Proofs and Programs

Week 1, TD 1 - Pure lambda-calculus

Philippe Audebaud, Aurore Alcolei

Wednesday 14 February 2017

N.B. In the following we do not detail every questions. Please only refer to full answers to know which level of details is expected from you. Feel free to contact your teacher for any further questions.

Exercice 1 (Warmup!)**.** TD

a)
- $\mathsf{I} \equiv \lambda x.x$ is the identity term so $\mathsf{I}\,\mathsf{I} \to_\beta \mathsf{I}$
- $\mathsf{T}\,\mathsf{I} \equiv \lambda x.\lambda y.x\,\mathsf{I} \to_\beta \lambda y.\mathsf{I} \equiv \lambda y.\lambda x.x \equiv \mathsf{F}$. (note that $\mathsf{F}\,\mathsf{I} \twoheadrightarrow \mathsf{I}$)
- the β-reduction performs capture-avoiding substitution hence it is sometimes necessary to rename bounded variable in a term before substituting: $(\lambda f.\lambda g.f)\,g \equiv (\lambda f.\lambda h.f)\,g \to_\beta \lambda h.g$.
- application has left priority so

$$(\lambda x.\lambda y.xy)(\lambda x.x\,(\lambda y.y))(\lambda x.xx) \to_\beta^2 (\lambda x.x\,(\lambda y.y))(\lambda x.xx) \to_\beta (\lambda y.y)(\lambda x.xx) \to_\beta (\lambda x.xx)$$

b) true true
true false

Exercice 2 (Turing completeness)**.** Complementary remark: The following exercises lead you through an encoding of some data structures in the pure λ-calculus, in particular we define booleans, pairs and integers. However the syntax of the pure λ-calculus is not restricted so it is always possible to write something like $n + true$, that is why it may be useful to think about these encodings as terms of ML (think ocaml if you wish) to get a better idea of what are their expected behaviours.

Exercice 3 (Booleans and conditionals)**.** TD

a) In ocaml T and F would be typed by $'a \to' a \to' a$

b) $\mathbf{if}\ : bool \to' a \to' a \to' a$. We define $\mathbf{if}\ \equiv \lambda b.\lambda t.\lambda e.bte$. This really is syntactic sugar! By definition, $\mathbf{if}\ \mathsf{T}\,t\,e \to_\beta \mathsf{T}\,t\,e \to_\beta t$ since T is the left selector. Idem for $\mathbf{if}\ \mathsf{F}\,t\,e$

c) **HW** (other encodings are possible) $\mathbf{orelse} \equiv \lambda b_1, b_2.\,\mathbf{if}\ b_1\,\mathsf{T}\,b_2$, $\mathbf{not} \equiv \lambda b.b\,\mathsf{F}\,\mathsf{T}$, $\mathbf{xor} \equiv \lambda b_1, b_2.\,\mathbf{if}\ b_1(\mathbf{not}b_2)b_2$.

Exercice 4 (Pairs and projections)**.** TD Given $a, b \in \Lambda$, it is easy to pack them; for instance by building the λ-term $\lambda x.x\ a\ b$. Let us explore that path:

a) $\mathbf{pair} \equiv \lambda a.\lambda b.\lambda x.xab :' a \to' b \to ('a \to' b \to' c) \to' c$. This construction is based on the isomorphism $(A \times B) \to C \cong A \to B \to C$ (curryfication).

b) $\pi_1 \equiv \lambda p.p\,\mathsf{T}$,$\pi_2 \equiv \lambda p.p\,\mathsf{F}$ (checking the operational behaviour is left to the reader)

c) $t \equiv \lambda p.\,\mathbf{pair}(f(\pi_1 p))(\pi_1 p)$, indeed $t(\mathbf{pair}\,ab) \twoheadrightarrow \mathbf{pair}(fa)a$ since in particular $\pi_1(\mathbf{pair}\,ab) \twoheadrightarrow a$. Variable f is free in t, we can abstract it to get the closed term $\phi \equiv \lambda f.t$.

