Foundations1 assignment 2017

Fairouz Kamareddine

October 8, 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, $\cdots$ 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., $\overline{\text{vx}} = x$, $\overline{\text{vy}} = y$, $\overline{\text{vz}} = z$, $\overline{\text{t1}} = \lambda x.x$, $\overline{\text{t2}} = \lambda y.x$, $\cdots$.
   For each of $\overline{\text{vx}}$, $\overline{\text{vy}}$, $\overline{\text{vz}}$, $\overline{\text{t1}}$, $\overline{\text{t2}}$, $\cdots \overline{\text{t9}}$ in $\mathcal{M}$, translate it into the corresponding terms of $\mathcal{M}'$, $\mathcal{M}''$, Λ and Λ' using the translation functions V, U, ω and ω'.
   Your output should be tidy as follows:

	V	U	ω	ω'
$\lambda x.x$	$[x]x$	I''	$\lambda 1$	$[\,]1$

   (1)

3. Just like I introduced SML terms vx, vy, vz, t1, t2, $\cdots$ t9 which implement terms in $\mathcal{M}$, please implement the corresponding terms each of the other sets $\mathcal{M}'$, Λ, Λ', $\mathcal{M}''$. 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));
```

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, $\cdots$ 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
```

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

6. Test these functions on all possible translations between these various sets for all the given terms vx, vy, vz, t1, $\cdots$ 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, $\cdots$ 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, $\cdots$ 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:
$\underline{(\lambda v.P)Q} \to_\beta P[v := Q]$.

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

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

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

† The syntax of combinatory logic is given by
$\mathcal{M}'' ::= \mathcal{V} \mid \mathrm{I}'' \mid \mathrm{K}'' \mid \mathrm{S}'' \mid (\mathcal{M}''\mathcal{M}'')$
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'') $\underline{\mathrm{I}''v} =_c v$ (K'') $\underline{\mathrm{K}''v_1v_2} =_c v_1$ (S'') $\underline{\mathrm{S}''v_1v_2v_3} =_c v_1v_3(v_2v_3)$.

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:
$FV''(v) = \{v\}$
$FV''(\mathrm{I}'') = FV''(\mathrm{K}'') = FV''(\mathrm{S}'') = \{\}$
$FV''(P''Q'') = FV''(P'') \cup FV''(Q'')$.

† Here is a possible translation function T from $\mathcal{M}'$ to $\mathcal{M}''$:
$T(v) = v$ $\quad T([v]P') = f(v, T(P'))$ $\quad T(\langle Q'\rangle P') = (T(P')T(Q'))$ 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 `e1` implements A_1' and `e2` implements A_2', then `IAPP(e1,e2)` implements $\langle A_1'\rangle A_2'$ which stands for the functionA_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;

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