COMP 3400 Assignmet 1 written

Hyeyeon Lee

True = $\lambda x.(\lambda y.x)$ False = $\lambda x.(\lambda y.y)$

And = $\lambda p.(\lambda q.(pq)p)$ Or = $\lambda p.(\lambda q.(pp)q)$

Question 1. [1 MARK]

Give the λ-expression for NOT that takes True to False and vice-versa. Your solution should be in its β-normal form.

Q1 $\lambda x.xFT$

$\lambda x.(x(\lambda p.\lambda q.q)(\lambda a.\lambda b.a))$

→ $\lambda x.(x(\lambda pq.q)(\lambda ab.a))$

Due to, $F=\lambda x.(\lambda y.y)$ F a b will always return b (second element in a b).

And due to $T=\lambda x.(\lambda y.x)$, T a b will always return a (first element in a b).

If the input is False the return has to be True and vice-versa,

T T T = T	F T T = T → 2
T T F = T	F T F = F
T F T = F → 1	F F T = T → 2
T F F = F → 1	F F F = F

The first element is the key of the result.

Therefore, for the return value to take the opposite value, the input needs to be in front of the others.

According to the expressions above when the first element is T the return value need to be F which is true for the statements marked as 1 and when the first element is F the return value needs to be T which is true for the statements marked as 2.

And the only case that fits these condition is when the expression is _FT

Therefore, expression of Not is $\lambda x.xFT = \lambda x.(x(\lambda pq.q)(\lambda ab.a))$

Question 2. [5 MARKS]

Recall that $\neg(p \wedge q) \equiv \neg p \vee \neg q$ and thereby Or is redundant because

$$p \vee q \equiv \neg(\neg p \wedge \neg q).$$

Give the λ-expression for $\neg(\neg p \wedge \neg q)$ and show it is equivalent to Or.

Q2 To Prove 'Porq ≡ ~(~P and ~q)' Prove the truth table of Porq and ~(~P and ~q) is equivalent.

(Accorinding to Q1 '~' is $\lambda x. x F T$)

T OR F	$(\lambda x.\lambda y.xxy)TF$	F OR T	$(\lambda x.\lambda y.xxy)FT$
=	$(\lambda y.xxy)[x:=T]F$	=	$(\lambda y.xxy)[x:=F]T$
=	$(\lambda y.TTy)F$	=	$(\lambda y.FFy)T$
=	(TTF)	=	FFT
=	$(\lambda x.(\lambda y.x))TF$	=	$(\lambda x.(\lambda y.y))FT$
=	$(\lambda y.T)F = T$	=	$(\lambda y.y)T = T$

T OR T	$(\lambda x.\lambda y.xxy)TT$	F OR F	$(\lambda x.\lambda y.xxy)FF$
=	$(\lambda y.xxy)[x:=T]T$	=	$(\lambda y.xxy)[x:=F]F$
=	$(\lambda y.TTy)T$	=	$(\lambda y.FFy)F$
=	TTT	=	FFF
=	$(\lambda x.(\lambda y.x))TT$	=	$(\lambda x.(\lambda y.y))FF$
=	$(\lambda y.T)T = T$	=	$(\lambda y.y)F = F$

Therefore, truth table of 'p or q' is

p	q	p OR q
T	T	T
T	F	T
F	T	T
F	F	F

Expressions below are the ones used to make truth table of $\sim(\sim p \text{ and } \sim q)$

