Foundations1 assignment 2018
Submit by Monday of week 9 (5 November 2018)
worth 15%
Sumit to the coursework submission box before 16:00 hours
Submit only typed material
No handwritten material

Throughout the assignment, assume the terms and definitions given in the DATA SHEET.

1. Just like the translation function $\omega : \mathcal{M} \mapsto \Lambda$ is defined in the DATA SHEET, give translation functions $\omega_1 : \Lambda \mapsto \mathcal{M}$ and $\omega_2 : \mathcal{M} \mapsto \Lambda'$ and $\omega_3 : \Lambda' \mapsto \mathcal{M}'$ and $V_1 : \mathcal{M}' \mapsto \mathcal{M}$. Your translation functions need to be complete with all subfunctions and needed information (just like ω was complete with all needed information). Submit all these functions here. (2)

   MOST DETAILS ARE BELOW: ADD WHAT IS MISSING.....

   - Assume an ordered list of variables $listorder = [x, y, z, x', y', z', x_1, y_1, z_1, x_2, y_2, z_2, \dots]$.
     Let $listelts(n, l)$ be the list that consists of the first n elements of l.
     Let $elt(n, l)$ be the nth element of l. For example, $elt(3, listorder) = z$.
     We define the free variables of a term in Λ as usual. For eample, in $\lambda 13(\lambda(\lambda 4)1)$, the free variables are: 3-1 = 2 (which corresponds to y) and 4-3= 1 (which corresponds to x).
     We define the largest free variable in a term A of Λ to be the largest free index in A. For example, in $\lambda 1\ 3(\lambda(\lambda 4)1)$, the largest free variable is the variable $3 - 1 = 2$ (which corresponds to y'0 and not $4 - 3 = 1$ (which corresponds to x). If A has no free variables then the largest free variable would be 0.
     We define $\omega_1 : \Lambda \mapsto \mathcal{M}$ as follows:
     $\omega_1(A) = \omega_1'(n + 1, listelts(n, listorder), A)$ where n is the largest free variable in A

$\omega_1'(n, l, m) = elt(m, l)$
$\omega_1'(n, l, \lambda A) = \lambda x.\omega_1'(n+1, x :: l, A)$ where $x = elt(n, listorder)$.
$\omega_1'(n, l, AB) = \omega_1'(n, l, A)\omega_1'(n + \text{number of } \lambda\text{s in } A, l, B)$

For example, the largest free variable in $\lambda 1\ 2$ is $2-1$ which corresponds to x. Hence, $\omega'(\lambda 1\ 2) = \omega_1'(2, listelts(1, listorder), \lambda 1\ 2) = \omega_1'(2, [x], \lambda 1\ 2) = \lambda y.\omega_1'(3, [y, x], 1\ 2) = \lambda y.\omega_1'(3, [y, x], 1)\omega_1'(3 + \text{number of } \lambda\text{s in } 1, [y, x], 2) = \lambda y.elt(1, [y, x])elt(2, [y, x]) = \lambda y.yx$.

For example: $\omega_1(\lambda 1(\lambda 21)3) = \lambda z.z(\lambda x'.zx')y$.

The largest free variable is $3 - 1 = 2$ and it corresponds to y.

$\omega_1(\lambda 1(\lambda 2\ 1)3) = \omega_1'(3, [x, y], \lambda 1(\lambda 2\ 1)3) = \lambda z.\omega_1'(4, [z, x, y], 1(\lambda 2\ 1)3) =$
$\lambda z.\omega_1'(4, [z, x, y], 1(\lambda 2\ 1))\omega_1'(5, [z, x, y], 3) =$
$\lambda z.\omega_1'(4, [z, x, y], 1)\omega_1'(4, [z, x, y], \lambda 21)\omega_1'(5, [z, x, y], 3) = \lambda z.z\omega_1'(4, [z, x, y], \lambda 21)\omega_1'(5, [z, x, y], 3) =$
$\lambda z.z\omega_1'(4, [z, x, y], \lambda 2\ 1)y = \lambda z.z(\lambda x'.\omega_1'(5, [x', z, x, y], 2\ 1))y =$
$\lambda z.z(\lambda x'.\omega_1'(5, [x', z, x, y], 2)\omega_1'(5, [x', z, x, y], 1))y = \lambda z.z(\lambda x'.z\omega_1'(5, [x', z, x, y], 1))y =$
$\lambda z.z(\lambda x'.zx')y$.

