# EP Exercises 详解 -- 期中篇（Lambda Calculus）

## Exercise1

### Q1

**Write as a λ-term the function that takes one input and maps it to the constant 0 as λ-term.**

**Solution:**

```
λx. 0
```

---

### Q2

**Write as a λ-term the function that takes two inputs and maps them to the constant 0.**

**Solution:**

```
λx. λy. 0
```

---

### Q3

**Write the following λ-term in the notation with all parentheses and all λs:**

(λxy. x (y z)) λx. y x x.

**Solution:**

```
((λx.(λy.(x (y z)))) (λx.((y x) x)))
```

---

### Q4

**Write the following λ-term in the notation with as few parentheses and λs as possible:**

((λx.(λy.((((x y) z) u) (v w)))) (λx.(y (x x)))).

**Solution:**

```
(λx. λy. x y z u (v w)) λx. y (x x)
```

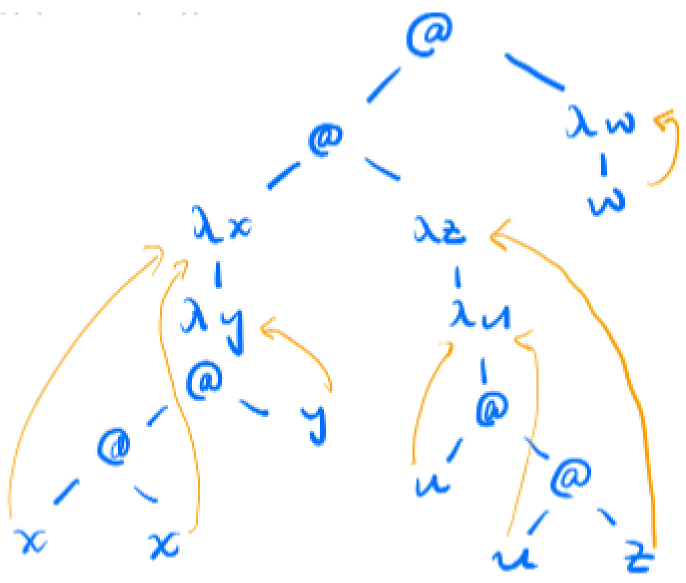
---

## Q5

**Draw the term tree of the following λ-term, and indicate which variables are bound, and by which λ they are bound:**

(λxy. x x y) (λzu. u (u z)) λw. w

**Solution:**



---

## Q6

**Indicate for all pairs of λ-terms below whether or not they are α-convertible:**

(a) x and y,

(b) λx. x x and λy. y y,

(c) λu. u u' and λu' . u' u',

(d) (λxyz. y (x y z)) λvw. v w and (λx'yz' . y (x ' y z' )) λvw' . v w' ,

(e) (λx. x) x and (λy. y) y,

### Definition

**α-convertible**:

- Two terms M and N are said to be α-equivalent if they are identical up to the renaming of bound variables.
- The renaming of bound variable is called α-convertible.

**Solution**

(a) x and y

```
not α-convertible
```

- variable 不可以随意改名

(b) λx. x x and λy. y y

```
α-convertible
```

- bound 的 variable 必须随绑定的 λ 一起改名

(c) λu. u u' and λu' . u' u'

```
not α-convertible
```

- and 左边的 u' 是 free variable (argument)，不能与 λ 一致。

(d) (λxyz. y (x y z)) λvw. v w and (λx'yz' . y (x ' y z' )) λvw' . v w'

```
α-convertible
```

(e) (λx. x) x and (λy. y) y

```
not α-convertible
```

- 左边返回 x， 右边返回 y，不一致。

---

## Q7

**Calculate the following substitutions:**

(a) (x y)[y := u w]

(b) (x y)[x := u w]

(c) (λx. x y)[x := u w]

(d) (λx. x y)[y := u w]

(e) (λx. x y)[y := x x]

(f) ((λx. x) x (λy. x u))[x := y]

## Definition

**β-reduction:**

- (λx. M) N $\to_\beta$ M[x := N]
- The expression on the left-hand side of the arrow means: the term M in which all **free** occurrences of *x* are replaced by the term N.

**Substitution:**

- (λx. M) N $\to_\beta$ M[x := N]
- 单独的 substituion 是指对 abstraction body （这里是 M）根据 β-reduction rule （箭头右侧）把 body 进行替换的行为。
- 注意，如 Q7 所示的题型，题干都只是 $\to_\beta$ 右侧的一部分，而不是完整的 λ-term, 注意不要混淆。

**Substitution遵循两个原则：**

- 只替换 abstraction body 里的 free variable；
- 如果替换后和 bound variable 重名，则需给 bound variable 和其绑定的 lambda 改名。

## Solution：

(a) (x y)[y := u w]

```
(x y)[y := u w] = x (u w)
```

- the parentheses are needed

(b) (x y)[x := u w]

```
(x y)[x := u w] = (u w) y = u w y
```

(c) (λx. x y)[x := u w]

```
(λx. x y)[x := u w] = λx. x y
```

(d) (λx. x y)[y := u w]

```
(λx. x y)[y := u w] = λx. x (u w)
```

(e) (λx. x y)[y := x x]

```
(λx. x y)[y := x x] = λx'. x' (x x)
```

(f) ((λx. x) x (λy. x u))[x := y]

```
((λx. x) x (λy. x u))[x := y] = (λx. x) y (λy'. y u)
```

---

## Q8

**Indicate whether the following λ-terms are in β-normal form:**

(a) 5 (λx. x x) 3,

(b) λx.(λy. y) 3,

(c) (λx. x) (λy. y) x,

(d) λx. x ((λy. y) x)

### Definition

**β-redex:**

- β-redex is defined as term of the form **(λx. M) N**
    - redex - stands for reducible expression
    - reducible - 指可以进行 β-reduction

**β-normal form:**

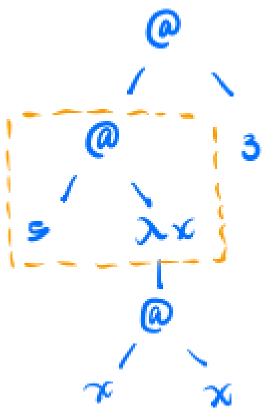
- A λ-term that does not contain a β-redex is said to be a normal form of β-normal form.

**β-reduction sequence:**

- A sequence that consisting of zero, one, or more β-reduction steps
- Notice that, if exactly one β-reduction step, we use the notation $\rightarrow_\beta$
- $\rightarrow_\beta *$ stands for zero, one or more steps
- $\rightarrow \rightarrow$ stands for two steps

### Solution:

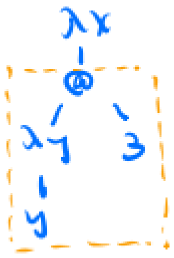
(a) 5 (λx. x x) 3



There exists no β-redex, so is in β-normal form.

