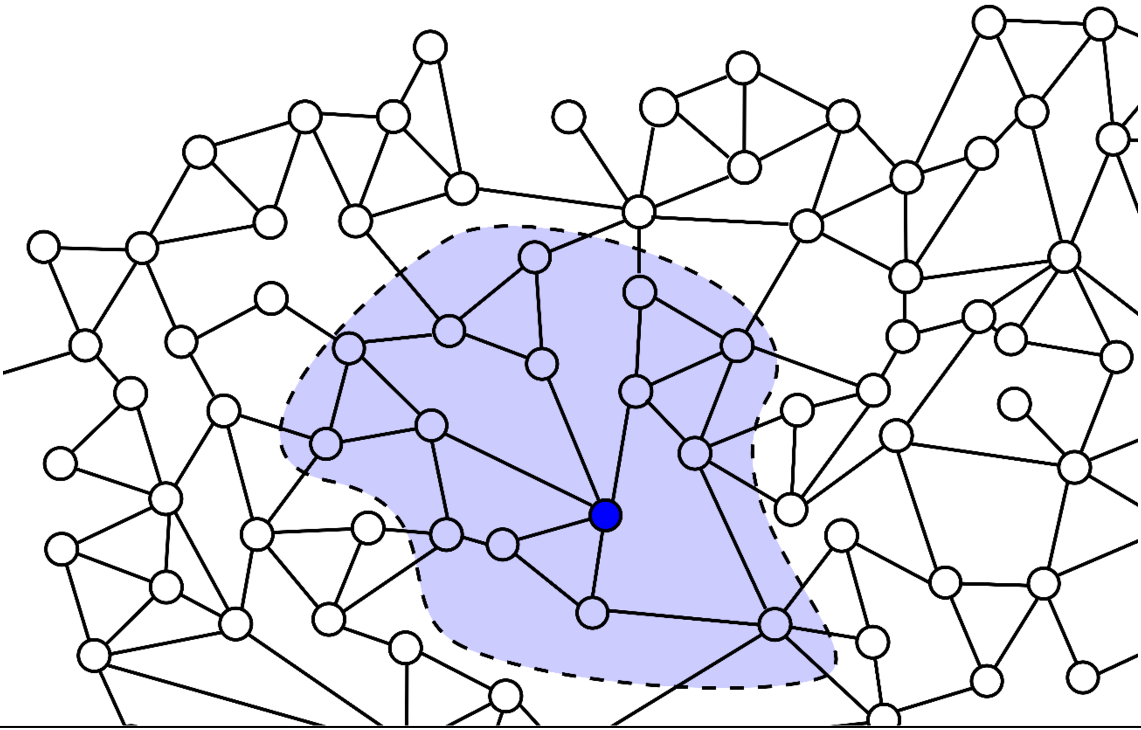
Homework

Lambda Calculus Exercises III

24th November 2025



Teacher: Christine Paulin

Herrá Alpuente, Luis Miguel: luis-miguel.herra-alpuente@universite-paris-saclay.fr

1. Prove that there exists no λ-term h such that h [t] = T for any t ∈ Λ with a normal form and h [t] = F for any t ∈ Λ with no normal form.

Suppose for contradiction that such a λ-term h exists. Define the sets:

A = {t ∈ Λ | t has a normal form} and B = {t ∈ Λ | t has no normal form}.

We verify that A and B satisfy the conditions of Scott's theorem. First, A is non-empty since λx.x ∈ A, and B is non-empty since **Ω ≡ (λx.xx)(λx.xx)** ∈ B. Second, both A and B are closed under β-equality**: if t =β u, then t has a normal form if and only if u has a normal form** by the Church-Rosser theorem.

If h exists as claimed, then h effectively characterizes A, meaning **h [t] =β T** when t ∈ A and h [t] =β F when t ∉ A. However, <u>this contradicts Scott's undecidability theorem, which states that no non-trivial set closed by β-equality can be effectively characterized</u>. Therefore, no such h can exist.

2. Using the encoding of algebraic datatypes, and one of the already defined encodings of numbers, propose an encoding of lists, and of the nth function.

Nil ≡ λc n. n

Cons(h, t) ≡ λc n. c h t

The nth function: We define nth recursively using the Θ combinator:

nth ≡ Θ (λf k ℓ. ℓ (λh t. if isZ k then h else f (P k) t) F)

Expanding this definition: nth ≡ Θ (λf k ℓ. ℓ (λh t. isZ k h (f (P k) t)) F)

When applied to an index k and a list ℓ, the term ℓ performs pattern matching. If ℓ is Nil, it returns F (indicating failure). If ℓ is Cons(h, t), it checks whether k is zero. If so, it returns h. Otherwise, it recursively calls f with P k and t; evaluating every suggested case scenario.

3. In your encoding, prove that nth k ℓ = nth (k + 1) (t : ℓ).

Proof by induction on ℓ:

Base case (ℓ = Nil):

<u>nth [k] Nil</u>

≡ Θ g [k] Nil where g = λf k ℓ. ℓ (λh t. if isZ k then h else f (P k) t) F

→* Nil (λh t. if isZ [k] then h else ...) F

≡ (λc n. n) (λh t. if isZ [k] then h else ...) F

→β F

<u>nth [k+1]</u> (Cons(t, Nil))

≡ Θ g [k+1] (Cons(t, Nil))

→* (Cons(t, Nil)) (λh t'. if isZ [k+1] then h else Θ g (P [k+1]) t') F

≡ (λc n. c t Nil) (λh t'. if isZ [k+1] then h else Θ g (P [k+1]) t') F

→β (λh t'. if isZ [k+1] then h else Θ g (P [k+1]) t') t Nil

→β if isZ [k+1] then t else Θ g (P [k+1]) Nil

≡ if F then t else Θ g [k] Nil (since P [k+1] =β [k] and isZ [k+1] =β F)
→β Θ g [k] Nil

≡ nth [k] Nil

→* F

Inductive case (ℓ = Cons(h, tail)):

Assume as induction hypothesis that nth [k] tail =β nth [k+1] (Cons(t, tail)).

nth [k] (Cons(h, tail)) ≡

Θ g [k] (Cons(h, tail))

→* (Cons(h, tail)) (λh' t'. if isZ [k] then h' else Θ g (P [k]) t') F

≡ (λc n. c h tail) (λh' t'. if isZ [k] then h' else Θ g (P [k]) t') F

→β (λh' t'. if isZ [k] then h' else Θ g (P [k]) t') h tail

→β if isZ [k] then h else Θ g (P [k]) tail

nth [k+1] (Cons(t, Cons(h, tail)))

≡ Θ g [k+1] (Cons(t, Cons(h, tail)))

→* (Cons(t, Cons(h, tail))) (λh' t'. if isZ [k+1] then h' else Θ g (P [k+1]) t') F

≡ (λc n. c t (Cons(h, tail))) (λh' t'. if isZ [k+1] then h' else Θ g (P [k+1]) t') F

→β (λh' t'. if isZ [k+1] then h' else Θ g (P [k+1]) t') t (Cons(h, tail))

→β if isZ [k+1] then t else Θ g (P [k+1]) (Cons(h, tail))

≡ if F then t else Θ g [k] (Cons(h, tail)) (since isZ [k+1] =β F and P [k+1] =β [k])

→β Θ g [k] (Cons(h, tail))

≡ nth [k] (Cons(h, tail))

Therefore, both expressions β-reduce to the same term, proving:

nth [k] ℓ =β nth [k+1] (Cons(t, ℓ)).