- and $\omega_2 : \mathcal{M} \mapsto \Lambda'$
  .....
- $\omega_3 : \Lambda' \mapsto \mathcal{M}'$
  .....
- $V_1 : \mathcal{M}' \mapsto \mathcal{M}$:
  ....

2. For each of the SML terms vx, vy, vz, t1, $\cdots$ t9 in `http://www.macs.hw.ac.uk/~fairouz/foundations-2018/slides/data-files.sml`, let the overlined term represent the corresponding term in $\mathcal{M}$. I.e., $\overline{\text{vx}} = x$, $\overline{\text{vy}} = y$, $\overline{\text{vz}} = z$, $\overline{\text{t1}} = \lambda x.x$, $\overline{\text{t2}} = \lambda y.x$, $\cdots$.
   For each of vx, vy, vz, t1, t2, $\cdots$ t9, give $\overline{\text{vx}}$, $\overline{\text{vy}}$, $\overline{\text{vz}}$, $\overline{\text{t1}}$, $\overline{\text{t2}}$, $\cdots \overline{\text{t9}}$ in $\mathcal{M}$, and the translation into the corresponding terms of Λ (using $\omega : \mathcal{M} \mapsto \Lambda$), of Λ' (using $\omega_2 : \mathcal{M} \mapsto \Lambda'$), of $\mathcal{M}'$ (using $\omega_3 : \Lambda' \mapsto \mathcal{M}'$), and into $\mathcal{M}$ (using $\omega_1 : \Lambda \mapsto \mathcal{M}$ and not using $V_1 : \mathcal{M}' \mapsto \mathcal{M}$).

   Your output should be tidy as follows:

	$\bar{\cdot}$	ω	ω_2	ω_3	ω_1
t_1	$\lambda x.x$	$\lambda 1$	$[\,]1$	$[x]x$	$\lambda x.x$

(1)

	$\bar{\cdot}$	ω_3
vx	x	x
vy	y	y
vz	z	z
t_1	$\lambda x.x$	$[x]x$
t_2	..	..
t_3	..	..
t_4	..	..
t_5	..	..
t_6	..	..
t_7	..	..
t_8	..	..
t_9	$(\lambda z.z((\lambda x.x)z))((\lambda x.x)(\lambda y.x)z)$	$\langle\langle z\rangle\langle[y]x\rangle[x]x\rangle[z]\langle\langle z\rangle[x]x\rangle z$

	ω_1	ω	ω_2
vx	x	1	1
vy	y	2	2
vz	z	3	3
t_1	$\lambda x.x$	$\lambda 1$	$[\,]1$
t_2	..	..	..
t_3	..	..	..
t_4	..	..	..
t_5	..	..	..
t_6	..	..	..
t_7	..	..	..
t_8	..	..	..
t_9	..	..	..

3. Just like SML terms vx, vy, vz, t1, t2, $\cdots$ t9 are introduced which implement terms in $\mathcal{M}$, please implement the corresponding terms each of the other sets $\mathcal{M}'$, Λ, Λ'. Your output must be like the output in `http://www.macs.hw.ac.uk/~fairouz/foundations-2018/slides/data-files.sml`, for the implementation of these terms of $\mathcal{M}$. I.e., your output for each set must be similar to the following: (1)