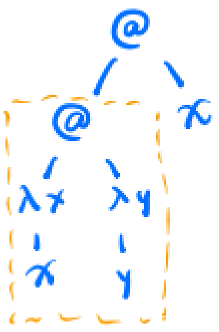
- 注意橙色框部分，application 左边是 constant / varibale 且右边是 normal form的情况下，可以判定此 @ 是 normal form，因为任何 λ-term 都不能对 constant / variable 进行带入。

(b) λx.(λy. y) 3



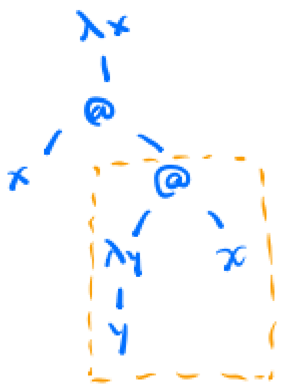
Not in β-normal form.
The term can be reduced to λx. 3

(c) (λx. x) (λy. y) x



Not in β-normal form.
The term has the following β-reduction to normal form:
(λx. x) (λy. y) x $\rightarrow_\beta$ (λy. y) x $\rightarrow_\beta$ x

(d) λx. x ((λy. y) x)



Not in β-normal form.
The term can be reduced to β-normal form:
λx. x ((λy. y) x) →β λx. x x

---

## Q9

**Consider the function double in Haskell:**

```
double x = x + x
```

**Use equational reasoning to compute double (double 4). Explain every step. Give at least one alternative computation.**

### Solution

double (double 4) = {unfold inner double}
double (4 + 4) = {do built-in addition}
double 8 = {unfold double}
8 + 8 = {do built-in addition}
16

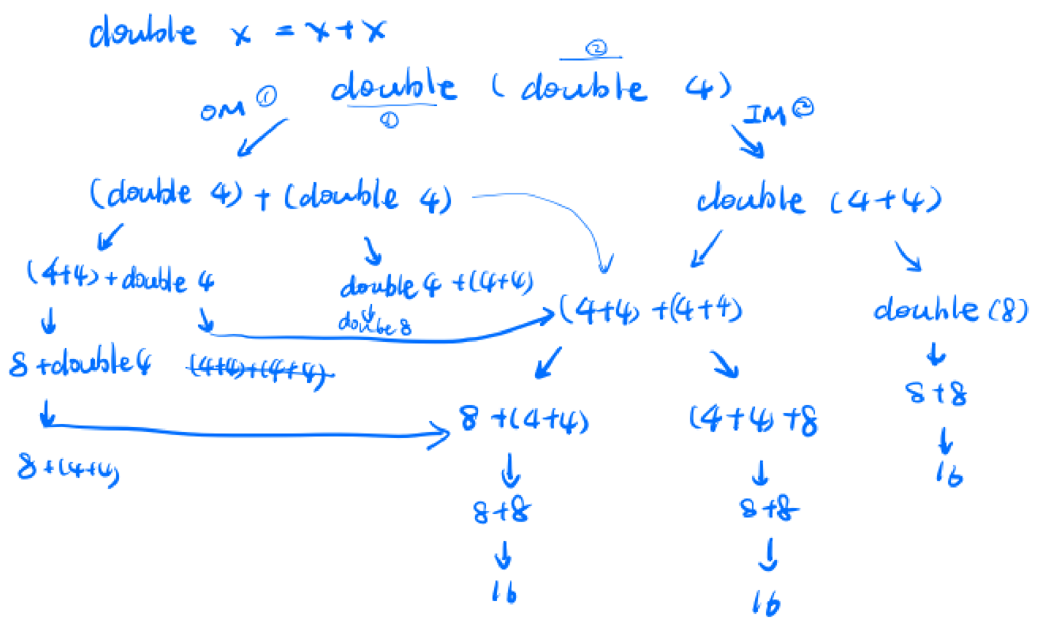
**An alternative computation:**

(double 4) + (double 4) = {unfold left double}

double (double 4) = {unfold outer double}
(double 4) + (double 4) = {unfold left double}
(4 + 4) + (double 4) = {do built-in addition}
8 + (double 4) = {unfold double}
8 + (4 + 4) = {do built-in addition}
8 + 8 = {do built-in addition}
16

**Extension**

**Reduction Graph**:



---

## Q10

**Depict the following terms as trees, and clearly indicate all β-redexes.**

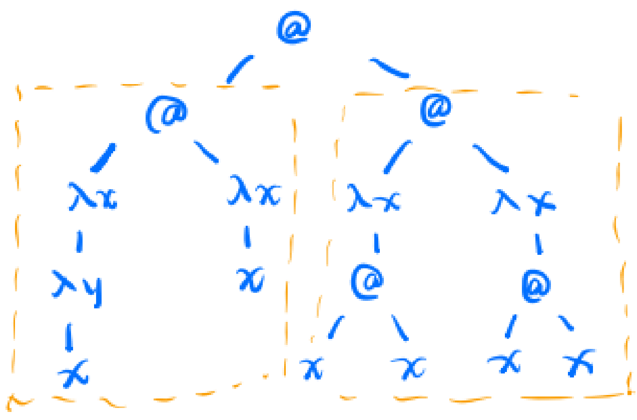
(a) K I Ω ;

(b) M1 = (λf. f 3) (λx.(λy. y) x);

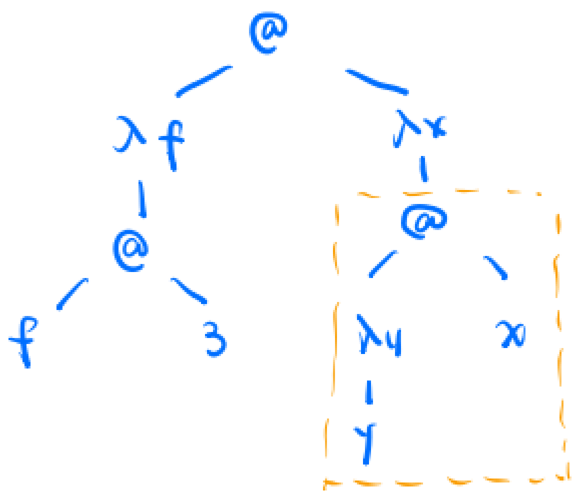
(c) M2 = (λx.((λu. u) x) (λz. x Ω)) λy. z

