

Foundations1 assignment 2017

Fairouz Kamareddine

October 17, 2017

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

1. Just like I defined translation functions $T : \mathcal{M}' \mapsto \mathcal{M}''$ and $\omega : \mathcal{M} \mapsto \Lambda$, give translation functions from $U : \mathcal{M} \mapsto \mathcal{M}''$ and $V : \mathcal{M} \mapsto \mathcal{M}'$ and $\omega' : \mathcal{M}' \mapsto \Lambda'$. Your translation functions need to be complete with all subfunctions and needed information (just like T and ω were complete with all needed information). Submit all these functions here. (1)
2. For each of the SML terms $vx, vy, vz, t1, \dots, t9$ in <http://www.macs.hw.ac.uk/~fairouz/foundations-2017/slides/data-files.sml>, let the overlined term represent the corresponding term in \mathcal{M} . I.e., $\overline{vx} = x, \overline{vy} = y, \overline{vz} = z, \overline{t1} = \lambda x.x, \overline{t2} = \lambda y.x, \dots$.
For each of $\overline{vx}, \overline{vy}, \overline{vz}, \overline{t1}, \overline{t2}, \dots, \overline{t9}$ in \mathcal{M} , translate it into the corresponding terms of $\mathcal{M}', \mathcal{M}'', \Lambda$ and Λ' using the translation functions V, U, ω and ω' .

	V	U	ω	ω'
	x	x	1	1
	y	y	1	1
	z	z	1	1
	$[x]x$	I''	$\lambda 1$	$\square 1$
	$[y]x$	$(K''x)$	$\lambda 2$	$\square 2$
	$\langle z \rangle \langle [y]x \rangle [x]x$	$((I''(K''x))z)$	$((\lambda 1)(\lambda 2))2$	$\langle 2 \rangle \langle \square 2 \rangle \square 1$
	$\langle z \rangle [x]x$	$(I''z)$	$(\lambda 1)1$	$\langle 1 \rangle \square 1$
	$\langle \langle z \rangle \langle [y]x \rangle [x]x \rangle \langle z \rangle \langle [y]x \rangle [x]x$	$((I''(K''x))z)((I''(K''x))z))$	$(\lambda 1)(\lambda 2)2((\lambda 1)(\lambda 2)2)$	$\langle \langle 2 \rangle \langle \square 2 \rangle \square 1 \rangle \langle 2 \rangle \langle \square 2 \rangle \square 1$
	$[x][y][z] \langle \langle z \rangle y \rangle \langle z \rangle x$	S''	$\lambda \lambda \lambda 31(21)$	$\square \square \square \langle \langle 1 \rangle 2 \rangle \langle 1 \rangle 3$
	$\langle [x]x \rangle \langle [x]x \rangle [x][y][z] \langle \langle z \rangle y \rangle \langle z \rangle x$	$((S''I'')I'')$	$(\lambda \lambda \lambda 31(21))(\lambda 1)(\lambda 1)$	$\langle \square 1 \rangle \langle \square 1 \rangle \square \square \langle \langle 1 \rangle 2 \rangle \langle 1 \rangle 3$
	$[z] \langle \langle z \rangle [x]x \rangle z$	$((S''I'')I'')$	$\lambda 1((\lambda 1)1)$	$\square \langle \langle 1 \rangle \square 1 \rangle 1$
	$\langle \langle z \rangle \langle [y]x \rangle [x]x \rangle [z] \langle \langle z \rangle [x]x \rangle z$	$((S''I'')I'')((I''(K''x))z))$	$(\lambda 1((\lambda 1)1))((\lambda 1)(\lambda 2)2)$	$\langle \langle 2 \rangle \langle \square 2 \rangle \square 1 \rangle \square \langle \langle 1 \rangle \square 1 \rangle 1$
	$[x]x$	I''	$\lambda 1$	$\square 1$

3. Just like I introduced SML terms $vx, vy, vz, t1, t2, \dots, t9$ which implement terms in \mathcal{M} , please implement the corresponding terms each of the other sets $\mathcal{M}', \Lambda, \Lambda', \mathcal{M}''$. Your output must be like my output in <http://www.macs.hw.ac.uk/~fairouz/foundations-2017/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));
```

The implementation in \mathcal{M}' :