   The implementation of terms in $\mathcal{M}$ is as follows:

```
val vx = (ID "x");
val vy = (ID "y");
val vz = (ID "z");
val t1 = (LAM("x",vx));
val t2 = (LAM("y",vx));
val t3 = (APP(APP(t1,t2),vz));
```

```
val t4 = (APP(t1,vz));
val t5 = (APP(t3,t3));
val t6 = (LAM("x",(LAM("y",(LAM("z",
                        (APP(APP(vx,vz),(APP(vy,vz))))))))));
val t7 = (APP(APP(t6,t1),t1));
val t8 = (LAM("z", (APP(vz,(APP(t1,vz))))));
val t9 = (APP(t8,t3));
```

- ```
  val Ivx = (IID "x");
  val Ivy = (IID "y");
  val Ivz = ..
  val It1 = ..
  val It2 = ..
  val It3 = ..
  val It4 = ..
  val It5 = (IAPP(It3,It3));
  val It6 = ..
  val It7 = ..
  val It8 = ..
  val It9 = ..
  ```
- ```
  val Bvx = (BID 1);
  val Bvy = (BID 2);
  val Bvz = (BID 3);
  val Bt1 = (BLAM(Bvx));
  val Bt2 = ..
  val Bt3 =..
  val Bt4 = ..
  val Bt5 = ..
  val Bt6 = ..
  val Bt7 = ..
  val Bt8 = ..
  val Bt9 = ..
  ```
- ```
  val IBvx = (IBID 1);
  val IBvy = (IBID 2);
  val IBvz = (IBID 3);
  val IBt1 = ..
  val IBt2 = ..
  val IBt3 =..
  val IBt4 =..
  val IBt5 =..
  ```
  ```

```
val IBt6 = ..
val IBt7 = ..
val IBt8 = ..
val IBt9 = ..
```

4. For each of $\mathcal{M}'$, Λ, Λ', implement a printing function that prints its elements nicely and you need to test it on every one of the corresponding terms vx, vy, vz, t1, t2, $\cdots$ t9. Your output for each such set must be similar to the one below (1)

```
(*Prints a term in classical lambda calculus*)
fun printLEXP (ID v) =
    print v
  | printLEXP (LAM (v,e)) =
    (print "(\\";
     print v;
     print ".";
     printLEXP e;
     print ")")
  | printLEXP (APP(e1,e2)) =
    (print "(";
     printLEXP e1;
     print " ";
     printLEXP e2;
     print ")");
```

Printing these $\mathcal{M}$ terms yields:

```
-printLEXP vx;
xval it = () : unit

-printLEXP vy;
yval it = () : unit

-printLEXP vz;
zval it = () : unit

-printLEXP t1;
(\x.x)val it = () : unit

-printLEXP t2;
(\y.x)val it = () : unit
```

```
-printLEXP t3;
(((\x.x) (\y.x)) z)val it = () : unit

-printLEXP t4;
((\x.x) z)val it = () : unit

-printLEXP t5;
((((\x.x) (\y.x)) z) (((\x.x) (\y.x)) z))val it = () : unit

-printLEXP t6;
(\x.(\y.(\z.((x z) (y z)))))val it = () : unit

-printLEXP t8;
(\z.(z ((\x.x) z)))val it = () : unit

-printLEXP t9;
((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z))val it = () : unit
```

- `(*Prints a term in item lambda calculus*)`
  `fun printIEXP (IID v) =.....`
  Printing these $\mathcal{M}$' terms yields:

  ```
  -printIEXP Ivx;
  xval it = () : unit

  -printIEXP Ivy;
  yval it = () : unit

  -printIEXP Ivz;
  zval it = () : unit

  .....REPEAT FOR ALL TERMS.....

  -printIEXP It9;
  <<z><[y]x>[x]x>[z]<<z>[x]x>zval it = () : unit
  ```

- `(*Prints a term in classical lambda calculus with de Bruijn indices*)`
  ```
  fun printBEXP (BID n) =
      print  (Int.toString n)
  ```

```
  | printBEXP (BLAM (e)) =......
```

Printing these Λ terms yields:

```
-printBEXP Bvx;
1val it = () : unit

-printBEXP Bvy;
2val it = () : unit

-printBEXP Bvz;
3val it = () : unit

-printBEXP Bt1;
(\1)val it = () : unit

.....REPEAT FOR ALL TERMS.....
```

- ```
  (*Prints a term in item lambda calculus with de Bruijn indices*)
  fun printIBEXP (IBID n) =....
  ```

  Printing these Λ' terms yield:

