

Supervision 3

Lent Term 2027 · Michael Lee

●●●● difficulty · ★ optional

WORK DUE	A single PDF, 48 hours before the supervision
EXPECTED TIME	5 hours 50 min (+ 10 min for the optional questions)
OPTIONAL	★ marks a question, or a part of one, you may skip

PREREQUISITES Lecture 11 · TIME 2 hours 10 min
(+ 10 min)

1 The Lambda Calculus: Normal Forms

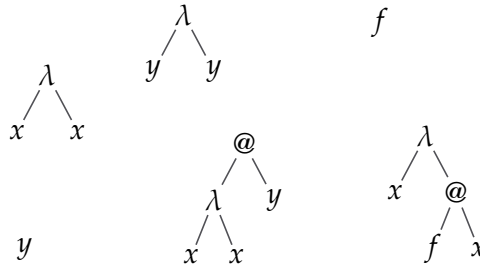
QUESTION 1

CONCEPTUAL · 40 MIN

●●●● CORE

This question considers different notions of equivalence in the lambda calculus

Consider the following lambda terms.



- Divide these lambda terms into groups that share the same β -normal form, up to α -equivalence.
- Divide these lambda terms into equivalence classes, where all lambda terms in the same equivalence class are contextually equivalent — they behave the same way.
- Another notion of equivalence is α -equivalence. Formally, α -equivalence is defined as follows:

$$\begin{array}{c}
 \frac{}{x \equiv_{\alpha} x} \text{ (VAR)} \quad \frac{z \# (MN) \quad M\{z/x\} \equiv_{\alpha} N\{z/y\}}{\lambda x. M \equiv_{\alpha} \lambda y. N} \text{ (LAM)} \\
 \\
 \frac{M \equiv_{\alpha} M' \quad N \equiv_{\alpha} N'}{MN \equiv_{\alpha} M'N'} \text{ (APP)}
 \end{array}$$

Divide these lambda terms into α -equivalence classes.

- A third notion of equivalence is β -equivalence, defined as follows:¹

¹Use the notes' definition of $M[N/x]$ without the α -renaming step, so (BETA) applies only when no free variable of N is captured.

$$\begin{array}{c}
\frac{}{(\lambda x. M) N \equiv_{\beta} M[N/x]} \text{ (BETA)} \quad \frac{M \equiv_{\beta} M'}{\lambda x. M \equiv_{\beta} \lambda x. M'} \text{ (LAM)} \\
\frac{M \equiv_{\beta} M' \quad N \equiv_{\beta} N'}{M N \equiv_{\beta} M' N'} \text{ (APP)} \\
\frac{}{M \equiv_{\beta} M} \text{ (REFL)} \quad \frac{M \equiv_{\beta} M'}{M' \equiv_{\beta} M} \text{ (SYM)} \quad \frac{M \equiv_{\beta} M' \quad M' \equiv_{\beta} M''}{M \equiv_{\beta} M''} \text{ (TRANS)}
\end{array}$$

Divide these lambda terms into β -equivalence classes.

- (e) A fourth notion of equivalence is η -equivalence, defined as follows:

$$\begin{array}{c}
\frac{x \notin \text{fv}(M)}{\lambda x. M x \equiv_{\eta} M} \text{ (ETA)} \quad \frac{M \equiv_{\eta} M'}{\lambda x. M \equiv_{\eta} \lambda x. M'} \text{ (LAM)} \quad \frac{M \equiv_{\eta} M' \quad N \equiv_{\eta} N'}{M N \equiv_{\eta} M' N'} \text{ (APP)} \\
\frac{}{M \equiv_{\eta} M} \text{ (REFL)} \quad \frac{M \equiv_{\eta} M'}{M' \equiv_{\eta} M} \text{ (SYM)} \quad \frac{M \equiv_{\eta} M' \quad M' \equiv_{\eta} M''}{M \equiv_{\eta} M''} \text{ (TRANS)}
\end{array}$$

Divide these lambda terms into η -equivalence classes.

- (f) A fifth notion of equivalence is $\alpha\beta$ -equivalence, which relates terms that are related under either α , or β , or some combination of the two. Formally:

$$\frac{M \equiv_{\alpha} M'}{M \equiv_{\alpha\beta} M'} \text{ (ALPHA)} \quad \frac{M \equiv_{\beta} M'}{M \equiv_{\alpha\beta} M'} \text{ (BETA)} \quad \frac{M \equiv_{\alpha\beta} M' \quad M' \equiv_{\alpha\beta} M''}{M \equiv_{\alpha\beta} M''} \text{ (TRANS)}$$

Divide these lambda terms into $\alpha\beta$ -equivalence classes.