```

val vx = IID "x";
val vy = IID "y";
val vz = IID "z";

val t1 = ILAM ("x", vx);
val t2 = ILAM ("y", vx);
val t3 = IAPP (vz, IAPP (t2, t1));
val t4 = IAPP (vz, t1);
val t5 = IAPP (t3, t3);
val t6 = ILAM("x", (ILAM ("y", (ILAM ("z", IAPP( IAPP (vz, vy), IAPP
  ↪ (vz, vx))))));
val t7 = IAPP( t1, IAPP (t1, t6));
val t8 = ILAM("z", IAPP( IAPP (vz, t1), vz));
val t9 = IAPP (t3, t8);

```

The implementation in \mathcal{M}'' :

```

val vx = CID "x";
val vy = CID "y";
val vz = CID "z";

val t1 = CI;
val t2 = CAPP (CK, vx);
val t3 = CAPP (CAPP (CI, CAPP (CK, vx)), vz);
val t4 = CAPP (CI, vz);
val t5 = CAPP (CAPP (CAPP (CI, CAPP (CK, vx)), vz), CAPP (CAPP (CI,
  ↪ CAPP (CK, vx)), vz));
val t6 = CS;
val t7 = CAPP (CAPP (CS, CI), CI);
val t8 = CAPP (CAPP (CS, CI), CI);
val t9 = CAPP (CAPP (CAPP (CS, CI), CI), CAPP (CAPP (CI, CAPP (CK,
  ↪ vz)), vz));

```

The implementation in Λ :

```
val vx = BID 1;
val vy = BID 2;
val vz = BID 3;

val t1 = BLAM (BID 1);
val t2 = BLAM (BID 2);
val t3 = BAPP (BAPP (t1, t2), BID 2);
val t4 = BAPP (BLAM (BID 1), BID 1);
val t5 = BAPP (BAPP (BAPP ( BLAM (BID 1), BLAM (BID 2)),BID 2), (BAPP
  ↪ (BAPP ( BLAM (BID 1), BLAM (BID 2)),BID 2)));
val t6 = BLAM (BLAM (BLAM ( BAPP (BAPP (BID 3, BID 2), BAPP (BID 2,
  ↪ BID 1))))) );
val t7 = BAPP (BAPP ( t6, t1), t1);
val t8 = BLAM (BAPP (BID 1,BAPP( BLAM ( BID 1), BID 1)));
val t9 = BAPP (t8, t3);
```

The implementation in Λ' :

```

val vx = IBID 1;
val vy = IBID 2;
val vz = IBID 3;

val t1 = IBLAM (IBID 1);
val t2 = IBLAM (IBID 2);
val t3 = IBAPP (IBAPP (t2, t1), IBID 2);
val t4 = IBAPP (IBID 1, IBLAM (IBID 1));
val t5 = IBAPP ((IBAPP (IBID 2, IBAPP (IBLAM (IBID 2), IBLAM (IBID
  ↪ 1))))), IBAPP (IBID 2, IBAPP (IBLAM (IBID 2), IBLAM (IBID 1))));
val t6 = IBLAM (IBLAM (IBLAM ( IBAPP (IBAPP (IBID 1, IBID 2), IBAPP
  ↪ (IBID 2, IBID 3))));
val t7 = IBAPP (t1, IBAPP ( t1, t6));
val t8 = IBLAM (IBAPP (IBAPP (IBID 1, IBLAM ( IBID 1)), IBID 1));
val t9 = IBAPP (t3, t8);

```

4. For each of \mathcal{M}' , Λ , Λ' , \mathcal{M}'' , 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, \dots 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

```

In \mathcal{M}' notation:

```
fun printEXP (IID x) = print x
| printEXP (ILAM (x, y)) = (
    print "[";
    print x;
    print "]" ;
    printEXP y
)
| printEXP (IAPP (a, b)) = (
    print "<";
    printEXP a;
    print ">";
    printEXP b
);

vx:      x
vy:      y
vz:      z
t1:      [x]x
t2:      [y]x
t3:      <z><[y]x>[x]x
t4:      <z>[x]x
t5:      <<z><[y]x>[x]x><z><[y]x>[x]x
t6:      [x] [y] [z] <<z>y><z>x
t7:      <[x]x><[x]x>[x] [y] [z] <<z>y><z>x
t8:      [z] <<z>[x]x>z
t9:      <<z><[y]x>[x]x>[z] <<z>[x]x>z
```