  ```
  - printIBEXP IBvx;
  1val it = () : unit
  - printIBEXP IBvy;
  2val it = () : unit
  - printIBEXP IBvz;
  3val it = () : unit
  - printIBEXP IBt1;
  []1val it = () : unit

  .....REPEAT FOR ALL TERMS.....
  ```

5. Implement in SML the translation functions ω, ω_1, ω_2, ω_3 and V_1 and give these implemented functions here. (2)

   - ω is implemented as follows:
   - ω_1 is implemented as follows:
   - ω_2 is implemented as follows:
  ```

- ω_3 is implemented as follows:
- $V_1 : \mathcal{M}' \mapsto \mathcal{M}$ is implemented as follows:

```
fun tran  (IID id) =   (ID id) |
    tran  (ILAM(id,e)) = LAM(id,(tran e))|
    tran  (IAPP(e1,e2))= APP((tran e2),(tran e1));
```

6. Test these functions on the translations of all the given terms vx, vy, vz, t1, $\cdots$ t9 and give your output clearly.

   For example, if we implement itran (a translation function from $\mathcal{M}$ to $\mathcal{M}'$), then printIEXP prints expressions in $\mathcal{M}'$. Hence,

```
- printIEXP (itran t5);
<<z><[y]x>[x]x><z><[y]x>[x]xval it = () : unit
```

   You need to show how all your terms are translated in all these sets and how you print them. (2)

   Straightforward, something like this:

   - ```
     - printIEXP (Itran vx);
     xval it = () : unit
     ........
     - printIEXP (Itran t7);
     <[x]x><[x]x>[x][y][z]<<z>y><z>xval it = () : unit
     ......
     ```

7. Translate $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ in each of $\mathcal{M}'$, Λ and Λ' and give the SML implementation of all these translations. (1)

   - In LEXP:

     ```
     val w = (APP(LAM("x",APP(vx,vx)),......
     ```

   - In $\mathcal{M}'$: $< [x] < x > x > [x] < x > x$. In IEXP:

     ```
     val Iw = (IAPP(ILAM("x",IAPP(Ivx,Ivx)),....
     ```

   - In Λ: $(\lambda 1\ 1)(\lambda 1\ 1)$. In BEXP:

     ```
     val Bw =   (BAPP(BLAM(BAPP(Bvx,Bvx)),......
     ```

   - In Λ': .....

     ```
     val IBw =  .....
     ```
     ```

8. Give an implementation of leftmost reduction with a counter in $\mathcal{M}$ and test it on a number of examples. For example:

```
-  countprintloreduce t5;
((((\x.x) (\y.x)) z) (((\x.x) (\y.x)) z))-->
(((\y.x) z) (((\x.x) (\y.x)) z))-->
(x (((\x.x) (\y.x)) z))-->
(x ((\y.x) z))-->
(x x)
Number of Steps: 4
val it = () : unit
```

(2)

- Leftmost is printloreduce in the data-files.sml (and it needs all the necessary functions in that file).

```
(* Below the  printloreduce with a counter *)
fun countprintloreduce e =
.......


- countprintloreduce vx;
x
Number of Steps: 0
val it = () : unit
-  countprintloreduce vy;
y
Number of Steps: 0
val it = () : unit
..............
-  countprintloreduce t7;
(((\x.(\y.(\z.((x z) (y z))))) (\x.x)) (\x.x))-->
((\y.(\z.(((\x.x) z) (y z)))) (\x.x))-->
(\z.(((\x.x) z) ((\x.x) z)))-->
(\z.(z ((\x.x) z)))-->
(\z.(z z))
Number of Steps: 4
val it = () : unit
.........
```

9. Give an implementation of rightmost reduction with a step counter in $\mathcal{M}$ and test it on a number of examples similarly to what you did for the above question. (2)

- As for right most reduction, recall the non working rireduce of the data sheet. This does not work. It repeats steps and also loops on terms that do not have redexes (like t6 which is the S combinator).

  The problem of rireduce comes from the last lines:

```
 if is_var(e1) then [e4] else
(rireduce (APP(one_rireduce e1, e3)))
in l2 @ l3
```

  We should check if e1 has a redex before doing ONE-RIREDUCE.

  We will replace these two lines by the case where e1 has a redex in which case, we do one step rightmost reduction on e1 and recurse. If there is no redex in e1, we return [].

  Also, in order to remove duplication, we ....