$FFT = \lambda x.(\lambda y.y)FT = T$ ⋯ ①

$TFT = \lambda x\ (\lambda y.x)FT = F$ ⋯ ②

$TFF = \lambda x.(\lambda y.x)FF = F$ ⋯ ③

$FTF = \lambda x.(\lambda y.y)TF = F$ ⋯ ④

$FFF = \lambda x.(\lambda y.y)FF = F$ ⋯ ⑤

$TTT = \lambda x.(\lambda y.x)TT = T$ ⋯ ⑥

$\sim(\sim T \text{ and } \sim T)$	$(\lambda x.xFT)((\lambda p.(\lambda q(pq)p))((\lambda a.aFT)T)((\lambda b.bFT)T))$
$=$	$(\lambda p.(\lambda q(pq)p))((\lambda a.aFT)T)((\lambda b.bFT)T)FT$
$=$	$(\lambda q.(pq)p)[p := ((\lambda a.aFT)T)]((\lambda b.bFT)T)FT$
$=$	$(\lambda q.(((\lambda a.aFT)T)q)((\lambda a.aFT)T))((\lambda b.bFT)T)FT$
$=$	$(((\lambda a.aFT)T)((\lambda b.bFT)T))((\lambda a.aFT)T)FT$
$=$	$((TFT)(TFT))(TFT)FT$ ⋯ ②
$=$	$(FF)FFT$ ⋯ ⑤
$=$	FFT ⋯ ①
$=$	T

~(~T and ~F)	$(\lambda p.(\lambda q(pq)p))((\lambda a.aFT)T)((\lambda b.bFT)F)FT$
=	$(\lambda q.(pq)p)[p := ((\lambda a.aFT)T)]((\lambda b.bFT)F)FT$
=	$(\lambda q.(((\lambda a.aFT)T)q)((\lambda a.aFT)T))((\lambda b.bFT)F)FT$
=	$(((\lambda a.aFT)T)((\lambda b.bFT)F))((\lambda a.aFT)T)FT$
=	$((TFT)(FFT))(TFT)FT$
	... ②, ①
=	$(FT)FFT$
	... ④
=	FFT
	... ①
=	T

~(~F and ~T)	$(\lambda p.(\lambda q(pq)p))((\lambda a.aFT)F)((\lambda b.bFT)T)FT$
=	$(\lambda q.(pq)p)[p := ((\lambda a.aFT)F)]((\lambda b.bFT)T)FT$
=	$(\lambda q.(((\lambda a.aFT)F)q)((\lambda a.aFT)F))((\lambda b.bFT)T)FT$
=	$(((\lambda a.aFT)F)((\lambda b.bFT)T))((\lambda a.aFT)F)FT$
=	$((FFT)(TFT))(FFT)FT$
	... ①, ②
=	$(TF)TFT$
	... ②
=	FFT
	... ①
=	T

$\sim(\sim F \text{ and } \sim F)$

$(\lambda p.(\lambda q (pq)p))((\lambda a.aFT)F)((\lambda b.bFT)F)FT$

$= (\lambda q.(pq)p)[p := ((\lambda a.aFT)F)]((\lambda b.bFT)F)FT$

$= (\lambda q.(((\lambda a.aFT)F)q)((\lambda a.aFT)F))((\lambda b.bFT)F)FT$

$= (((\lambda a.aFT)F)((\lambda b.bFT)F))((\lambda a.aFT)F)FT$

$= ((FFT)(FFT))(FFT)FT$ ... 1

$= (TT)TFT$ ... 6

$= TFT$ ... 2

$= F$

According to Proof above truth table of '$\neg(\neg p \wedge \neg q)$' is

$\sim(\sim p \text{ and } \sim q)$	p	q
T	T	T
T	T	F
T	F	T
F	F	F

Since the truth table of '$p \vee q$' and '$\neg(\neg p \wedge \neg q)$' is equivalent, expression '$p \vee q$' and '$\neg(\neg p \wedge \neg q)$' is equivalent.

Question 3. [4 MARKS]

Reduce the following lambda expression to its β-normal form.

$$(\lambda xy.x)(\lambda abc.cab)z(\lambda z.zz).$$

$(\lambda xy.x)(\lambda abc.cab)z(\lambda z.zz)$ } α conversion, β reduction

$= (\lambda y.x)[x := (\lambda abc.cab)]z(\lambda p.pp)$

$= (\lambda y.(\lambda abc.cab))z(\lambda p.pp)$ } β reduction

$= (\lambda abc.cab)[y := z](\lambda p.pp)$

$= (\lambda abc.cab)(\lambda p.pp)$ } β redution

$= (\lambda bc.cab)[a := (\lambda p.pp)]$

$= (\lambda bc.c(\lambda p.pp)b)$

Question 4. [2 MARKS]
Define a function `f1` such that

```
> :type f1
f1 :: (a -> b, a) -> b
```

up to renaming of the type variables. Your function should be total and *not* be undefined.