In \mathcal{M}'' notation:

```
fun printEXP (CID v) = print v
| printEXP (CI) = print "I'"
| printEXP (CS) = print "S'"
| printEXP (CK) = print "K'"
| printEXP (CAPP (a, b)) = (
    print "(";
    printEXP a;
    printEXP b;
    print ")"
);
```

```
vx:      x
vy:      y
vz:      z
t1:      I'
t2:      (K'x)
t3:      ((I'(K'x))z)
t4:      (I'z)
t5:      (((I'(K'x))z)((I'(K'x))z))
t6:      S'
t7:      ((S'I')I')
t8:      ((S'I')I')
t9:      (((S'I')I')((I'(K'z))z))
```

In Λ notation:

```
fun printEXP (BID x) = print (Int.toString x)
| printEXP (BLAM x) = (
    print "\206\187";
    printEXP x;
    print ")"
)
| printEXP (BAPP (a, b)) = (
    print "(";
    printEXP a;
    printEXP b;
    print ")"
);
```

```
vx:      1
vy:      2
vz:      3
t1:      ( $\lambda 1$ )
t2:      ( $\lambda 2$ )
t3:      ((( $\lambda 1$ ) ( $\lambda 2$ )) 2)
t4:      (( $\lambda 1$ ) 1)
t5:      (((( $\lambda 1$ ) ( $\lambda 2$ )) 2) ((( $\lambda 1$ ) ( $\lambda 2$ )) 2))
t6:      ( $\lambda$  ( $\lambda$  ( $\lambda$  ((32) (21)))))
t7:      ((( $\lambda$  ( $\lambda$  ( $\lambda$  ((32) (21))))) ( $\lambda 1$ )) ( $\lambda 1$ ))
t8:      ( $\lambda$  (1 (( $\lambda 1$ ) 1)))
t9:      (( $\lambda$  (1 (( $\lambda 1$ ) 1))) ((( $\lambda 1$ ) ( $\lambda 2$ )) 2))
```

In Λ' notation:

```
fun printEXP (IBID x) = print (Int.toString x)
| printEXP (IBLAM x) = (
    print "[]";
    printEXP x
)
| printEXP (IBAPP (a, b)) = (
    print "<";
    printEXP a;
    print ">";
    printEXP b
);
```

```
vx:      1
vy:      2
vz:      3
t1:      [] 1
t2:      [] 2
t3:      <<[] 2> [] 1>2
t4:      <1> [] 1
t5:      <<2><[] 2> [] 1><2><[] 2> [] 1
t6:      [] [] <<1>2><2>3
t7:      <[] 1><[] 1> [] [] <<1>2><2>3
t8:      [] <<1> [] 1>1
t9:      <<<[] 2> [] 1>2> [] <<1> [] 1>1
```

5. Implement in SML the translation functions T , U and V and give these implemented functions here. (1)

$T : \mathcal{M}' \mapsto \mathcal{M}''$

```

(* Note that the ordering here is due to the inability to equality
  ↪ test during patern matching,
  * f (x, x) is erroneous in SML. Also due to the fact that any if
  ↪ statement must have an else clause.
  * Also becuase ( CID x, CID y) and (CID x, COM y) are not mutually
  ↪ exclusive *)
fun f (CID x, CAPP (a, b)) =
    if (((CID x) = b) andalso (not (Combinatorics.free (CID x)
    ↪ a))) then
        a
    else
        CAPP( CAPP (CS, f ((CID x), a)), f ((CID x), b))
| f (CID x, a) =
    if ((CID x) = a) then (* Clause 1 *)
        CI
    else (* clause 2 *)
        CAPP (CK, a);

fun toCombinatorics (IID x) = CID x
| toCombinatorics (IAPP (a, b)) = CAPP (toCombinatorics b,
  ↪ toCombinatorics a)
| toCombinatorics (ILAM (x, a)) = f (CID x, (toCombinatorics a));