```
fun newrireduce (ID id) =  [(ID id)] |
......


(* Below the  printnewrireduce with a counter *)
fun countprintnewrireduce e =
if (has_redex e) then
    (printlisttermsarrow(newrireduce e); print "\n";
     print "Number of Steps:";
     print(Int.toString(List.length (newrireduce e)-1));
     print "\n")
else ((printLEXP e); print "\n"; print "Number of Steps: 0 \n");
```

-
```
-  countprintnewrireduce t1;
(\x.x)
Number of Steps: 0
val it = () : unit
.....
-  countprintnewrireduce t8;
(\z.(z ((\x.x) z)))-->
(\z.(z z))
Number of Steps:1
val it = () : unit
```

10. Test your implementations of leftmost and rightmost on a number of examples to deduce which is more efficient and which is more likely to terminate. (1)

- Efficiency: rightmost is more efficient than leftmost.

```
- countprintloreduce (APP(LAM("x",APP(vx,vx)),t9));
((\x.(x x)) ((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z)))-->
(((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z)) ((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z)))-->
(((((\x.x) (\y.x)) z) ((\x.x) (((\x.x) (\y.x)) z))) ((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z)))-->
((((\y.x) z) ((\x.x) (((\x.x) (\y.x)) z))) ((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z)))-->
((x ((\x.x) (((\x.x) (\y.x)) z))) ((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z)))-->
((x (((\x.x) (\y.x)) z)) ((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z)))-->
((x ((\y.x) z)) ((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z)))-->
((x x) ((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z)))-->
((x x) ((((\x.x) (\y.x)) z) ((\x.x) (((\x.x) (\y.x)) z))))-->
((x x) (((\y.x) z) ((\x.x) (((\x.x) (\y.x)) z))))-->
((x x) (x ((\x.x) (((\x.x) (\y.x)) z))))-->
((x x) (x (((\x.x) (\y.x)) z)))-->
((x x) (x ((\y.x) z)))-->
((x x) (x x))
Number of Steps:13
val it = () : unit
- countprintnewrireduce (APP(LAM("x",APP(vx,vx)),t9));
((\x.(x x)) ((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z)))-->
((\x.(x x)) ((\z.(z ((\x.x) z))) ((\y.x) z)))-->
((\x.(x x)) ((\z.(z ((\x.x) z))) x))-->
((\x.(x x)) (x ((\x.x) x)))-->
((\x.(x x)) (x x))-->
((x x) (x x))
Number of Steps:5
val it = () : unit

DO MORE EXAMPLES.....
```

- Termination: if a term has a normal form, leftmost finds it but rightmost does not.

```
-Let val O = ...
...
- printLEXP(APP(t2,O));
((\y.x) ((\x.(x x)) (\x.(x x))))val it = () : unit
- countprintloreduce (APP(t2,O));
((\y.x) ((\x.(x x)) (\x.(x x))))-->
x
Number of Steps:1
val it = () : unit
- countprintnewrireduce (APP(t2,O));
^C
```