**Solution:**

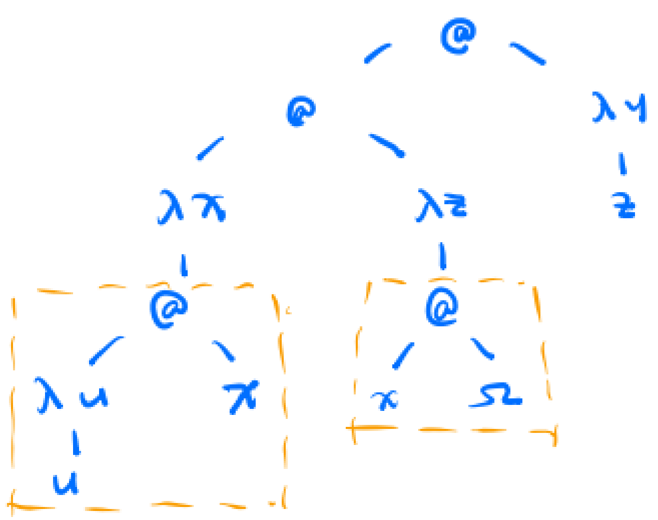
(a) $KI\Omega = (\lambda x.\lambda y.x)(\lambda x.x)(\lambda x.xx)(\lambda x.xx)$



(b) $M_1 = (\lambda f. f\ 3)(\lambda x. (\lambda y. y) x)$



(c) $M_2 = (\lambda x. ((\lambda u. u) x)(\lambda z. x\Omega))\lambda y. z$



---

## Q11

**(a) Give λ-terms M and N such that M and N are β-normal forms but M[x := N] is not a β-normal form.**

### Solution

Take for example

M = x y,

N = λu. u.

Then M and N are both in β-normal form,

but

M[x := N] = (λu. u) y -> y

is not in β-normal form.

**(b) Give λ-terms M and N such that M and N both have a β-normal form, but M[x := N] does not have a β-normal form.**

## Definition

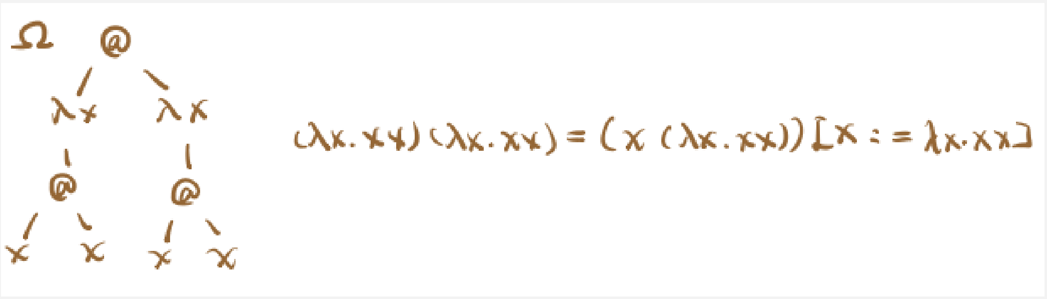
**Has normal form:**

- We say that a term has a normal form if it can be β-reduced in zero, one or more steps to a normal form.
- A normal form itself can also be considered as normal form.

Example:

- x has a normal form
- (λu.u )y has a normal form
- (λx.λy.x)w has a normal form

Notice that Ω does not have a normal form, since every reduction gets the same result.



## Solution

Take for example

M = x (λu. u u),

N = λu. u u

Then

M and N are in β-normal form (so in particular they have a β-normal form),

but

M[x := N] = (λu. u u) (λu. u u) = Ω does not have a β-normal form.

**(c) Give terms M, N, P such that (M[x := N])[y := P] ≠ (M[y := P])[x := N].**

Take for example
M = x,
N = y,
P = z.
Then
(M[x := N])[y := P] = P = z
whereas
(M[y := P])[x := N] = N = y.

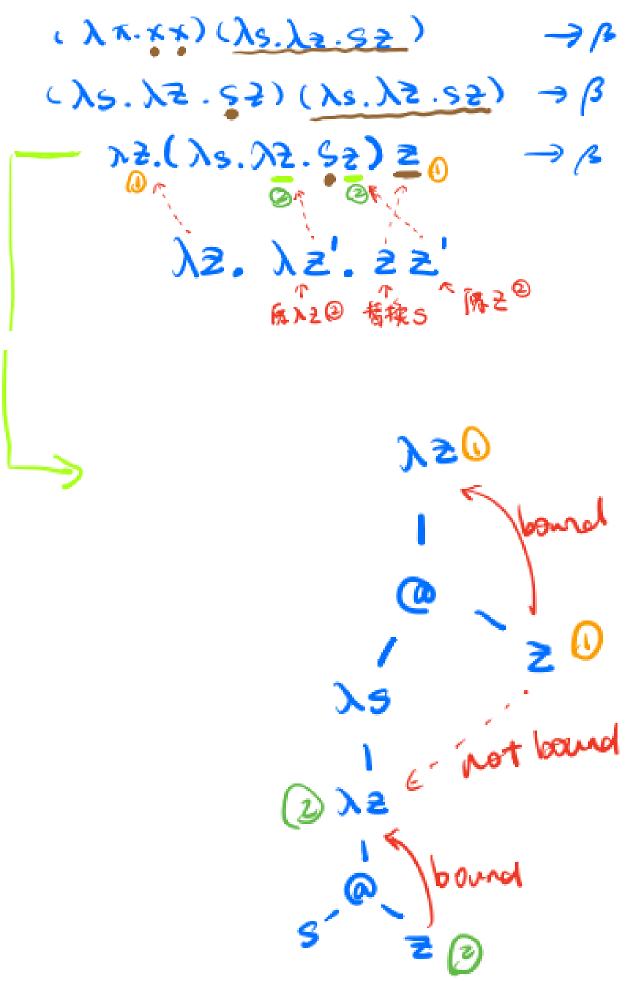
---

## Q12

**Reduce (λx. x x) (λs.λz. s z) to β-normal form.**

**Solution:**

(λx. x x) (λs.λz. s z) →β
(λs. λz. s z) (λs. λz. s z) →β
λz.(λs. λz. s z) z →β
λz. λz'. z z'



而x 替换成 (λs. λz.sz)

s 替换成 (λs.λz.sz)

s 替换成 z①
但这里 替换之后，
z① 和 λz② 是不应该
绑定的，所以λz② 和
z② 需改名.

