Foundations1 assignment 2016

Fairouz Kamareddine

This assignment is worth 15% of the total mark. It is handed Tuesday of week 5 and due back on Monday of week 9 at noon. You need to have it stamped with the submission date from the students' office and put it in the assignment box provided by the student's office. Please type your solutions. No handwritten solutions. Good luck.

---

1 Lambda Calculus in Item Notation

1.1 Recall Usual Lambda calculus

- Let $\mathcal{V} = \{x, y, z, \ldots\}$ and let $v, v', v_1, v_2, \ldots$ range over $\mathcal{V}$.
- Let $\mathcal{M}$ be the set of terms of the λ-calculus and let $A, B, C, \ldots$ range over $\mathcal{M}$. We can also index metavariables: i.e., also, $A_1, A_2, B_1, ...$ are metavariables that range over $\mathcal{M}$.
- Recall that if $A \in \mathcal{M}$ then $A ::= v|(A_1A_2)|(\lambda v.A_1)$.
- At `http://www.macs.hw.ac.uk/~fairouz/foundations-2016/slides/data-files.sml`, you find an implementation in SML of the set of terms $\mathcal{M}$ and many operations on it.

1.2 Item Lambda calculus

- Let $\mathcal{V} = \{x, y, z, \ldots\}$ and let $v, v', v_1, v_2, \ldots$ range over $\mathcal{V}$.
- Take $\mathcal{M}'$ to be a set of terms over which $A', B', C', \ldots$ range, where $A' ::= v|\ [v]A'_1|\langle A'_2\rangle A'_1$. Also here we can index metavariables: i.e., also, $A'_1, A'_2, B'_1, ...$ are metavariables that range over $\mathcal{M}'$. Note that we have no parenthesis and no dots.
- In $\mathcal{M}'$, you must understand $\langle A'_2\rangle A'_1$ to mean that A'_2 is input to A'_1 (i.e., that A'_1 is applied to A'_2).

- Call forms like $\langle A' \rangle$ and $[v]$, *wagons*. Here, $\langle A' \rangle$ is an applicator-wagon and $[v]$ is an abstractor-wagon. You can see that any term of $\mathcal{M}'$ is a number of wagons followed by a variable (which we call the heart of the term). E.g., the term $\langle z \rangle \langle [y] y \rangle [x] \langle y \rangle x$ consists of the four wagons $\langle z \rangle$ and $\langle [y] y \rangle$ and $[x]$ and $\langle y \rangle$ and the heart x.

- Call any two wagons next to each other of the form $\langle A' \rangle [v]$ lovers. I.e., $\langle A' \rangle]$ and $[v]$ are lovers.

- Call any wagon $\langle A' \rangle$ which is not immediately left of a wagon of the form $[v]$, a bachelor.

- Call any wagon $[v]$ which is not immediately right of a wagon of the form $\langle A' \rangle$, a bachelor.

1.3 The assignment, theoretical

1. Give a translation function $\mathcal{I}$ from $\mathcal{M}$ to $\mathcal{M}'$ and another translation function $\mathcal{O}$ from $\mathcal{M}'$ to $\mathcal{M}$ such that $\mathcal{I}(\mathcal{O}(A')) = A'$ and $\mathcal{O}(\mathcal{I}(A)) = A$. Give examples how your translations work. For example:

   - $\mathcal{I}((\lambda x.xy)(\lambda y.y)z) = \langle z \rangle \langle [y] y \rangle [x] \langle y \rangle x$ and
   - $\mathcal{O}(\langle z \rangle \langle [y] y \rangle [x] \langle y \rangle x) = (\lambda x.xy)(\lambda y.y)z$.

2. Recall the definitions of free variables, substitution, $\rightarrow_\alpha$, $\rightarrow_\beta$, $\rightarrow_\eta$, $\twoheadrightarrow_\alpha$, $\twoheadrightarrow_\beta$, $\twoheadrightarrow_\eta$, and $=$ for $\mathcal{M}$ and redefine these notions for $\mathcal{M}'$.

1.4 The assignment, implementation

1. Now, look at the file data-files.sml available at `http://www.macs.hw.ac.uk/~fairouz/foundations-2016/slides/data-files.sml`.

   - Run the functions in data-files.sml on the commands below and submit for each of these commands below the sml output from YOUR RUNNING of the functions in data-files.sml. For each of these commands, write next to your output what you think is taking place.
     freeVars t1;
     freeVars t2;
     freeVars t3;
     freeVars t4;
     free "x" vx;

```
free "x" t1;
free "x" t2;
subs vy "x" t2;
subs vx "y" t2;
printLEXP t3;
print "mreduce";
print "\n";
mreduce t3;
printmreduce t3;
print "\n";
printLEXP t3;
print "rireduce";
print "\n";
printLEXP t3;
rireduce t3;
printrireduce t3;
printLEXP t3;
print "loreduce"; print "\n"; printLEXP t3;
loreduce t3;
printloreduce t3;
printLEXP t5;
print "mreduce";
print "\n";
mreduce t5;
printmreduce t5;
print "\n";
printLEXP t5;
print "rireduce";
print "\n";
printLEXP t5;
rireduce t5;
printrireduce t5;
printLEXP t5;
print "loreduce";
print "\n";
printLEXP t5;
loreduce t5;
printloreduce t5;
printLEXP t7;
print "mreduce";
```

```
print "\n";
mreduce t7;
printmreduce t7;
print "\n";
printLEXP t7;
print "rireduce";
print "\n";
printLEXP t7;
rireduce t7;
printrireduce t7;
printLEXP t7;
print "loreduce";
print "\n";
printLEXP t7;
loreduce t7;
printloreduce t7;
```

- Write a recursive type IEXP of the expressions of $\mathcal{M}'$.
- Write SML terms Ivx, Ivy, Ivz, It1, It2, ..It8 of IEXP which correspond to the terms vx, vy, vz, t1, t2, ..t8.
- Write other SML terms too of IEXP (e.g., It9 (the term which applies It8 to It3) and It10 (the term which applies It8 to It8)).
- For each function defined in data-files.sml, write a corresponding SML function on IEXP. Hence, write the SML functions Ifree, Ifindme, IfreeVars, Isubs, printIEXP, Iaddlam, Iaddbackapp, Iaddfrontapp, Iprintlistreduce, Iis_var, etc.
- Test these functions using the terms It1, It2, ..It10 and other interesting terms. Submit your program output for these tests. Your tests should illustrate that your functions work properly.

