COMP6463 (Semester 2, 2013) Assignment for Typed Lambda Calculus Some Solutions and Comments

October 28, 2013

General Comments: Most importantly, *read the instructions.* In this assignment you were asked for some questions to show *all* the steps of an algorithm taught in lectures, but in others just to show clearly how you got your answers. If you don't read such instructions you risk either causing yourself extra work or failing to provide what was wanted.

When submitting scanned copies of handwritten material, please

- use a black or dark blue pen (*not* pencil) with a thick or medium point
- *don't* write right to the edge of the paper, as many printers cannot print to the edge of the paper

Note: I haven't included full solutions, particularly for those questions which were generally well done. (So what I have written isn't necessarily a sufficient or satisfactory answer.) If you have queries about any of them, contact me: phone 57778 or email jeremy.dawson@anu.edu.au

Question 1 (10 marks) Find a most general unifier for the set of pairs

$$E = \{(\alpha \to \gamma \to \delta, \alpha' \to \beta'), (\alpha \to \beta \to \gamma, \beta' \to \alpha')\}$$

. Show *all* the steps of the unification algorithm shown in lectures.

This was generally well done, except for one thing: when you obtain a substitution you must apply it to *both*

- the remainder of the problem (ie, to get the new E), *and*
- the right-hand sides of the substitutions which you have previously obtained

One answer is $\{\gamma := \beta, \delta := \beta, \alpha := \beta \to \beta, \alpha' := \beta \to \beta, \beta' := \beta \to \beta\}$

Question 2 (25 marks) For each term below, use the Principal Type Algorithm (PT) to determine whether it is typeable in λ_{Cu}. If it is typeable, give a principal type and a principal deduction. Show *all* the steps used in applying the PT Algorithm and in finding any necessary unifiers.

(a) $\lambda f \lambda z.\ f\,(f\,z)$

(b) $\lambda x \lambda y.\ x$

(c) $(\lambda f \lambda z.\ f\,(f\,z))\ (\lambda x \lambda y.\ x)$ (note, this (a) applied to (b))

Again, this was mostly done well. One important point is that in a typing judgement, $\Gamma \vdash M : \tau$, Γ is a *typing context*, whose members are *variables* and their types, such as $x : \tau$. The typing context does *not* contain arbitrary *terms* and their types (such as $M : \tau$).

Answers (without showing all the steps of the algorithms) are

(a) $\lambda f \lambda z.\ f\,(f\,z) : (\alpha \to \alpha) \to \alpha \to \alpha$

(b) $\lambda x \lambda y.\ x : \alpha \to \beta \to \alpha$

(c) Cannot be typed because we cannot unify $(\alpha \to \beta \to \alpha) \to \gamma$ (type of function with argument $\lambda x \lambda y.\ x$) with $(\alpha' \to \alpha') \to \alpha' \to \alpha'$ (type of $\lambda f \lambda z.\ f\,(f\,z)$)

Question 3 (15 marks) One way of encoding natural numbers in λ-calculus is via the so-called *Church numerals*. A number n is encoded as the λ-term $\lambda f \lambda x. f^n x$, where $f^n x$ means f applied n times to x.

For example, 0 is represented as $\lambda f \lambda x. x$; 1 is represented as $\lambda f \lambda x. f\,x$, 2 is represented as $\lambda f \lambda x. f\,(f\,x)$ and so on.

Write $C_1 = \lambda f \lambda x. f\,x$, $C_2 = \lambda f \lambda x. f\,(f\,x)$, $C_3 = \lambda f \lambda x. f\,(f\,(f\,x))$, etc. Thus $C_i\ f = f^i$ for any i.