## Q13

**Are the following terms terminating and/or weakly normalizing?**

(a) (λx. x x)(λfy.(f y))

(b) (λx. x Ω) (λy. z)

(c) (λx. x Ω) (λy. y)

(d) (λx. x x x) (λx. x x x)

## Definition

**Terminating / Strongly normalizing:**

- A term P is said to be terminating or strongly normalizing (SN) if **all** reduction sequences starting in P are finite.
- That is: P is not the start of an infinite reduction.
- β-normal forms are terminating.
- Notice that a term that is not strongly normalizing has a an infinite reduction sequence.

Example:

- (λx. y) ((λx. x) z)
- (λx. x) (λy. y)

**Weakly normalizing:**

- A term that admits a reduction sequence to normal form, we also say that the term has a normal form, is said to be weakly normalizing (WN).
- A strongly normalizing term is weakly normalizing.
- There are weakly normalizing terms that are not strongly normalizing.

Example:

- (λx. y) Ω $\to_\beta$ y  (WN not SN because can also be reduced as: (λx. y) Ω $\to_\beta$ (λx. y) Ω $\to_\beta$ . . .. )

## Solution

(a) (λx. x x)(λfy.(f y))

(λx. x x)(λfy.(f y)) is strongly normalizing and hence also weakly normalizing because all its reductions are finite.
We have
    (λx. x x)(λf. λy.(f y)) →β (λf. λy.(f y)) (λf. λy.(f y)) →β
    λy.(λf. λy.(f y)) y →β
    λy' λy. y' y
which ends in normal form, and there is no other reduction sequence possible.
So self-application does not necessarily leads to non-termination.

(b) (λx. x Ω) (λy. z)

(λx. x Ω) (λy. z) is weakly normalizing because it has a reduction to β-normal form:
    (λx. x Ω) (λy. z) →β (λy. z) Ω →β z.

The term (λx. x Ω) (λy. z) is not strongly normalizing,
because it has Ω as subterm, which reduces in one step to itself.

(c) (λx. x Ω) (λy. y)

(λx. x Ω) (λy. y) is not weakly normalizing and hence not strongly normalizing.
The reducts of (λx. x Ω) (λy. y) are (λx. x Ω) (λy. y) itself, and (λy. y) Ω, and Ω.

We can see this by reducing the term using the leftmost-outermost reduction strategy
    (λx. x Ω) (λy. y) →β (λy. y) Ω → Ω → Ω → . . ..
If the leftmost-outermost reduction strategy yields an infinite reduction, then the initial term does not have a normal form.

(d) (λx. x x x) (λx. x x x)

(λx. x x x) (λx. x x x) is not weakly normalizing and hence not strongly normalizing.
Using the
    A = λx. x x x,
the reduction sequence is
A A →β A A A →β A A A A →β . . .¿
(non-termination)

# Exercise 2

## Q1

**Show that Curry's fixed point combinator**

λf.(λx. f (x x)) (λx. f (x x))

**and Turing's fixed point combinator**

T = (λx. λy. y (x x y)) (λx. λy. y (x x y))

**are indeed fixed point combinators.**

### Definition

**Fixed point:**

- A fixed point of a function f is an element x from the domain of f for which the following holds: f(x) = x.
- In general a function may have zero, one or more fixed points. For example, $x \to\ x^2$ has besides 0 also 1 as fixed point.
- An important result in the λ-calculus is that **every term has a fixed point**.

**Fixed point combinator:**

- A method to, given a λ-term F, construct a fixed point for F. This is done using a so-called fixed point combinator.
- The fixed point combinator due to Curry is the term Y defined as follows:
  - Y = λf.(λx. f (x x)) (λx. f (x x))
  - F (Y F) =β Y F
  - Notice, here we only have conversion, no reduction.
- The following fixed point combintator is due to Turing:
  - T = (λx. λy. y (x x y)) (λx. λy. y (x x y))
  - We write T = t twith t = λx. λy. y (x x y).
  - TF →β F (tt F ) = F (TF )

### Solution:

We show:

for every λ-term F we have Y F =β F (Y F).

Let

F be an arbitrary λ-term.

We have :

Y F = (λf.(λx. f (x x)) (λx. f (x x))) F

→β (λx. F (x x)) (λx. F (x x))

→β F ((λx. F (x x)) (λx. F (x x)))

←β F ((λf.(λx. f (x x)) (λx. f (x x))) F)

= F (Y F)

So indeed

F (Y F) =β Y F

Now we consider Turing’ s fixed point combinator T = (λx. λy. y (x x y)) (λx. λy. y (x x y)).

We aim to show:

for every λ-term F we have TF =β F (TF ).

Take

an arbitrary λ-term F.

We use

t = λx. λy. y (x x y).

We have:

TF = (λx. λy. y (x x y)) (λx. λy. y (x x y)) F

→β (λy. y (t t y)) F

→β F (tt F )

= F (TF )

So we indeed

TF =β F (TF )

We even have

TF →∗β F (TF )

Remark:

if we have to show that a λ-term P is a fixed point combinator, we have to show that P F =β= F (P F) for every λ-term F.

So the reasoning starts with taking an arbitrary λ-term.

## Q2

**Reduce Y (λa. b) and T(λa. b) to obtain a fixed point of λa. b.**

**Solution**

We consider Y (λa. b):

$$Y\,(\lambda a.\,b) = (\lambda f.(\lambda x.\,f\,(x\,x))\,(\lambda x.\,f\,(x\,x)))\,(\lambda a.\,b)$$
$$\to_\beta (\lambda x.(\lambda a.\,b)\,(x\,x))\,(\lambda x.(\lambda a.\,b)\,(x\,x))$$
$$\to_\beta (\lambda a.\,b)\,((\lambda x.(\lambda a.\,b)\,(x\,x)))\,((\lambda x.(\lambda a.\,b)\,(x\,x)))$$
$$\to_\beta b$$

Alternatively:
we use that we already know

$Y\,F =_\beta F\,(Y\,F)$, for any F, from the previous question.

That gives,
taking

λa. b for F:

$$Y\,(\lambda a.\,b) =_\beta (\lambda a.\,b)\,(Y\,(\lambda a.\,b)) \to_\beta b$$

Here we get β-equality.

For the question show β-equality of Y (λa. b) and b both approaches are correct.
For the question reduce Y (λa. b) only the first approach is correct.

But of course the point is to see that for the specific example of the function that gives back b for any input,
the term b is a fixed point, which is indeed found using the fixed point combinator Y.