- (g) A sixth notion of equivalence is $\alpha\beta\eta$ -equivalence, which relates terms that are related under α , β , or η , or any combination of the three. Formally:

$$\begin{array}{c}
\frac{M \equiv_{\alpha} M'}{M \equiv_{\alpha\beta\eta} M'} \text{ (ALPHA)} \quad \frac{M \equiv_{\beta} M'}{M \equiv_{\alpha\beta\eta} M'} \text{ (BETA)} \quad \frac{M \equiv_{\eta} M'}{M \equiv_{\alpha\beta\eta} M'} \text{ (ETA)} \\
\frac{M \equiv_{\alpha\beta\eta} M' \quad M' \equiv_{\alpha\beta\eta} M''}{M \equiv_{\alpha\beta\eta} M''} \text{ (TRANS)}
\end{array}$$

Divide these lambda terms into $\alpha\beta\eta$ -equivalence classes.

- (h) Explain which notion of equivalence is used by the notes.

QUESTION 2

CONCEPTUAL · 10 MIN

●●●●● BASIC

Suggest an algorithm for deciding if two terms are α -equivalent.

HINT The simplest approach involves defining a new representation in which you get α -equivalence for free. You have been taught one such representation in Part IB Semantics of Programming Languages.

QUESTION 3

CONCEPTUAL · 30 MIN

● ● ● ● ● BASIC

- (a) Fully reduce the following lambda term to its β -normal form using normal-order reduction

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

- (b) Fully reduce the following lambda term to its β -normal form using normal-order reduction

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

- (c) In what sense is β -reduction non-deterministic? In what sense is it deterministic?
- (d) Is normal-order reduction deterministic?
- (e) Using an example, demonstrate the difference between call-by-value and normal-order reduction.
- (f) Using an example, demonstrate the difference between call-by-name and normal-order reduction.

QUESTION 4

CONCEPTUAL · 30 MIN

● ● ● ● ● CORE

- (a) Explain why β -normal forms, as defined in the notes, are not a good normal form for $\alpha\beta\eta$ -equivalence.
- (b) Suggest a better normal form for $\alpha\beta\eta$ -equivalence.
- (c) Modify β -reduction, as defined by the notes, to obtain a different reduction system that reduces to the normal form identified in [Part 4\(b\)](#).

QUESTION 5

CONCEPTUAL · 10 MIN

● ● ● ● ● CORE

- (a) Find a context C where $C[\lambda x. x \Omega]$ has a β -normal form but $C[\lambda x. \Omega]$ does not.
- (b) Are all terms with no β -normal form contextually equivalent?
- (c) Can a term have no head normal form but have a β -normal form?
- (d) Can a term with no β -normal form have a head normal form?

QUESTION 6

CONCEPTUAL · 10 MIN (+ 10 MIN)

● ● ● ● ● CORE

This question is about **head order reduction**. Head order reduction reduces the head redex until the term is in head normal form, at which point it stops, even if the entire term is not in β -normal form. Let $\rightarrow_h$ represent one step of head order reduction.

You can use the following fact:

FACT

If a term M has a head normal form, then reducing the term with head order reduction will eventually terminate, and will transform the term into a head normal form.

- (a) Using $\lambda x. (\lambda y. y) x \Omega$ as an example, contrast head order reduction with normal-order reduction (you can write one step of normal-order reduction as $\rightarrow_n$).
- ★ (b) Assume $M \rightarrow_h M'$. Show that $M[N/x] \rightarrow_h M'[N/x]$.
- ★ (c) Show that, if M has no head normal form, then $M[N/x]$ has no head normal form either.
- (d) Assume M has no head normal form. Let M occur in head position in P

$$P = \lambda x_1 \dots x_k. M N_1 N_2 \dots N_m.$$

Does P have a head normal form? Why, or why not?

HINT You do not need to give a proof, only an informal explanation. If you want to prove this statement, proceed by induction on m . Also do [Parts 6\(b\)](#) and [6\(c\)](#), since their results are used in the proof.

2 Lambda-Definable Functions

PREREQUISITES Lecture 12 · TIME 3 hours 40 min

QUESTION 7

CONCEPTUAL · 30 MIN

● ● ● ● ● CORE

Consider the OCaml ADT for natural numbers

```
type nat = Z | S of nat
```

In this question, treat OCaml as a pure, total language: every function terminates and has no side effects. Assume also that encode and decode can be given the types

```
val encode : nat -> ('a. ('a -> 'a) -> 'a -> 'a)
val decode : ('a. ('a -> 'a) -> 'a -> 'a) -> nat
```

which cannot be written directly in OCaml.