2. Implement the translation function $\mathcal{I}$ you gave in section 1.3. That is, write an SML function ClassicalItem which translates terms of $\mathcal{M}$ into terms of $\mathcal{M}'$ (i.e., translates LEXP into IEXP). Test this function with things like:
   printIEXP(ClassicalItem t9) and anything else you feel interesting and which illustrates that your function works as expected. Submit the results of your tests.

3. Implement the translation function $\mathcal{O}$ you gave in section 1.3. That is, write an SML function ItemClassical which translates terms of $\mathcal{M}'$

into terms of $\mathcal{M}$ (i.e., translates IEXP into LEXP). Test this function with things like:
printLEXP(ItemClassical It5) and anything else you feel interesting and which illustrates that your function works as expected. Submit the results of your tests.

4. Recall that the leftmost redex on $\mathcal{M}$ is defined as follows:

   - leftmost(v) = undefined
   - leftmost($\lambda v.A$) = leftmost(A)
   - leftmost(AB) = if AB is a beta-redex then AB else if A has a beta-redex then leftmost(A) else leftmost(B).

   Define an SML function lreduce on LEXP which takes an LEXP and reduces it using the leftmost strategy (i.e., it always reduces the leftmost redex) until a normal form is found. lreduce should return the original term, the final term in normal form and all the terms in between.
   Write an SML function printlreduce on LEXP which prints nicely the leftmost reduction path from the original term until the normal form. For example:
   - printlreduce t8;
   (\z.(z ((\x.x) z))→
   (\z.(z z))
   - printlreduce t9;
   ((\z.(z ((\x.x) z))) (((\x.x) (\y.x)) z))→
   ((((\x.x) (\y.x)) z) ((\x.x) (((\x.x) (\y.x)) z)))→
   (((\y.x) z) ((\x.x) (((\x.x) (\y.x)) z)))→
   (x ((\x.x) (((\x.x) (\y.x)) z)))→
   (x (((\x.x) (\y.x)) z))→
   (x ((\y.x) z))→
   (x x)

   Test printlreduce on many interesting examples. Say what happens to terms which do not terminate.

5. On $\mathcal{M}'$, the leftmost redex will be defined as:

   - Ileftmost(v) = undefined
   - Ileftmost($[v]A'$) = Ileftmost(A')

- Ileftmost($\langle B' \rangle A'$) = if $\langle B' \rangle A'$ is a beta-redex then $\langle B' \rangle A'$ else if A' has a beta-redex then Ileftmost(A') else Ileftmost(B').

Write an SML function Ilreduce which reduces terms of IEXP to normal form by reducing the Ileftmost redex.
Again, write an SML function printIlreduce which prints the path from the term to its normal form nicely. For example:
- printIlreduce It8;
$[z]\langle\underline{\langle z \rangle [x]} x\rangle z \rightarrow$
$[z]\underline{\langle z \rangle z}$

Test printIlreduce on many interesting examples. Say what happens to terms which do not terminate.

6. Add a counter to count the number of reduction steps Ilreduce needs to do to reach the normal form of a term. For example, printIlreduce t8 does one step (how many steps do you think printIlreduce does). Make sure your program says for each term how many reduction steps are needed by printIlreduce to reach the normal form using the leftmost strategy. What would you say if the term does not terminate?

7. Repeat the above two questions but for the rightmost reduction strategy where the redex contracted is defined as follows:

   - $rm(v)$ = undefined
   - $rm([v]A') = rm(A')$
   - $rm\langle B' \rangle A' = rm(B')$ if $rm(B')$ is defined
   - $rm(\langle B' \rangle A') = \langle B' \rangle A'$ if $rm(B')$ is undefined and $\langle B' \rangle A'$ is a β-redex
   - $rm(\langle B' \rangle A') = rm(A')$ if $rm(B')$ is undefined and $\langle B' \rangle A'$ is not a β-redex

   In particular, you must write an SML function printIrreduce on IEXP which prints nicely the rightmost reduction path from the original term until the normal form. You also need a counter which counts the number of reduction steps that printIrreduce needs to do before reaching the normal form.

   Test printIrreduce on many interesting examples. Say what happens to terms which do not terminate. Make sure your program says for each term how many reduction steps are needed by printrreduce to

reach the normal form using the rightmost strategy. What would you say if the term does not terminate?

8. Give many examples which show that printIrreduce is more efficient than printIlreduce.

9. Give many examples which show that printIlreduce may terminate when printIrreduce loops.

1.5 What should you submit?

- Your paper submitted assignment should contain in typed form the answers to all the questions in Sections 1.3 and 1.4, as well as the code of the program and the output of the tested examples. All your examples must be run using your program and the output of your program must be clearly shown. Please add a section at the end of your assignment explaining the limitations of your program.

- In addition to submitting the typed paper material mentioned above, please email me at the same time a message which ONLY contains the SML code of your program.

Good luck.