Now we consider T(λa. b):
first explicitly,
using again

$t = \lambda x.\,\lambda y.\,y\,(x\,x\,y)$

first explicitly:

$$T(\lambda a.\,b) = (\lambda x.\,\lambda y.\,y\,(x\,x\,y))\,(\lambda x.\,\lambda y.\,y\,(x\,x\,y))\,(\lambda a.\,b)$$
$$\to_\beta (\lambda y.\,y\,(t\,t\,y))\,(\lambda a.\,b)$$
$$\to_\beta (\lambda a.\,b)\,(tt(\lambda a.\,b)\,)$$
$$\to_\beta b$$

Alternatively, and faster:
we know from the previous question that

$TF \to_{*\beta}$

$F\,(TF\,)$ for any F.

So we have

```
   T(λa. b) →*β (λa. b) (T(λa. b) ) →β b
```

- 注意，在上面的解法中，我们使用了 abbreviation 替换（t = λx. λy. y (x x y)）来方便计算，使用这种方法时注意被替换的 term 最好是 closed term （不包含 free variable 的 term），否则会很麻烦。比如，如果 term 中包含一个 free variable y，那么我们替换时，需要写做 t(y) = ... 表示 y 是一个 argument，替换后的项会跟着 argument 变动。
- 注意上面解法中有出现 $\rightarrow_\beta *$ 和 $\rightarrow_\beta$，这代表精确的reduction步骤数，具体解释参考 Exercise 1 -> Q8 -> Definition -> β-reduction sequence。

---

## Q3

**Suppose P is a fixed point combinator (we do not know which one).**

**Show that P (S I) is also a fixed point combinator.**

**We use S = λx. λy. λz.(x z) (y z) and I = λx. x.**

### Solution

```
We use Assume a fixed point combinator P,
so we know that
   P F =β F (P F) for any F.
We aim to show that
   P (S I) is a fixed point combinator.

Therefore, we take
   an arbitrary λ-term F,
and we aim to show
   (P (S I)) F =β F ((P (S I)) F),
   or,
   using fewer parentheses:
      P (S I) F =β F (P (S I) F).
We have:
   P (S I) F =β (S I) (P (S I)) F
         = (λx. λy. λz.(x z) (y z))I(P (S I)) F
         →*β (I F) ((P (S I)) F)
         →β F ((P (S I)) F)

In the first step we use that P is a fixed point combinator.
In the second step we unfold the definition of S.
In the third step we do three β-reduction steps.
```

In the fourth step we reduce I F to F in one step.

So indeed
  P (S I) F =β F (P (S I) F).

- This exercise illustrates that from a fixed point combinator we can build another fixed point combinatory by applying it to S I.
- With 'another’we mean‘not β-equal’.
- Note that we did not show that they are not β-equal.

---

## Q4

**Give the reduction graphs of the following terms:**

(a) (I x) (I x);

(b) I(I I);

(c) (λx.x x)(I z);

**Solution:**

(a) (I x) (I x);

$$\begin{tikzcd}
 & (Ix)(Ix) \arrow[dl, "left"'] \arrow[dr, "right"] & \\
x(Ix) \arrow[dr] & & (Ix)x \arrow[dl, "what\ remained\ of\ r."] \\
 & x\,x &
\end{tikzcd}$$

(b) I(I I);

① 
$$\begin{tikzcd}
 & I\,(\underline{I\,I}) \arrow[dl] \arrow[dr] & \\
\underline{I\,I} \arrow[dr, bend right] & & I\,I \arrow[dl, bend left] \\
 & I\,I \arrow[d] & \\
 & I &
\end{tikzcd}$$

② 
$$\begin{tikzcd}
I\,(I\,I) \arrow[d, bend right] \arrow[d, bend left] \\
I\,I \arrow[d] \\
I
\end{tikzcd}$$

- both are correct

- similar as the following example:

① $\Omega^2$ ② $\Omega \to \Omega \to \Omega^2$

(c) (λx.x x)(I z);

$$\begin{tikzcd}
 & & (\lambda x.xx)(Ix) \arrow[dl] \arrow[dr] & \\
 & (Ix)(Ix) \arrow[dl] \arrow[dr] & & (\lambda x'.x'x')x \arrow[ddll, bend left] \\
x(Ix) \arrow[dr] & & (Ix)x \arrow[dl] & \\
 & xx & &
\end{tikzcd}$$

---

## Q5

与 Exercise 1 -> Q13 一致 （同一题）

---

## Q6

**Consider the term M = (λx.((λu. u) x) (λz. x Ω)) λy. z.**

(a) Depict the term tree of M. (Unfold the definition of Ω.)

(b) Give two different α-equivalent renderings of M.

(c) Reduce M using the leftmost-innermost (call-by-value) strategy.

(d) Reduce M using the leftmost-outermost (call-by-need) strategy.

**(没有答案 + 课上没讲， 待补充)**

---

## Q7

**Consider the term**

(λx. λy. x (I I)) (λz. v z) Ω.

**Reduce is according to the leftmost-innermost strategy, according to the leftmost-outermost to normal form (if possible),**

**and according to the lazy reduction strategy to weak head normal form (if possible).**

### Definition

**Call by value / Leftmost-innermost:**

- The leftmost-innermost reduction strategy tells us to contract in a term that is not in normal form from the leftmost-innermost redex.
- That is the redex that is the leftmost redex of all redexes that do not contain a redex as subterm.
- In a leftmost-innermost reduction sequence, **the arguments of a function are passed to the function body only if they are completely evaluated**.

Example:

- (λx. f x) ((λx. x) 3) →β (λx. f x) 3 →β f 3
- (λx. y) Ω →β (λx. y) Ω →β (λx. y) Ω →β . . .

Advantages:

- redex 不会被重复copy （只有 NF term 会被 copy）
- easy implementation

Disadvantages:

- Not normalizing. 可能原始 term 有 NF，但是通过Leftmost-innermost 方法reduct后，导致了 infinite reduction
- 不是每一步reduction都对最后的结果做了有效贡献。

**Call by need / Leftmost-outermost:**

- The leftmost-outermost reduction strategy prescribes to contract in a term that is not in normal form the lefmost-outermost redex.
- That is, from the outermost redexes (that are not contained by another redex) the leftmost-one.
- In a leftmost-outermost reduction sequence, **the arguments are passed to the function body without evaluating them to normal form**.
- If a term M has a normal form P, then the leftmost-outermost reduction sequences starting in M ends after **finitely many steps** in P.
- If a term M has no normal form, then the leftmost-outermost reduction sequences starting in M is infinite.

