Examen final MCAL lambda-calcul

1h30, documents non autorisés

Pour ne pas alourdir, on ne manipulera que des λ-termes non typés, sauf mention explicite du contraire (par exemple pour des codages utilisant les notations Coq).

Le sujet comporte trois parties. La première est composée de questions élémentaires. La seconde porte sur la représentation de structures de données finies obtenues en composant des types énumérés au moyen de sommes et de produits de types. La troisième a pour but de calculer le logarithme en base 2 d'un entier.

Toutes les réponses seront reportés sur la feuille séparée comportant un tableau QCM par partie. Il est important de ***ne pas*** *répondre au hasard : les réponses fausses seront comptées négativement. Bien raisonner ou effectuer les calculs au brouillon avant de cocher votre réponse. Pour chaque question il y aura au moins une réponse correcte proposée. Lorsque plusieurs réponses correctes sont possibles, leur nombre sera indiqué et il conviendra de cocher toutes pour avoir le maximum de points à cette question.*

Le barème est indicatif.

Dans toute la suite, U, V, W, éventuellement avec des indices, représentent des λ-termes. On introduira des abréviations sous la forme $\mathtt{aa} \stackrel{\text{def}}{=} \mathrm{U}$, par exemple $\mathtt{I} \stackrel{\text{def}}{=} \lambda x.x$.

Partie 1

Voir tableau QCM. On rappelle que si U, V, W sont des λ-termes, alors U V W est un λ-terme qui est une abréviation de (U V) W. On rappelle également qu'un *rédex* est une position dans un λ-terme donnant lieu à β-réduction.

Partie 2 : sommes et produits

On rappelle que l'on peut représenter les booléens (type énuméré à 2 valeurs) au moyen des λ-termes $\mathtt{vr} \stackrel{\text{def}}{=} \lambda x y.\, x$ et $\mathtt{fa} \stackrel{\text{def}}{=} \lambda x y.\, y$. Si B est l'un de ces λ-termes, ou si B se réduit en l'un de ces λ-termes, B U V se réduit donc (en 2 étapes) soit en U soit en V.

De même on va représenter un type énuméré à 3 valeurs au moyen des λ-termes $\mathtt{e}_{3_0} \stackrel{\text{def}}{=} \lambda x_0 x_1 x_2.\, x_0$, $\mathtt{e}_{3_1} \stackrel{\text{def}}{=} \lambda x_0 x_1 x_2.\, x_1$ et $\mathtt{e}_{3_2} \stackrel{\text{def}}{=} \lambda x_0 x_1 x_2.\, x_2$ et un type énuméré à 4 valeurs au moyen des λ-termes $\mathtt{e}_{4_0} \stackrel{\text{def}}{=} \lambda x_0 x_1 x_2 x_3.\, x_0$, $\mathtt{e}_{4_1} \stackrel{\text{def}}{=} \lambda x_0 x_1 x_2 x_3.\, x_1$, $\mathtt{e}_{4_2} \stackrel{\text{def}}{=} \lambda x_0 x_1 x_2 x_3.\, x_2$ et $\mathtt{e}_{4_3} \stackrel{\text{def}}{=} \lambda x_0 x_1 x_2 x_3.\, x_3$.

On rappelle que la somme A+B de deux types A et B est définie en λ-calcul typé polymorphe par $\forall \mathrm{T}, (\mathrm{A} \to \mathrm{T}) \to (\mathrm{B} \to \mathrm{T}) \to \mathrm{T}$. On construit donc des λ-termes de type A + B soit à partir de λ-termes U de type A : $\Lambda \mathrm{T}.\lambda k_1^{\mathrm{A}\to\mathrm{T}} k_2^{\mathrm{B}\to\mathrm{T}}.k_1\, \mathrm{U}$, soit à partir de λ-termes V de type B : $\Lambda \mathrm{T}.\lambda k_1^{\mathrm{A}\to\mathrm{T}} k_2^{\mathrm{B}\to\mathrm{T}}.k_2\, \mathrm{V}$. Les versions non typées sont $\lambda k_1 k_2.k_1\, \mathrm{U}$ et $\lambda k_1 k_2.k_2\, \mathrm{V}$. Par exemple on peut rassembler des valeurs des types énumérés précédents à 2 ou 3 valeurs en formant
$\mathtt{bpt}_0 \stackrel{\text{def}}{=} \lambda k_1 k_2.k_1\, \mathtt{vr}$, $\mathtt{bpt}_1 \stackrel{\text{def}}{=} \lambda k_1 k_2.k_1\, \mathtt{fa}$, $\mathtt{bpt}_2 \stackrel{\text{def}}{=} \lambda k_1 k_2.k_2\, \mathtt{e}_{3_0}$, $\mathtt{bpt}_3 \stackrel{\text{def}}{=} \lambda k_1 k_2.k_2\, \mathtt{e}_{3_1}$ et $\mathtt{bpt}_4 \stackrel{\text{def}}{=} \lambda k_1 k_2.k_2\, \mathtt{e}_{3_2}$.