```

$U : \mathcal{M} \mapsto \mathcal{M}''$

```

(* Note that the ordering here is due to the inability to equality
  ↪ test during patern matching,
  * f (x, x) is erroneous in SML. Also due to the fact that any if
  ↪ statement must have an else clause.
  * Also becuae ( CID x, CID y) and (CID x, COM y) are not mutually
  ↪ exclusive *)
fun f (CID x, CAPP (a, b)) =
    if (((CID x) = b) andalso (not (Combinatorics.free (CID x)
  ↪ a))) then
        a
    else
        CAPP( CAPP (CS, f ((CID x), a)), f ((CID x), b))
| f (CID x, a) =
    if ((CID x) = a) then (* Clause 1 *)
        CI
    else (* clause 2 *)
        CAPP (CK, a); (* See comment on supporting
  ↪ documentation for omission of conditional *)

fun toCombinatorics (ID x) = CID x
| toCombinatorics (APP (a, b)) = CAPP (toCombinatorics a,
  ↪ toCombinatorics b)
| toCombinatorics (LAM (x, a)) = f (CID x, (toCombinatorics a));

```

$V : \mathcal{M} \mapsto \mathcal{M}'$

```
fun toItem (ID x) = IID x
  | toItem (LAM (x, y)) = ILAM (x, toItem y)
  | toItem (APP (a, b)) = IAPP (toItem b, toItem a);
```

6. Test these functions on all possible translations between these various sets for all the given terms vx, vy, vz, t1, ... t9 and give your output clearly.

For example, my itran translates from \mathcal{M} to \mathcal{M}' and my 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)

7. Define the subterms in \mathcal{M}'' and implement this function in SML. You should give below the formal definition of *subterm''*, its implementation in SML and you need to test on finding the subterms for all combinator terms that correspond to vx, vy, vz, t1, ... t9. For example, if ct1 and ct2 are the terms that correspond to t1 and t2 then

```
- subterm2 ct1;
val it = [CI] : COM list
- subterm2 ct2;
val it = [CAPP (CK,CID "x"),CK,CID "x"] : COM list
```

(1)

8. Implement the combinatory reduction rules $=_c$ given in the data sheets and use your implementation to reduce all combinator terms that correspond to vx, vy, vz, t1, ... t9 showing all reduction steps. For example,

```
-creduce ct3;
ct3 =
I(Kx)z -->
Kxz -->
x
-creduce ct5;
```

```

ct5 =
I(Kx)z(I(Kx)z)-->
K x z(I(Kx)z)-->
x(I(Kx)z)-->
x(Kxz) -->
xx

```

(1)

9. For `creduce` in the above question, implement a counter that counts the number of `->`s used to reach a normal form. For example,

```

-creduce ct3;
ct3 =
I(Kx)z -->
Kxz -->
x
2 setps
-creduce ct5;
ct5 =
I(Kx)z(I(Kx)z)-->
K x z(I(Kx)z)-->
x(I(Kx)z)-->
x(Kxz) -->
xx
4 steps

```

(1)

10. Implement η -reduction on \mathcal{M} and test it on many examples of your own. Give the implementation as well as the test showing all the reduction steps one by one until you reach a η -normal form. (1)
11. Translate $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ in each of \mathcal{M}' , \mathcal{M}'' , Λ and Λ' and give the SML implementation of all these translations. (1)
12. Assume `comega` is your SML implementation of the term that corresponds to Ω . Run `-creduce comega`; and say what happens. (1)
13. Give an implementation of leftmost reduction and rightmost reduction in \mathcal{M} and test them on a number of examples that show which terminates more and which is more efficient. (2)

DATA SHEET

At <http://www.macs.hw.ac.uk/~fairouz/foundations-2017/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. You can also use any other help SML functions I have given you. 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:

$$(\lambda v. P)Q \rightarrow_{\beta} P[v := Q].$$

† 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: $\langle Q' \rangle [v]P' \rightarrow_{\beta'} [x := Q']P'$.