Example:

- (λx. f x) ((λx. x) 3) →β f ((λx. x) 3) →β f 3
- (λx. y) Ω →β y

Advantages:

- 保证结果Normalizing。只要原始 term 有 NF，则一定会被找到 （根据confluence，一个 λ-term 有且只有一种 reduction 到 NF 的结果【unique】，即 一个λ-term 如果有 NF，那么不管通过什么路径找到 NF，这个 NF 都是一样的）
- 每一步的结果都对最终结果有效做功。（ leftmost-outermost reductions steps contribute to the normal form）

Disadvantages:

- Redexes may be copied ( does not necessarily yield a shortest reduction to normal form)
- relatively difficult to implement

**Weak head normal form:**

- A weak head normal form, abbreviation WHNF, is a λ-term of one of the following two shapes:
  - λx. P with P [some] λ-term
  - x P1 . . . Pn with n ≥ 0 and P1, . . . , Pn [arbitrary] λ-terms
  - 注意在 normal form 的定义中，只需将 [ ] 内容替换成 [normal]
- Every normal form is a WHNF, but not every WHNF is a normal form.
- A λ-term that is not in WHNF, is of the shape
  - (λx. N) P1 . . . Pn
  - 即，顶端是 @，且左下是 λ

**Lazy reduction:**

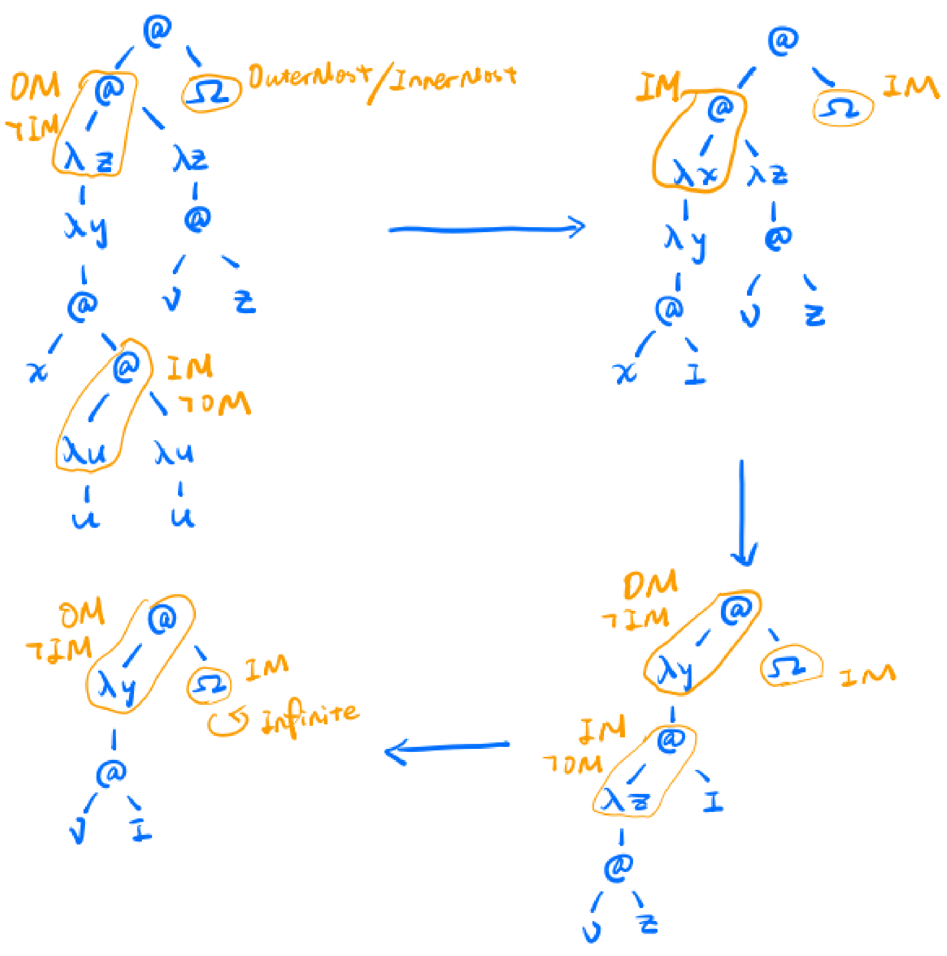
- Lazy evaluation is a **refinement of the leftmost-outermost** reduction strategy.
- Reduce terms if possible to weak head normal form. Moreover, copying of redexes is avoided by using sharing.

**Lazy redex:**

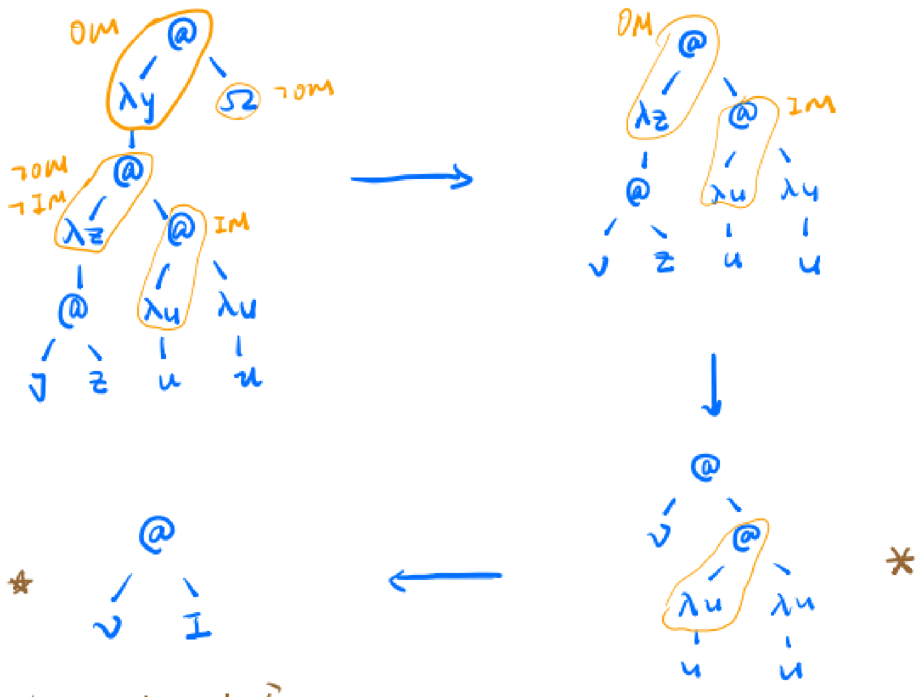
- The redex (λx. N) P1  in the term (λx. N) P1 . . . Pn is sometimes said to be the lazy redex.

## Solution:

- Leftmost-innermost:



- Leftmost-outermost:



★ is a Normal form.

* for lazy reduction to WHNF, stop here!