- (a) Write the number 3 as an OCaml datatype of type nat


```
val three : nat
```
- (b) Write a fold function for nat, with signature


```
val fold : nat -> ('a -> 'a) -> 'a -> 'a
```
- (c) Let f be a function of type $'a \rightarrow 'a$ and x be some arbitrary value of $'a$. What does `fold three f x` evaluate to?
- (d) Explain what `fold` does to each `S` and `Z`.
- (e) Show there is a function `encode` such that `encode three = e`, where e is your answer to [Part 7\(c\)](#), abstracted over f and x .
- (f) Show there is a function `decode` such that `decode e = three`, where e is your answer to [Part 7\(c\)](#), abstracted over f and x .

- (g) Explain what your answers to [Part 7\(e\)](#) and [Part 7\(f\)](#) imply about the relationship between naturals and folds over naturals.
- (h) Explain the relationship between folds over naturals and the Church encoding of naturals given in the notes.

QUESTION 8

CONCEPTUAL · 10 MIN

● ● ● ● ● BASIC

- (a) Define a fold for booleans.

```
val fold : bool -> 'a -> 'a -> 'a
```

HINT A fold over type T is a general way to use / consume type T .

- (b) Using a similar procedure to [Part 7\(e\)](#) of [Question 7](#), encode true and false as functions of type $'a \rightarrow 'a \rightarrow 'a$.
- (c) Compare your answer to the encoding of booleans in the notes.

QUESTION 9

CONCEPTUAL · 20 MIN

● ● ● ● ● CORE

- (a) Write a fold (fold right) for lists of type t , where t is some concrete type.

```
type 'a list = Nil | Cons of 'a * 'a list
val fold : t list -> (t -> 'a -> 'a) -> 'a -> 'a
```

- (b) Using a similar procedure to [Part 7\(e\)](#) of [Question 7](#), encode Nil as a function of type $(t \rightarrow 'a \rightarrow 'a) \rightarrow 'a \rightarrow 'a$.
- (c) Using a similar procedure to [Part 7\(e\)](#) of [Question 7](#), and assuming xs is encoded as a function of type $(t \rightarrow 'a \rightarrow 'a) \rightarrow 'a \rightarrow 'a$, encode $\text{Cons}(x, xs)$ as a function of type $(t \rightarrow 'a \rightarrow 'a) \rightarrow 'a \rightarrow 'a$.
- (d) Derive the Church encoding of lists: Nil and Cons.

QUESTION 10

CONCEPTUAL · 20 MIN

● ● ● ● ● CORE

Let H , G , and G' be lambda terms. Assume

$$\forall A. H (G A) =_{\beta} G' (H A)$$

Prove that, for any natural n , for all lambda terms B

$$H (\underline{n} G B) =_{\beta} \underline{n} G' (H B)$$

QUESTION 11

CONCEPTUAL · 40 MIN

● ● ● ● ● CORE

- (a) Define a lambda term **Add** representing addition of Church numerals, **Add** $\underline{m} \underline{n} \equiv_{\alpha\beta\eta} \underline{m + n}$.
- (b) Define a lambda term **Times** representing multiplication of Church numerals, **Times** $\underline{m} \underline{n} \equiv_{\alpha\beta\eta} \underline{m \times n}$.
- (c) Define a lambda term **Exp** representing exponentiation of Church numerals, **Exp** $\underline{m} \underline{n} \equiv_{\alpha\beta\eta} \underline{m^n}$.

Optionally prove the correctness of your definitions.

You may take as given two important encodings, **Zero**, or $\underline{0}$, which encodes 0, and **Succ**, which encodes S .

$$\begin{aligned}\mathbf{Zero} &= \lambda f x. x \\ \mathbf{Succ} &= \lambda k f x. f (k f x)\end{aligned}$$

HINT It is easiest to first write it out as a fold on plain naturals, and then encode it piece by piece. There is a useful rule of thumb: **To encode a function defined by a fold, replace `fold n` by $\underline{n}$ and encode the other arguments.**

This is likely to not produce the simplest definition. You can use the law you proved in the previous question to simplify your initial definitions.

