TD1: On the expressivity of the λ-calculus

simon.castellan@ens-lyon.fr

We recall the syntax of the λ-calculus:

$$M, N \;:=\; x \quad | \quad \lambda x.\, M \quad | \quad M\,N \qquad\qquad \text{where } x \text{ is a variable}$$

We define β-reduction on those terms

$$(\lambda x.\, M)\; N \to_\beta M[N/x]\ .$$

where $M[N/x]$ denotes the substitution of N for x inside the term M.
Terms are considered up to α-equivalence that identifies terms up to bound variables: $\lambda x.\ x =_\alpha \lambda y.y$. Recall that β-reduction generates an equivalence called β-equivalence (written $=_\beta$).

Exercise 1 (Warmup!)
Reduce the following terms using $\to_\beta$ until you cannot any more:

$$(\lambda x.\ x)\ (\lambda x.\ x) \qquad (\lambda f.\ \lambda g.\ g)\ (\lambda x.x)$$
$$(\lambda f.\ \lambda g.\ f)\ g \qquad (\lambda x.\ \lambda y.x\ y)\ (\lambda x.\ x\ (\lambda y.\ y))\ (\lambda x.\ x)$$

Decide whether the following β-equivalences hold:

$$I =_\beta I\ I \qquad (\lambda x.\ x\ x)\ I\ I =_\beta (\lambda x.\ \lambda y.\ y)\ (\lambda x.\lambda y.\ x)I$$
$$x\ (I\ I) =_\beta xI \qquad (\lambda b.\lambda x.\ \lambda y.\ b\ yx)(\lambda x.\ \lambda y.\ y) =_\beta (\lambda b.\ \lambda x.\ \lambda y.\ b\ (b\ y\ x)\ (b\ x\ y))\ (\lambda x.\ \lambda y.\ x)$$

where $I = \lambda x.\ x$.

Exercise 2 (Turing completeness)
The goal of this exercise is to give ingredients to prove that the λ-calculus is turing-complete, by showing how to encode a simple ML-like programming languages with:

- Booleans and conditionals
- Products
- Integers and basic operations
- Recursion

With these constructions, it is easy to encode turing machines inside the λ-calculus.

Question 1 (Booleans and conditionals).

1. In a ML programming language, how many "canonical terms" of type $(\alpha \to \alpha \to \alpha)$ is there?
2. Deduce two λ-terms `tt` and `ff`.

3. Define a λ-term `if` such that

$$\texttt{if tt } x\ y =_\beta x \quad \text{and} \quad \texttt{if ff } x\ y =_\beta y$$

Question 2 (Product types).

1. In ML, what is the type $\alpha \times \beta \to \gamma$ isomorphic to? Give a ML-term of "type" $\forall\gamma.(\alpha \times \beta \to \gamma) \to \alpha$
2. Deduce λ-terms π_1 and π_2. Give a term `pair` such that

$$\pi_1\ (\texttt{pair}\ a\ b) =_\beta a \quad \pi_2\ (\texttt{pair}\ a\ b) =_\beta b.$$

Question 3 (Church encodings of integers). We want to represent an integers $n \in \mathbb{N}$ by the term $\overline{n} = \lambda x.\ \lambda f.\ f^n\ x$ (*informal notation*).

1. Define λ-terms $\overline{0}$ and `succ` according to this encoding.
2. Write an iterator, i.e. a term Iter such that for all terms M, N, we have

$$\text{Iter}\ \ M\ \ N\ \overline{0} =_\beta M \qquad \text{and} \qquad \text{Iter}\ \ M\ \ N\ (\text{S}\ \ \overline{n}) =_\beta N\ (\text{Iter}\ \ M\ \ N\ \overline{n})\ .$$

3. Write terms encoding addition (`add`) and multiplication (`mult`).
4. How can we define predecessor?

Question 4 (Recursion).

1. Give a λ-term Ω that diverges, ie. such that has an infinite reduction sequence $\Omega \to t_1 \to \dots$
2. Consider the term $Y = \lambda f.(\lambda x.\ f(x\ x))(\lambda x.\ f\ (x\ x))$. Show that

$$Y\ \ f =_\beta f\ \ (Y\ \ f)$$

3. Give a λ-term `fact` such that $\texttt{fact}\ \overline{n} = \overline{n!}$. Prove that $\texttt{fact2} = 2!$.

Exercise 3 (Barendregt natural numbers)
The Barendregt natural numbers $\ulcorner n \urcorner$ are defined by:

$$\ulcorner 0 \urcorner := I \qquad\qquad \ulcorner n+1 \urcorner := \lambda k.k\ \texttt{ff}\ \ulcorner n \urcorner$$

with $\texttt{ff} = \lambda x.\ \lambda y.\ y$.

1. Implement the functions successor, predecessor, and conditional (if-zero).
2. Suggest an implementation of addition.
3. Compare the Church encoding and the Barendregt encoding of natural numbers.

Exercise 4 (Lists and trees)
We want to encode lists in λ-calculus by terms of the form $\lambda c.\lambda n.M[c, n]$. Intuitively, a list is a function with two arguments: the first is a function in the case of a non-empty list, the second is some value in the case of the empty list. For instance the list `["Bacon"; "Lettuce"; "Tomato"]` will be represented by:

$$\lambda c.\lambda n.(c\ \texttt{"Bacon"}\ (c\ \texttt{"Lettuce"}\ (c\ \texttt{"Tomato"}\ n)))\ .$$

1. Write the operators nil and cons.
2. Write an *iterator* fold such that

$$\text{fold}\ f\ u\ \text{nil} =_\beta u \qquad \text{and} \qquad \text{fold}\ f\ u\ (\text{cons}\ a\ l) =_\beta f\ a\ (\text{fold}\ f\ u\ l)\ .$$

3. Write terms for the concatenation and mirror functions.
4. Suggest an encoding for binary trees.