Exercice 5 (Church encodings of integers)**.** TD

a) $\mathsf{Z} \equiv \lambda f.\lambda x.x$, $\mathsf{S} \equiv \lambda n.f(nfx)$.

b) isZero? $\equiv \lambda n.n(\lambda m.\,\mathsf{F})\,\mathsf{T}$

c) $\mathbf{iter} \equiv \lambda a, b, n.nba$. By induction on n we show that $\mathbf{iter}\ a\,b\,\bar{n} =_\beta b^n a$:

- case $n = 0$: **iter** $a\, b\, \mathbf{Z} =_\beta a \equiv b^0 a$
- case $n = n'+1$: **iter** $a\, b\, (\mathbf{S}\, \bar{n'}) =_\beta b(\bar{n'}ab)$, $(\bar{n'}ab) =_\beta$ **iter** $a\, b\, \bar{n'}$ and by induction **iter** $a\, b\, \bar{n'} =_\beta b^{n'} a$ hence **iter** $a\, b\, (\mathbf{S}\, \bar{n'}) =_\beta b^n a$.

d) **iter** is syntactic sugar and the proof does not hold with the use of $\twoheadrightarrow$.

e) **HW add** $\equiv \lambda n_1, n_2.$ **iter** $n_2\, \mathbf{S}\ n_1$. **mult** $\equiv \lambda n_1, n_2.$ **iter** $\mathbf{Z}$ (**add** n_2) n_1

f) **HW pred** $\equiv \lambda n.\pi_2$ (**iter** (**pair** $\mathbf{Z}\ \mathbf{Z}$) $(\phi\, \mathbf{S})$ n).

Exercice 6 (Recursion)**.** TD

a) $\mathbf{\Upsilon}$ is called a *fixed point combinator* this means that given a function f, $\mathbf{\Upsilon}\, f$ is a fixed point for this function id est, $\mathbf{\Upsilon}\, m =_\beta m\, (\mathbf{\Upsilon}\, m)$. The relation does not holds with $\twoheadrightarrow$.

b) Let **fact_aux** $\equiv \lambda f.\lambda n.$ **if** (isZero? n) $(\mathbf{S}\, \mathbf{Z})$ (**mult** n (f (**pred** n)))) then **fact** $\equiv \mathbf{\Upsilon}$ **fact_aux**. (here you can think of $\mathbf{\Upsilon}$ as the let rec in ocaml).

$$\begin{aligned}
\mathbf{fact}\, 2 &=_\beta \mathbf{fact_aux}\ \ \mathbf{fact}\, 2 \\
&=_\beta \mathbf{mult}\ 2\ (\mathbf{fact}\, \mathbf{pred}\ 2) \\
&=_\beta \mathbf{mult}\ 2\ (\mathbf{fact}\, 1) \\
&=_\beta \mathbf{mult}\ 2\ (\mathbf{fact_aux}\, \mathbf{fact}\, 1) \\
&=_\beta \mathbf{mult}\ 2\ (\mathbf{mult}\ 1\ (\mathbf{fact}\, 0)) \\
&=_\beta \mathbf{mult}\ 2\ (\mathbf{mult}\ 1\ (\mathbf{fact_aux}\, \mathbf{fact}\, 0)) \\
&=_\beta \mathbf{mult}\ 2\ (\mathbf{mult}\ 1\ 1) \\
&=_\beta 2
\end{aligned}$$

c) **HW** immediate.

Exercice 7 (Barendregt natural numbers)**. HW**

a) **successor** $\equiv \lambda n.$**pair** $\mathbf{F}\, n$, **predecessor** $\equiv \lambda n.\pi_2\ n$, **test-to-zero** $\equiv \lambda n.n\ \ \mathbf{T}$.

b) **addition** $\equiv \mathbf{\Upsilon}(\lambda \text{add\_aux}.\lambda n_1, n_2.$ **if** (**test-to-zero** n_1) n_2 (add_aux (**predecessor** n_1) (**successor** n_2)))

c) Barendregt natural numbers are easier to compute with but compare to Church encoding their structure does not allow iteration, one need to use a fixed point combinator.