+x | x<-evens]
```

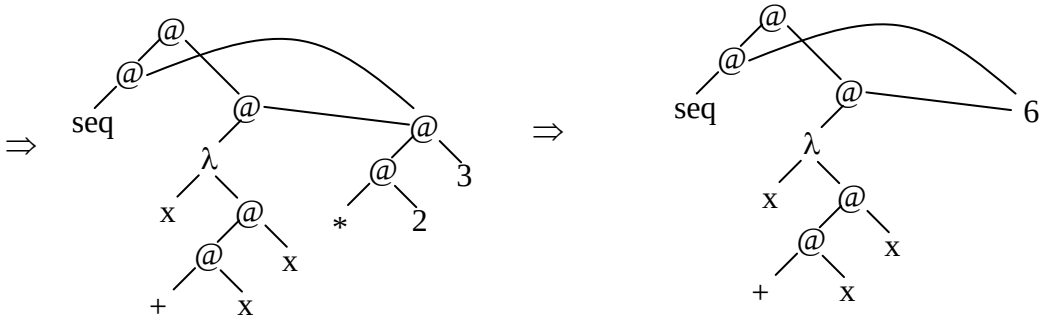


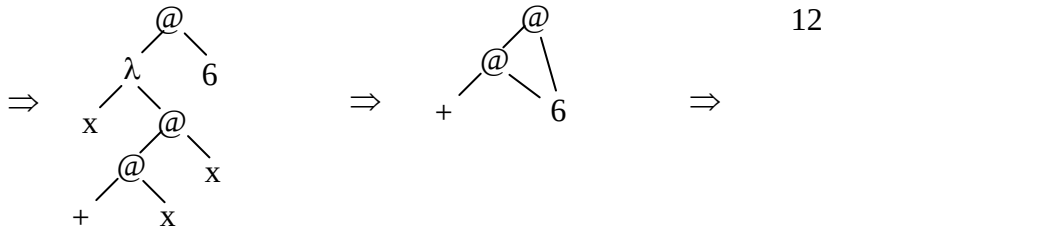
7 a)


$!
λ
f
λ
x
@
@
@
seq
x
@
f
x

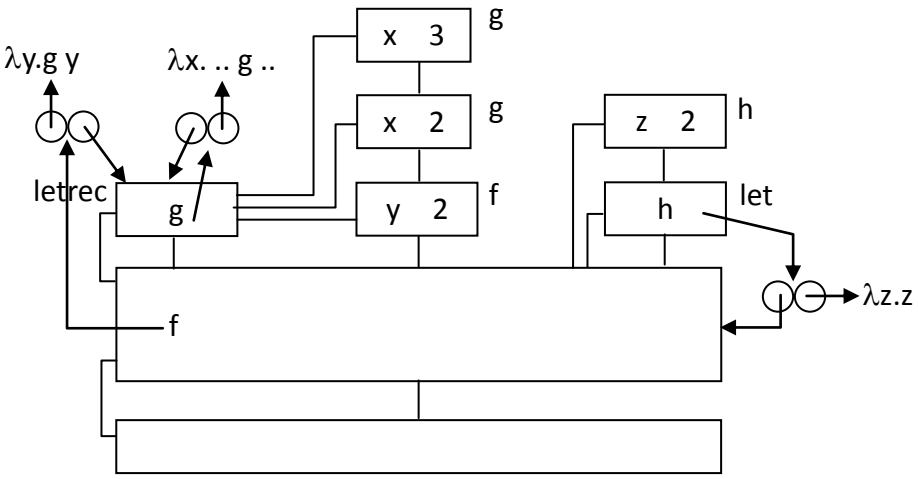
b)


@
@
$!
λ
@
3
x
@
*
2
@
x
+
x
⇒
@
λ
@
x
@
@
3
seq
x
@
*
2
λ
x
x
@
@
x
+
x

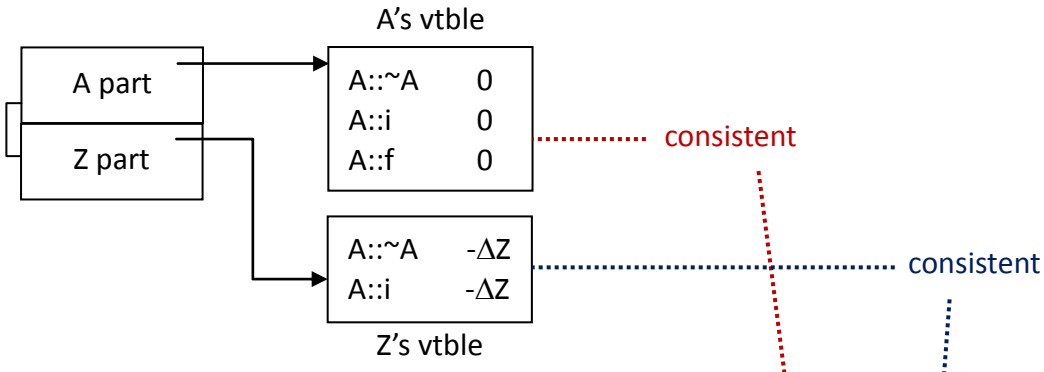

⇒
@
@
seq
@
@
3
λ
x
@
*
2
@
x
+
x
⇒
@
@
seq
@
6
λ
x
@
@
x
+
x


⇒
@
λ
6
x
@
@
x
+
x
⇒
@
@
+
6
⇒
12

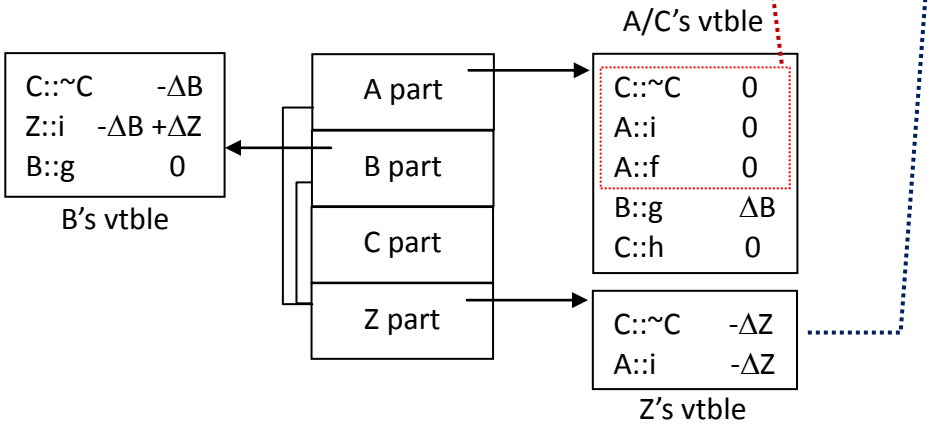
8


λy.g y
λx. .. g ..
letrec
g
x 3
g
x 2
g
y 2
f
z 2
h
h
let
λz.z
f

9 a)


A's vtble
A part
Z part
A::~A 0
A::i 0
A::f 0
consistent
A::~A -ΔZ
A::i -ΔZ
Z's vtble
consistent

b)


A/C's vtble
C::~C -ΔB
Z::i -ΔB +ΔZ
B::g 0
B's vtble
A part
B part
C part
Z part
C::~C 0
A::i 0
A::f 0
B::g ΔB
C::h 0
C::~C -ΔZ
A::i -ΔZ
Z's vtble