## Q8

**Reduce in the term**

$(\lambda x.\ f\ (x\ \Omega\ I))\ (\lambda u.\ \lambda v.\ v\ w)$

**repeatedly the lazy redex until a WHNF is reached.**

**Solution:**

$(\lambda x.\ f\ (x\ \Omega\ I))\ (\lambda u.\ \lambda v.\ v\ w) \to f\ ((\lambda u.\ \lambda v.\ v\ w)\ \Omega\ I)$

We stop here, because the variable f applied to the term

$((\lambda u.\ \lambda v.\ v\ w)\ \Omega\ I)$

is a weak head normal form.

---

## Q9

**Reduce the following term in a minimum number of steps to normal form:**

$(\lambda x.(\lambda y.\ z\ y\ y)\ (I\ I))\ (I\ I).$

**Solution:**

Reading from left to right, the first redex

$I\ I$ will be substituted for y

so it will be copied twice, and the second redex

$I\ I$ will be substituted for x

so it will be erased (or copied zero times).

Therefore, we minimize the number of β-reduction steps

if we reduce the first

$I\ I$ to $I$

before it is substituted for y

and we do not reduce the second

$I\ I$

but just let it be erased by substituting it for x.

$$\begin{aligned}(\lambda x.(\lambda y.\ z\ y\ y)\ (I\ I))\ (I\ I) &\to_\beta (\lambda y.\ z\ y\ y)\ (I\ I)\\ &\to_\beta (\lambda y.\ z\ y\ y) I\\ &\to_\beta z\ I\ I\end{aligned}$$

---

## Q10

**Consider the function** foldr:

foldr f b [] = b

foldr f b (h:t) = f h (foldr f b t)

**Use it to give a definition of the function myconcat that takes as input a list of lists,**

**and that gives back as output the concatenation of those lists.**

**Example:**

```
*Main> myconcat [[1,2,3] , [4,5,6]]
[1,2,3,4,5,6]
```

**Give the first four steps of the evaluation of**

myconcat [[1,2,3] , [4,5,6]]

**using informal equational reasoning.**

**Solution:**

```
We define:
   myconcat = foldr append []
   append [] k = k
   append (h : t) k = h : (append t k)

We now evaluate
   myconcat[[1, 2, 3], [4, 5, 6]]:

myconcat[[1, 2, 3], [4, 5, 6]] = foldr append [] [[1, 2, 3], [4, 5, 6]]
               = append [1, 2, 3] (foldr append [] [[4, 5, 6]])
               = 1 : append [2, 3],(foldr append [] [[4, 5, 6]])
               = 1 : 2 : append [3],(foldr append [] [[4, 5, 6]])
               = 1 : 2 : 3 : append [],(foldr append [] [[4, 5, 6]])
               = 1 : 2 : 3 : (foldr append [] [[4, 5, 6]])
               = 1 : 2 : 3 : (append [4, 5, 6] (foldr append [] []))
               = 1 : 2 : 3 : 4 : (append [5, 6, ] (foldr append [] []))
               = 1 : 2 : 3 : 4 : 5 : (append [6],(foldr append [] []))
               = 1 : 2 : 3 : 4 : 5 : 6 : (append [](foldr append [] []))
               = 1 : 2 : 3 : 4 : 5 : 6 : (foldr append [] [])
               = 1 : 2 : 3 : 4 : 5 : 6 : []
               = [1, 2, 3, 4, 5, 6]
```

- 考试时不会出需要 evaluate 这么多步骤的题目，但基础思想不变。

## Q11

**Show the following:** (ite : if - then - else)

ite true P Q $\to_\beta$ P

## Definition

**Booleans:**

- representations of true and false should be two different closed normal forms
- true = λxy. x  (带入两个argument，返回第一个，扔掉第二个)
- false = λxy. y（带入两个argument，返回第二个，扔掉第一个）
- conditions (if–then–else) operator ite:
  - ite = λbxy. b x y
  - ite true P Q $\to_\beta$ P  （如果带入argument为真，则返回 P）
  - ite false P Q $\to_\beta$ Q  （如果带入argument为假，则返回 Q）
- not = λx. x false true
- and = λxy. x y false
- or = λxy. x true y

## Solution

Note that P and Q are arbitrary λ-terms.

ite true P Q = (unfold the definition)
(λx. λy. λz. x y z) true P Q →β
(λy. λz.true y z) P Q →β
(λz.true P z) Q →β
true P Q = (unfold the definition)
(λx. λy. x) P Q →β
(λy. P) Q →β
P

- Note that P and Q are not necessarily closed, but by the variable convention free variables in P or in Q are not captured by substituting P or Q below a λ.

## Q12

**Show the following:**

or true false $\rightarrow_\beta$ true.

### Solution

| | | |
|---|---|---|
| or true false | = | (unfold the definition) |
| (λx. λy. x true y)true false | →β | |
| (λy.true true y)false | →β | |
| true true false | = | (unfold the definition) |
| (λx. λy. x)true false | →β | |
| (λy.true)false | →β | |
| true | | |

---

## Q13

**Show that**

Suc $c_1 \rightarrow_\beta c_2$,

**with Suc defined as** λxsz. s (x s z).

### Definition

**Church numerals:**

- $c_0 = \lambda s.\, \lambda z.\, z$
- $c_1 = \lambda s.\, \lambda z.\, sz$
- $c_2 = \lambda s.\, \lambda z.\, s(sz)$
- $c_n = \lambda s.\, \lambda z.\, s^n(z)$.

**Successor (Suc):**

- the representation of the successor function adding one to a natural number
- $Suc = \lambda xsz.\, s(xsz)$
- $Suc\; c_2 = c_3$

### Solution:

```
Suc c1                    =   (unfold the definition)
(λxsz. s (x s z)) c1            →β
λs. λz. s (c1 s z)            =      (unfold the definition)
λs. λz. s ((λu. λv. u v) s z)       →β
λs. λz. s ((λv. s v) z)           →β
λs. λz. s (s z)             =    (fold the definition)
c2
```

---

## Q14

**Show that**

$$Plus\ c_1\ c_2\ \rightarrow_\beta\ c_3$$

**with Plus defined as** λmn. λsz. m s (n s z).

### Solution:

```
Plus c1 c2                       = (unfold the definition)
(λmn. λsz. m s (n s z)) c1 c2          →β
(λn. λsz. c1 s (n s z)) c2              →β
λsz. c1 s (c2 s z)                    = (unfold the definition)
λsz. c1 s ((λu λv. u (u v)) s z)        →β
λsz. c1 s ((λv. s (s v)) z)              →β
λsz. c1 s (s (s z))                 = (unfold the definition)
λsz.(λu λv. u v) s (s (s z))              →β
λsz.(λv. s v) (s (s z))                 →β
λsz. s (s (s z))                     = (fold the definition)
c3
```