Since $f^{mn} = (f^m)^n$, we can define multiplication of these Church numerals by (T for "times")

$$T\ C_m\ C_n\ f = C_m(C_n f), \qquad \text{that is,} \qquad T = \lambda m \lambda n \lambda f.\ m\,(n\,f)$$

Likewise, since $f^{m+n}(x) = f^m(f^n(x))$ we can define addition of Church numerals by (P for "plus")

$$P\ C_m\ C_n\ f\ x = C_m\,f\,(C_n\,f\,x) \qquad \text{that is,} \qquad P = \lambda m \lambda n \lambda f \lambda x.\ m\,f\,(n\,f\,x)$$

Since T and P are both functions which take two Church numerals as arguments and produce a third Church numeral:

(a) Find the principal types for T and for P (using the definitions of T and P above)

(b) Can the types of T and of P be unified to a common type? What is it?

(c) How does this compare with the type you might expect T and P to have, given that they are binary operators on Church numerals? Why?

Briefly, to obtain the type of $\lambda m \lambda n \lambda f.\ m\,(n\,f)$, you can reason like this: n has a function type, say $\alpha \to \beta$, with $f : \alpha$ and $n\,f : \beta$. Likewise m has a function type, with argument of type β, so let $m : \beta \to \gamma$, and $m\,(n\,f) : \gamma$. Thus

$$\lambda m \lambda n \lambda f.\ m\,(n\,f) : (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$$

Likewise, for $\lambda m \lambda n \lambda f \lambda x.\ m\,f\,(n\,f\,x)$, n is a function with two curried arguments, say $n : \alpha \to \beta \to \gamma$, with $f : \alpha$, $x : \beta$ and $n\,f\,x : \gamma$. Then m is also a function with two curried arguments, with arguments having types α and γ, so let $m : \alpha \to \gamma \to \delta$, and so $m\,f\,(n\,f\,x) : \delta$. Thus

$$\lambda m \lambda n \lambda f \lambda x.\ m\,f\,(n\,f\,x) : (\alpha \to \gamma \to \delta) \to (\alpha \to \beta \to \gamma) \to \alpha \to \beta \to \delta$$

To unify these types, we rename the type variables for P to

$$\lambda m \lambda n \lambda f \lambda x.\ m\,f\,(n\,f\,x) : (\alpha' \to \gamma' \to \delta') \to (\alpha' \to \beta' \to \gamma') \to \alpha' \to \beta' \to \delta'$$

To unify these we will first apply steps 4c and 4a repeatedly, to give

$$E = \{(\beta, \alpha'), (\gamma, \gamma' \to \delta'), (\alpha, \alpha'), (\beta, \beta' \to \gamma'), (\alpha, \alpha'), (\gamma, \beta' \to \delta')\}$$

and (skipping the details) one unifier is

$$U = \{\gamma' := \beta', \alpha' := \beta' \to \beta', \beta := \beta' \to \beta', \alpha := \beta' \to \beta', \gamma := \beta' \to \delta'\}$$

giving the common type for T and P of

$$((\beta' \to \beta') \to \beta' \to \delta') \to ((\beta' \to \beta') \to \beta' \to \beta') \to (\beta' \to \beta') \to \beta' \to \delta'$$

Since a Church numeral, in general, is $\lambda f \lambda x. f^n x$, where $f^n x$ means f applied n times to x, it will use f of type $f : \alpha \to \alpha$ and so the Church numeral will have type $(\alpha \to \alpha) \to \alpha \to \alpha$. Thus a binary operation on Church numerals will have type

$$((\alpha \to \alpha) \to \alpha \to \alpha) \to ((\alpha \to \alpha) \to \alpha \to \alpha) \to (\alpha \to \alpha) \to \alpha \to \alpha$$

This is less general than the common type found above for P and T — this means that T and P, as defined, can both operate on two arguments which are not necessarily of the same type as Church numerals.