On rappelle que le produit A × B de deux types A et B est défini en λ-calcul typé polymorphe par $\forall \mathrm{T}, (\mathrm{A} \to \mathrm{B} \to \mathrm{T}) \to \mathrm{T}$. On construit donc des λ-termes de type A × B soit à partir de λ-termes U et V respectivement de type A et B par :

$\Lambda\mathrm{T}.\lambda k^{\mathrm{A}\to\mathrm{B}\to\mathrm{T}}.k\,\mathrm{U}\,\mathrm{V}$. La version non typée est $\lambda k.k\,\mathrm{U}\,\mathrm{V}$. Par exemple on peut fabriquer les couples de booléens, ce qui donne une autre type fini à 4 valeurs :
$\mathtt{bfb}_0 \stackrel{\text{def}}{=} \lambda k.k\,\mathtt{vr}\,\mathtt{vr}$, $\mathtt{bfb}_1 \stackrel{\text{def}}{=} \lambda k.k\,\mathtt{vr}\,\mathtt{fa}$, $\mathtt{bfb}_2 \stackrel{\text{def}}{=} \lambda k.k\,\mathtt{fa}\,\mathtt{vr}$ et $\mathtt{bfb}_3 \stackrel{\text{def}}{=} \lambda k.k\,\mathtt{fa}\,\mathtt{fa}$.
Une autre façon de fabriquer un type fini à 4 valeurs est d'utiliser la somme de types sur les booléens :
$\mathtt{bpb}_0 \stackrel{\text{def}}{=} \lambda k_1 k_2.k_1\,\mathtt{vr}$, $\mathtt{bpb}_1 \stackrel{\text{def}}{=} \lambda k_1 k_2.k_1\,\mathtt{fa}$, $\mathtt{bpb}_2 \stackrel{\text{def}}{=} \lambda k_1 k_2.k_2\,\mathtt{vr}$ et $\mathtt{bpb}_3 \stackrel{\text{def}}{=} \lambda k_1 k_2.k_2\,\mathtt{fa}$.
La dernière question de cette partie consiste à coder la fonction qui envoie $\mathtt{bfb}_0$, $\mathtt{bfb}_1$, $\mathtt{bfb}_2$ et $\mathtt{bfb}_3$ respectivement vers $\mathtt{bpb}_0$, $\mathtt{bpb}_1$, $\mathtt{bpb}_2$ et $\mathtt{bpb}_3$.