---

## Q15

**Show that for arbitrary P and Q we have the following:**

$$\pi_1\ (\pi\ P\ Q)\ \rightarrow_\beta\ P.$$

### Definition

**Pairs:**

- an operator for pairing can be defined in the λ-calculus. Its definition is as follows:
    - π = λlrz. z l r
    - π P Q = (λlrz. z l r) P Q $\rightarrow_\beta$ λz. z P Q

**Projection:**

- π1 = λu. u (λlr. l) (相当于 (λu. u) true)
- π2 = λu. u (λlr. r)
- π1 (π P Q) $\to_\beta$ P (作用是从一个pair中取的第一个元素)
- π2 (π P Q) $\to_\beta$ Q (作用是从一个pair中取的第二个元素)

**Solution**

```
π1 (π P Q)
= (λu.u (λlr. l)) (π P Q)
-> (π P Q) true
= ((λlrz. z l r) P Q) true        ((λlrz. z l r) P Q 处理这种reduction时，就是 按 λ
绑定顺序代入，比如这里 P 对应 l， Q 对应 r， 把对应位置的代入了，没有
argument的就保留，如 λz)
-> (λz.z P Q) true
-> true P Q
= (λxy. x) P Q
-> P
```

---

## Q16

**We represent lists using**

nil := λxy. y and cons := λht. λz. z h t = π

**Give a λ-term for empty (the function that takes as input a list and yields true if the list is empty and false otherwise), and show that empty nil reduces to true.**

**Definition：**

**Lists:**

- Pairs can be used to define lists.
- A list is seen as either the empty list, often denoted by nil, or an element (the head) followed by a list (the tail).
- The operation that adds in the front an element to a list is often called cons.
- nil = λxy. y
- cons = λht. λz. z h t
- head = λl. l(λht. h)
- tail = λl. l(λht. t)
- head (cons H T) ->β H

- tail (cons H T) ->β T

**Solution:**

```
empty = λl . l (λx. λy. λz. false) true

# 推导过程:
We want :
    empty nil = true
    empty (cons H T) = false
so:
    empty nil = (λl . l ① ②) nil
    (λl. l 代表要代入的list， 这里就是 nil )
we know:
    ② is true
    (因为 nil 代入 λl 之后，empty nil -> nil ① ②， 而 nil = λxy. y，也就是
return第二个，不看第一个，那么 ① 不管是什么，都不影响，但是 ② 必须是
true)
then:
    empty = λl. l ① true

we then have:
    empty (cons H T) = (λl. l ① true) (cons H T) ->
    (cons H T) ① true =
    ((λht. λz.z ht) H T) ① true ->
    (λz. z H T) ① true ->
    (① H T) true
    (推到这里, 只剩下判断 ① 到底要如何表达,
    根据 (① H T) true 得出 ① 需要代入 3 个 argument （H, T, true），并且返
回结果 false。
    明显这里我们不管 H T 是什么状况，最后都是return false,
    因此需要一个term 代入三个 argument，并且无视这三个argument，返回
false)

so:
    ① = λa.λb.λc. false

Therefore:
    empty = λl. l (λa.λb.λc. false) true

Verify empty nil = true:

empty nil =
```

(λl. l (λa.λb.λc. false) true) (λxy. y) ->
λxy. y ((λa.λb.λc. false) true) ->
true

---

## Q17

**We define**

τ = λx1. λx2. λx3. λz. z x1 x2 x3.

**This λ-term can be used to build triples:**

$\tau\, P_1\, P_2\, P_3 \ \to_\beta\ \lambda z.\ zP_1\, P_2\, P_3.$

**Now define λ-terms τ1, τ2 and τ3 representing the first, second, and third projection.**

**Show that they have the right behaviour, that is:**

$\tau_1\ (\tau\, P_1\, P_2\, P_3)\ \to_\beta\ P_1,$

$\tau_2\ (\tau\, P_1\, P_2\, P_3)\ \to_\beta\ P_2,$

$\tau_3\ (\tau\, P_1\, P_2\, P_3)\ \to_\beta\ P_3.$

## Solution:

The idea is to adapt the first and second projection from the pairing operator to this case of triples.
We define the following:
τ1 = λy. y (λx1. λx2. λx3. x1)
τ2 = λy. y (λx1. λx2. λx3. x2)
τ3 = λy. y (λx1. λx2. λx3. x3)

We now show that these terms have the right behaviour.

Take
arbitrary terms P1, P2, P3.

We have the following:
τ1 (τ P1 P2 P3) = τ1 ((λx1. λx2. λx3. λz. z x1 x2 x3) P1 P2 P3)
→∗β τ1 (λz. z P1 P2 P3)
= (λy. y (λx1. λx2. λx3. x1)) (λz. z P1 P2 P3)
→β (λz. z P1 P2 P3) λx1. λx2. λx3. x1
→β (λx1. λx2. λx3. x1) P1 P2 P3
→∗β P1

## Q18

**a. Give a specification for the operation exclusive or, notation xor, that takes two booleans as input and yields true if exactly one of the inputs is true.**

### Solution:

The (to be found) term xor should satisfy the following:

  xor true true =β false
  xor true false =β true
  xor false true =β true
  xor false false =β false

**b. Give a λ-term for xor.**

### Solution:

We define

  xor = λx. λy. x (not y) y, with not = λx. x false true.

A bit of intuition:

  xor takes two inputs,
more intuitively: two booleans as input (something which cannot be enforced in the untyped λ-calculus).
Hence the start with two abstractions.

If the first input is or evaluates to true,
then from the first two lines of the specification we see that we should return the negation of the second input.

Hence the first argument of x, the one that is returned if x is true, is not y.

If the first input is or evaluates to false, then from the last two lines of the specification we see that we should return the second input.
Hence the second argument of x, the one that is returned if x is false, is y.

More straight forward:

xor = λx λy. ?
(two arguments)

```
true = λx. λy . x
false = λx. λy. y

xor = λx. λy. x ① ②
if ① true,
then, return not ②,
because:
   x true false = true
   x true true = false

therefore,
   xor = λx. λy. x (not y) ②
if ① false,
then, return ②,
because:
   x false true = true
   x false false = false

 therefore,
   xor = λx. λy. x (not y) y
```

**c. Show for one of the four possible inputs that xor satisfies the specification.**

**Solution:**

```
xor false true
= (λx. λy. x (not y) y)false true
→*β false (not true)true
= (λx. λy. y) (not true)true
→*β true

More detailed version:

xor true false
= ( λx. λy. x (not y) y) true false
->-> true (not false) false
= (λx. λy.x) (not false) false
->-> not false
= (λx.x false true) false
-> false false true
= (λx. λy. y) false true
->-> true
```