Question 4 (10 marks) Consider the term $(\lambda x.\ x)\ x$

(a) In this term, identify the free and bound occurrences of x

(b) Give a typing derivation for this term (Hint: you may need to use α-equivalence)

The first x is called a "binding occurrence" of x, the second one is bound, the last is free.

$$\dfrac{\dfrac{x : \alpha, y : \alpha \vdash y : \alpha}{x : \alpha \vdash \lambda y.\ y : \alpha \to \alpha} \quad x : \alpha \vdash x : \alpha}{x : \alpha \vdash (\lambda y.\ y)\ x : \alpha}$$

and $(\lambda y.\ y)\ x$ is α-equivalent to $(\lambda x.\ x)\ x$

Question 5 (20 marks) Assuming the substitution lemma,

(a) prove that the basic β-reduction step, $(\lambda x.M)\ N \longrightarrow_\beta M[x := N]$, preserves the type: that is, if $(\lambda x.M)\ N : \tau$ then $M[x := N] : \tau$

(b) from this, prove that any single step β-reduction of a term (including where the reduction takes place at a subterm of the term) preserves the type of the term

Some important general comments:

- If you know or are assuming that $\lambda x.M$ has a type derivation, you know the last step of it must look like this
$$\dfrac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta}$$
This is *not* because, in general, a rule such as the $\to$-intro rule can be reversed; rather it is because, when you look at *all* the rules, you find the only one which gives a type to a term of the form $\lambda x.M$ is the $\to$-intro rule.
Likewise, for a term of the form $M\ N$, the *only* rule which gives it a type is the $\to$-elim rule. It is for this reason that if you know that $M\ N : \beta$, then you can say that for some α, it must hold that $M : \alpha \to \beta$ and $N : \alpha$.
- In relation to this result (ie, type preservation of β-reduction), there are two aspects which call for an inductive proof.
Firstly, where there are multiple steps of β-reduction (as in the answer to Question 6(b)), it must be proved that a single step preserves the type, and from that, by induction over the number of steps, that a multi-step β-reduction preserves the type. But here, the question refers only to a single step, so the induction for a multi-step β-reduction is not needed.
Secondly, a single step β-reduction can be shown to fit the definition of β-reduction by a "multi-step" argument. For example,
$$\dfrac{\dfrac{(\lambda x.M)\ N \longrightarrow_\beta M[x := N]}{P((\lambda x.M)\ N) \longrightarrow_\beta P(M[x := N])}}{\lambda y.\ P((\lambda x.M)\ N) \longrightarrow_\beta \lambda y.\ P(M[x := N])}$$
Here there are multiple steps in showing that the reduction in the last line is in fact a β-reduction, and this also requires a proof by induction.
- This induction could be formulated in several different ways. To prove that each β-reduction shown above preserves the type you want to assume that the β-reduction on the line above it preserves the type.
For this you could assume, as the inductive hypothesis, that each β-reduction which is shown to be a β-reduction using fewer steps of the rules (on slide 6, lecture 2) preserves type.
Or, in looking at $M \longrightarrow_\beta M'$, you could assume by induction that any β-reduction $N \longrightarrow_\beta N'$ where N is a smaller term than M, satisfies type preservation.
But a good proof would make it clear what is the basis of the inductive argument used.

Now for the answers.

(a) If $\Gamma \vdash (\lambda x.M)N : \beta$ then the last steps of the typing derivation must be

$$\dfrac{\dfrac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.M : \alpha \to \beta} \quad \Gamma \vdash N : \alpha}{\Gamma \vdash (\lambda x.M)N : \beta}$$

(by the first comment above). Then using the top two of these typing judgements as the premises of the substitution lemma gives $\Gamma \vdash M[x := N] : \beta$, as required.

(b) This requires an inductive argument (see third comment above). The base case of the induction is part (a) above, and then we need to look at the three rules on slide 6, lecture 2, namely

$$\dfrac{M \longrightarrow_\beta N}{M\ P \longrightarrow_\beta N\ P} \qquad \dfrac{M \longrightarrow_\beta N}{P\ M \longrightarrow_\beta P\ N} \qquad \dfrac{M \longrightarrow_\beta N}{\lambda x.M \longrightarrow_\beta \lambda x.N}$$

We need to look at each of the three rules, assuming in each that the β-reduction above the line preserves the type, and showing that the β-reduction below the line preserves the type.
For the first rule, assume that $\Gamma \vdash M\ P : \beta$. Then this must be derived by

$$\dfrac{\Gamma \vdash M : \alpha \to \beta \quad \Gamma \vdash P : \alpha}{\Gamma \vdash M\ P : \beta}$$