((a→b), a) → b
f

→ f1 (f , a) = f a

Question 5. [2 MARKS]
Same instructions as Question 4 but with

```
f2 :: a -> (b, c) -> b
```

→ f2 a (b,c) = b

Question 6. [2 MARKS]
Same instructions as Question 4 but with

```
f3 :: (a -> a) -> a -> [a]
```

f

→ f3 f a = [f a]

// (t → a) → t → a

Question 7. [2 MARKS]
Same instructions as Question 4 but with

```
f4 :: (b -> r) -> (a -> b) -> (a -> r)
```

f1 f2 return

→ f4 f1 f2 a = f1 (f2 a)

Question 8. [1 MARK]

Same instructions as Question 4 but with

f1

```
f5 :: ((a, b, c) -> d) -> a -> b -> c -> d
```

tuple

→ f5 f a b c = f (a, b, c)

Question 9. [1 MARK]

Same instructions as Question 4 but with

```
f5_inv ::  (a -> b -> c -> d) -> (a, b, c) -> d
```

f tuple

→ f (a, b, c) = f a b c

Question 10. [10 MARKS]

Most of the programs we write in Haskell will be *recursive* or *inductive* in nature. The purpose of this question is to help us get into the mindset of reasoning inductively.

Use the *principle of mathematical induction* to prove a $2^n \times 2^n$ Blockus board with north-west corner removed can be covered with V3 pieces.

Note: This question will be marked very thoroughly. We will be looking for the presence of all necessary components of induction to be *stated clearly*. You will be marked down for being unnecessarily verbose or for making unsubstantiated claims. Every statement you write should be clearly inferred from the statements that precede it (not statements that come after).

Essentially we are looking for *clear* and *concise* proofs.

State P

Statement to Prove is $2^n \times 2^n$ sized blockus board with north-west corner removed can be covered with V3 pieces

Base case

Base case when n is 0, the size of the board will be $2^0 \times 2^0 = 1$

when the corner of the Board is removed there is nothing to do more.

Therefore, it is true.

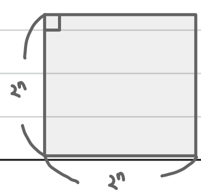
Inductive Hypothesis

To prove the statement is true for all natural numbers n, first assume the statement is True for n and prove it is also true for n+1. This will prove $2^n \times 2^n$ with removed north-west corner can be covered with V3 blocks.

Induction

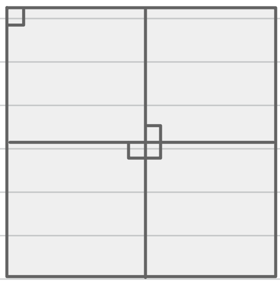
Assume $2^n \times 2^n$ size blockus board with one coner removed can be filled with V3 block.

that size of 1 board with corner removed can be covered with V3 blocks



$2^{n+1} \times 2^{n+1}$ sized blokus board can be said the number of blocks in the board is 4 times that of $2^n \times 2^n$ sized board. Due to, $2^{n+1} \times 2^{n+1} = 2 \times 2^n \times 2 \times 2^n = 4 \times 2^n \times 2^n$.

Therefore, $2^{n+1} \times 2^{n+1}$ sized board is identical to having 4 $2^n \times 2^n$ sized board attatched like the drawing below.



By the hypothesis, $2^n \times 2^n$ size board with north-west corner removed can be covered with v3, like the drawing below leave one removed corner in the north-west corner and place the $2^n \times 2^n$ blocks to have the removed corners together. As a result, they will be v3 size empty space and after covering with v3 the whole $2^{n+1} \times 2^{n+1}$ board will be covered except the north-west corner.

Therefore, $2^{n\times 1} \times 2^{n\times 1}$ sized board can be covered with v3 blocks when 1×1 size block in the north-west corner is removed.

By the Principal of mathmetical Induction, for every natural number it is true that $2^n \times 2^n$ size board with removed north-west corner can be covered by only using v3 blocks.