† 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

$$\Lambda ::= \mathbb{N} \mid (\lambda \Lambda) \mid (\Lambda \Lambda).$$

† We define free variables in the classical λ -calculus with de Bruijn indices as follows:

$$FV(n) = \{n\}, FV(AB) = FV(A) \cup FV(B) \text{ and } FV(\lambda A) = FV(A) \setminus \{1\}.$$

† For $[x_1, \dots, x_n]$ a list (not a set) of variables, we define $\omega_{[x_1, \dots, x_n]} : \mathcal{M} \mapsto \Lambda$ inductively by:

$$\omega_{[x_1, \dots, x_n]}(v_i) = \min\{j : v_i \equiv x_j\}$$

$$\omega_{[x_1, \dots, x_n]}(AB) = \omega_{[x_1, \dots, x_n]}(A)\omega_{[x_1, \dots, x_n]}(B)$$

$$\omega_{[x_1, \dots, x_n]}(\lambda x. A) = \lambda \omega_{[x, x_1, \dots, x_n]}(A)$$

$$\text{Hence } \omega_{[x, y, x, y, z]}(x) = 1, \omega_{[x, y, x, y, z]}(y) = 2 \text{ and } \omega_{[x, y, x, y, z]}(z) = 5.$$

$$\text{Also } \omega_{[x, y, x, y, z]}(xyz) = 1 \ 2 \ 5.$$

$$\text{Also } \omega_{[x, y, x, y, z]}(\lambda xy. xz) = \lambda \lambda 2 \ 7.$$

Assume our variables are ordered as follows: v_1, v_2, v_3, \dots .

We define $\omega : \mathcal{M} \mapsto \Lambda$ by $\omega(A) = \omega_{[v_1, \dots, v_n]}(A)$ where $FV(A) \subseteq \{v_1, \dots, v_n\}$.

So for example, if our variables are ordered as $x, y, z, x', y', z', \dots$ then $\omega(\lambda xyx'. xzx') = \omega_{[x, y, z]}(\lambda xyx'. xzx') = \lambda \omega_{[x, x, y, z]}(\lambda yx'. xzx') = \lambda \lambda \omega_{[y, x, x, y, z]}(\lambda x'. xzx') = \lambda \lambda \lambda \omega_{[x', y, x, x, y, z]}(xxzx') = \lambda \lambda \lambda 3 \ 6 \ 1.$

† The syntax of the λ -calculus in item notation is given by

$$\Lambda' ::= \mathbb{N} \mid []\Lambda' \mid \langle \Lambda' \rangle \Lambda'.$$

† The syntax of combinatory logic is given by

$$\mathcal{M}'' ::= \mathcal{V} \mid I'' \mid K'' \mid K'' \mid (\mathcal{M}'' \mathcal{M}'')$$

We assume that application associates to the left in \mathcal{M}'' . I.e., $P''Q''R''$ stands for $((P''Q'')R'')$.

We use the reduction rules:

$$(I'') \underline{I''}v =_c v \quad (K'') \underline{K''}v_1v_2 =_c v_1 \quad (K'') \underline{K''}v_1v_2v_3 =_c v_1v_3(v_2v_3).$$

Note that these rules are from left to right (and not right to left) even though they are written with an $=$ sign.

† We define free variables in combinatory logic as follows:

$$\begin{aligned} FV''(v) &= \{v\} \\ FV''(I'') &= FV''(K'') = FV''(K'') = \{\} \\ FV''(P''Q'') &= FV''(P'') \cup FV''(Q''). \end{aligned}$$

† Here is a possible translation function T from \mathcal{M}' to \mathcal{M}'' :

$T(v) = v$ $T([v]P') = f(v, T(P'))$ $T(\langle Q' \rangle P') = (T(P')T(Q'))$ where
 f takes a variable and a combinator-term and returns a combinator term according to the following numbered clauses:

1. $f(v, v) = I''$
2. $f(v, P'') = K''P''$ if $v \notin FV(P'')$
3. $f(v, P_1''P_2'') = \begin{cases} P_1'' & \text{if } v \notin FV(P_1'') \text{ and } P_2'' \equiv v \\ S''f(v, P_1'')f(v, P_2'') & \text{otherwise.} \end{cases}$

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

datatype COM = CAPP of COM*COM | CID of string | CI | CK | CS;
```