QUESTION 12

CONCEPTUAL · 30 MIN

● ● ● ● ● CORE

CORRECTION

Definition 32 of the notes reads as follows:

$f \in \mathbb{N}^n \rightarrow \mathbb{N}$ is λ -definable if there is a closed λ -term F that represents it: for all $(x_1, \dots, x_n) \in \mathbb{N}^n$ and $y \in \mathbb{N}$:

- if $f(x_1, \dots, x_n) = y$, then $F \underline{x_1} \cdots \underline{x_n} =_{\beta} \underline{y}$;
- if $f(x_1, \dots, x_n) \uparrow$, then $F \underline{x_1} \cdots \underline{x_n}$ has no β -normal form.

This definition is very slightly wrong: when $f(x_1, \dots, x_n)$ is undefined, we want $F \underline{x_1} \cdots \underline{x_n}$ to have no *head normal form*, not just no β -normal form.

For the following question, use the corrected definition:

$f \in \mathbb{N}^n \rightarrow \mathbb{N}$ is λ -definable if there is a closed λ -term F that represents it: for all $(x_1, \dots, x_n) \in \mathbb{N}^n$ and $y \in \mathbb{N}$:

- if $f(x_1, \dots, x_n) = y$, then $F \underline{x_1} \cdots \underline{x_n} =_{\beta} \underline{y}$;
- if $f(x_1, \dots, x_n) \uparrow$, then $F \underline{x_1} \cdots \underline{x_n}$ has no **head normal form**.

Consider the following λ -terms:

$$\mathbf{I} \triangleq \lambda x. x \quad \mathbf{C} \triangleq \lambda g f x. g (f x) \quad \mathbf{B} \triangleq \lambda g f x. f x \mathbf{I} (g (f x))$$

- (a) Show that $\underline{n} \mathbf{I} =_{\beta} \mathbf{I}$ for every $n \in \mathbb{N}$.

- (b) Assume partial functions $f, g: \mathbb{N} \rightarrow \mathbb{N}$ are represented by closed lambda terms F and G respectively. By choosing f and g , show that $\mathbf{C} G F$ does not represent $g \circ f$.
- (c) In this part, you may assume the following fact, and the result of [Part 6\(d\)](#).

FACT

If $M =_{\beta} N$, M has a head normal form if, and only if, N has a head normal form.

Assume partial functions $f, g: \mathbb{N} \rightarrow \mathbb{N}$ are represented by closed lambda terms F and G respectively. Show that their composition $(g \circ f)(x) \triangleq g(f(x))$ is represented by $\mathbf{B} G F$.

QUESTION 13

CONCEPTUAL · 40 MIN

● ● ● ● ● BASIC

- (a) Give a λ -term **Not** representing Boolean negation.
- (b) Give λ -terms **And** and **Or** representing Boolean conjunction and disjunction.
- (c) Give a λ -term **Minus** representing truncated subtraction (i.e. $\mathbf{Minus} \underline{m} \underline{n} \equiv_{\alpha\beta\eta} \underline{0}$ if $m < n$).
- (d) Give λ -terms **Eq**, **NEq**, **LT**, **LEq**, **GT**, **GEq**, representing the numeric comparison operations $=, \neq, <, \leq, >, \geq$ respectively. You can define them in any order you find most convenient.
- (e) Define the λ -term **UCr** that represents the uncurrying higher-order function: if $\langle M, N \rangle$ denotes **Pair** $M N$, then $\mathbf{UCr} F \langle M, N \rangle =_{\beta} F M N$.
- (f) Give a λ -term **MapPair** that applies a function to both elements of a pair: that is,

$$\mathbf{MapPair} F \langle M, N \rangle =_{\beta} \langle F M, F N \rangle.$$

- (g) Give a λ -term **SqSum** which represents the function $\langle m, n \rangle \mapsto m^2 + n^2$.

QUESTION 14

CONCEPTUAL · 30 MIN

● ● ● ● ● CORE

- (a) Write the `triangle_g` function below as a lambda term, **TriangleG**. You may use the following encoded terms: **If**, **n**, **Eq₀**, **Add**, **Pred**.

```
1 let triangle_g f n =
2   if n = 0 then 0
3   else n + f (n - 1)
```

OCaml

- (b) What is the relationship between `triangle`, defined below, and `triangle_g`?

```
1 let rec triangle n =
2   if n = 0 then 0
3   else n + triangle (n - 1)
```

OCaml

- (c) Write `triangle` as a lambda term, **Triangle**.
 - (d) Given a lambda term M , what does $Y\ M$ compute, where Y is the Y combinator?
 - (e) Prove that **Triangle** $\underline{3} =_{\beta} \underline{6}$.
 - (f) Can you encode, directly, the Y combinator in OCaml, without any tricks? Why, or why not?²
-

²Assume the `-rectypes` flag is off.