```
Interrupt
(We could here place a limit on how much are we allowing
the system to reduce before it stops instead of us having
to interrupt it while looping.)
```

DO MORE EXAMPLES.....

DATA SHEET

At `http://www.macs.hw.ac.uk/~fairouz/foundations-2018/slides/data-files.sml`, you find an implementation in SML of the set of terms $\mathcal{M}$ and many operations on it. You can use all of these in your assignment. Anything you use from elsewhere has to be well cited/referenced.

† The syntax of the classical λ-calculus is given by $\mathcal{M} ::= \mathcal{V} \mid (\lambda\mathcal{V}.\mathcal{M}) \mid (\mathcal{M}\mathcal{M})$.
We assume the usual notational conventions in $\mathcal{M}$ and use the reduction rule:
$\underline{(\lambda v.P)Q} \rightarrow_\beta P[v := Q]$.

† The syntax of the λ-calculus in item notation is given by $\mathcal{M}' ::= \mathcal{V} \mid [\mathcal{V}]\mathcal{M}' \mid \langle\mathcal{M}'\rangle\mathcal{M}'$.
We use the reduction rule: $\underline{\langle Q'\rangle[v]}P' \rightarrow_{\beta'} [x := Q']P'$.

† In $\mathcal{M}$, (PQ) stands for the application of function P to argument Q.
† In $\mathcal{M}'$, $\langle Q'\rangle P'$ stands for the application of function P' to argument Q' (note the reverse order).

† The syntax of the classical λ-calculus with de Bruijn indices is given by
$\Lambda ::= \mathbb{N} \mid (\lambda\Lambda) \mid (\Lambda\Lambda)$.

† For $[x_1, \cdots, x_n]$ a list (not a set) of variables, we define $\omega_{[x_1,\cdots,x_n]} : \mathcal{M} \mapsto \Lambda$ inductively by:
$\omega_{[x_1,\cdots,x_n]}(v_i) = \min\{j : v_i \equiv x_j\}$
$\omega_{[x_1,\cdots,x_n]}(AB) = \omega_{[x_1,\cdots,x_n]}(A)\omega_{[x_1,\cdots,x_n]}(B)$
$\omega_{[x_1,\cdots,x_n]}(\lambda x.A) = \lambda\omega_{[x,x_1,\cdots,x_n]}(A)$
Hence $\omega_{[x,y,x,y,z]}(x) = 1$, $\omega_{[x,y,x,y,z]}(y) = 2$ and $\omega_{[x,y,x,y,z]}(z) = 5$.
Also $\omega_{[x,y,x,y,z]}(xyz) = 1\ 2\ 5$.
Also $\omega_{[x,y,x,y,z]}(\lambda xy.xz) = \lambda\lambda 2\ 7$.

Assume our variables are ordered as follows: $v_1, v_2, v_3, \cdots$.
We define $\omega : \mathcal{M} \mapsto \Lambda$ by $\omega(A) = \omega_{[v_1,\cdots,v_n]}(A)$ where $FV(A) \subseteq \{v_1, \cdots, v_n\}$.
So for example, if our variables are ordered as $x, y, z, x', y', z', \cdots$ then $\omega(\lambda xyx'.xzx') = \omega_{[x,y,z]}(\lambda xyx'.xzx') = \lambda\omega_{[x,x,y,z]}(\lambda yx'.xzx') = \lambda\lambda\omega_{[y,x,x,y,z]}(\lambda x'.xzx') = \lambda\lambda\lambda\omega_{[x',y,x,x,y,z]}(xzx') = \lambda\lambda\lambda 3\ 6\ 1$.

† The syntax of the λ-calculus in item notation is given by
$\Lambda' ::= \mathbb{N} \mid [\,]\Lambda' \mid \langle\Lambda'\rangle\Lambda'$.

† Assume the following SML datatypes which implement $\mathcal{M}$, Λ, $\mathcal{M}'$ and Λ' respectively (here, if `e1` implements A'_1 and `e2` implements A'_2, then `IAPP(e1,e2)` implements $\langle A'_1\rangle A'_2$ which stands for the function A'_2 applied to argumentA'_1):

```
datatype LEXP =
   APP of LEXP * LEXP | LAM of string *  LEXP |  ID of string;
```

```
datatype BEXP =
   BAPP of BEXP * BEXP | BLAM of BEXP |  BID of int;

datatype IEXP =
   IAPP of IEXP * IEXP | ILAM of string *  IEXP |  IID of string;

datatype IBEXP =
   IBAPP of IBEXP * IBEXP | IBLAM of    IBEXP |  IBID of int;
```