Then as $M \longrightarrow_\beta N$, we have by induction $\Gamma \vdash N : \alpha \to \beta$, and then the typing derivation

$$\dfrac{\Gamma \vdash N : \alpha \to \beta \quad \Gamma \vdash P : \alpha}{\Gamma \vdash N\ P : \beta}$$

For the second rule, assume that $\Gamma \vdash P\ M : \beta$. Then this must be derived by

$$\dfrac{\Gamma \vdash P : \alpha \to \beta \quad \Gamma \vdash M : \alpha}{\Gamma \vdash P\ M : \beta}$$

Then as $M \longrightarrow_\beta N$, we have by induction $\Gamma \vdash N : \alpha$, and then the typing derivation

$$\dfrac{\Gamma \vdash P : \alpha \to \beta \quad \Gamma \vdash N : \alpha}{\Gamma \vdash P\ N : \beta}$$

For the third rule, assume that $\Gamma \vdash \lambda x.\ M : \tau$. Then this must be derived by

$$\dfrac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x.\ M : \alpha \to \beta}$$

where $\tau = \alpha \to \beta$, see first comment above. Then as $M \longrightarrow_\beta N$, we have by induction $\Gamma, x : \alpha \vdash N : \beta$, and then the typing derivation

$$\dfrac{\Gamma, x : \alpha \vdash N : \beta}{\Gamma \vdash \lambda x.\ N : \alpha \to \beta}$$

Question 6 (20 marks) We had an example in lectures of $M \longrightarrow_\beta N$, where N is typeable but M is not. In that example, the explanation was that M is an abstraction which ignored its argument, and N is an untypeable argument. Here is an example with a different explanation of how this can happen.

(a) Show that $(\lambda x.\ x\ x)\ (\lambda y.\ y)$ is not typeable.

(b) Find its β-normal form.

(c) Show that its β-normal form is typeable.

(d) Can you explain how β-reduction changes an untypeable term to a typeable one in this case?

(a) The subterm $x\,x$ is not typeable, because you get

$$x : \alpha \vdash x : \alpha \qquad x : \beta \vdash x : \beta$$

You then need to unify α with β (types of the free variable x) and also α with $\beta \to \gamma$ (because $x : \alpha$ is a function that takes $x : \beta$ as argument). (See PT algorithm, case IV). You then get to trying to unify $(\beta, \beta \to \gamma)$, which fails.

(b) β-reduction gives $(\lambda x.\ x\ x)\ (\lambda y.\ y) \longrightarrow_\beta (\lambda y.\ y)\ (\lambda y.\ y) \longrightarrow_\beta (\lambda y.\ y)$

(c)

$$\frac{x : \alpha \vdash x : \alpha}{\lambda x.\ x : \alpha \to \alpha}$$

(d) Some students said that anything of the form $M\ M$ is not typeable. This is not correct. Where x is a free variable, $x\,x$ is not typeable because the occurrences of x must have the same type.

But $(\lambda y.\ y)\ (\lambda y.\ y)$ (the intermediate stage of the β-reduction) *is* typeable, because the two occurrences of $(\lambda y.\ y)$ have different types: the first is $(\lambda y.\ y) : (\alpha \to \alpha) \to \alpha \to \alpha$ whereas the second is $(\lambda y.\ y) : \alpha \to \alpha$.

The key point for getting the marks here is that in $x\,x$ the two occurrences of x must have the same type, but the two occurrences of $(\lambda y.\ y)$ may have different types.

In terms of the PT algorithm, case IV, we have

$$\vdash (\lambda y.\ y) : \beta \to \beta \qquad \vdash (\lambda y.\ y) : \alpha \to \alpha$$

The type unification required here is just $E = \{(\beta \to \beta, (\alpha \to \alpha) \to \gamma)\}$. Unlike when we apply case IV of the PT algorithm to $x\,x$, there are no free variables in $(\lambda y.\ y)$ which must have the same type in the two occurrences of it.