Partie 3 : logarithme en base 2

En généralisant ce qui précède on peut faire correspondre bijectivement $\mathtt{pbool} \times \mathrm{A}$ et $\mathrm{A} + \mathrm{A}$, pour n'importe quel type A, même si A contient un nombre infini d'éléments (par exemple `pnat`, le type polymorphe des entiers de Church).
Pour calculer le logarithme en base 2 d'un entier de Church plus grand que (le codage de) 1, une possibilité est de le diviser par 2 et de recommencer jusqu'à ce que l'on obtienne 1 : le nombre de divisions par 2 nécessaires donne le résultat recherché.
Pour diviser un entier de Church n par 2, on va itérer n fois, en partant de 0, une opération qui ajoute 1 une fois sur deux. Pour cela on peut travailler sur un type `nn` qui est, à votre choix, $\mathtt{pbool} \times \mathtt{pnat}$ ou bien $\mathtt{pnat} + \mathtt{pnat}$. Par exemple, en travaillant avec ce dernier type, on a des « entiers de gauche » et des « entiers de droite » : pour le voir, on considère que $\mathtt{pnat} + \mathtt{pnat}$ est $\mathrm{A} + \mathrm{B}$ avec $\mathrm{A} = \mathtt{pnat}$ et $\mathrm{B} = \mathtt{pnat}$; on forme une valeur de ce type à partir d'un entier n de type `pnat` en faisant jouer au type de n soit le rôle de A (à gauche), soit le rôle de B (à droite). L'opération envoie un entier « de gauche » n vers le même entier mais « de droite », et un entier « de droite » n vers son successeur « de gauche » $\mathtt{pS}\,n$.
Pour cette partie on suppose connus le codage de l'entier 0, noté `p0`, le codage de l'opération successeur, notée `pS`, et le codage du test à 0 d'un entier de Church, noté `tzer`. (Pour simplifier on calcule en réalité $1 + \log_2 n$ au lieu de $\log_2 n$, puisqu'on s'arrête à 0 et non à 1).

On rappelle aussi qu'une fonction récursive f peut se coder en appliquant un combinateur de point fixe à la fonctionnelle associée à f. On admettra que l'on dispose d'un tel combinateur de point fixe, noté `Y`.

Cette partie n'est pas à rendre sous forme de QCM, mais vos réponses seront données sur l'espace laissé libre entre chaque question page 4.

PAGE DE RÉPONSES

NUMÉRO ÉTUDIANT :

Partie 1 (8 pts)

Questions	Réponses
1. Dans les codages vus en cours, le λ-terme $\lambda xy.\,y$ peut représenter (2 bonnes réponses)	❐ le booléen **vrai** ☑ le booléen **faux** ❐ l'entier 1 ☑ l'entier 0
2. Le λ-calcul pur est un modèle de calcul basé sur un unique mécanisme, qui représente	❐ l'appel d'une fonction récursive ❐ l'affection d'une valeur à une variable en mémoire ☑ un passage de paramètre comme dans un appel de fonctions ❐ l'addition des booléens
3. En λ-calcul pur non typé, on peut additionner des booléens (on peut appliquer le λ-terme qui code l'addition à des λ-termes qui codent **vrai** ou **faux**)	☑ oui, et le λ-terme obtenu peut se réduire ❐ oui, mais le λ-terme obtenu n'a aucun rédex ❐ non, parce que cela n'a pas de sens ❐ non, car c'est syntaxiquement incorrect
4. Le λ-terme $(\lambda x.x)\,((\lambda x.x)\,(\lambda x.x))$	❐ ne comporte aucun rédex ❐ comporte exactement un rédex ☑ comporte exactement deux rédexes ❐ comporte exactement trois rédexes
5. Le λ-terme $(\lambda x.x)\,((\lambda x.x)\,(\lambda x.x))$ se réduit en une étape de β-réduction en	❐ $\lambda x.xx$ ☑ $(\lambda x.x)\,(\lambda x.x)$ ❐ $(\lambda x.x)\,(\lambda x.xx)$ ou en $(\lambda x.xx)\,(\lambda x.x)$ ❐ $\lambda x.x$
6. Le λ-terme $(\lambda x.x)\,(\lambda x.x)\,(\lambda x.x)$	❐ ne comporte aucun rédex ☑ comporte exactement un rédex ❐ comporte exactement deux rédexes ❐ comporte exactement trois rédexes
7. Le λ-terme $(\lambda x.x)\,(\lambda x.x)\,(\lambda x.x)$ se réduit en une étape de β-réduction en	❐ $\lambda x.xx$ ☑ $(\lambda x.x)\,(\lambda x.x)$ ❐ $(\lambda x.x)\,(\lambda x.xx)$ ou en $(\lambda x.xx)\,(\lambda x.x)$ ❐ $\lambda x.x$

Partie 2 : sommes et produits (10 pts)

Questions	Réponses
1. Le λ-terme $\lambda e.\, e\ \texttt{vr}\ \texttt{fa}\ \texttt{vr}$ code la fonction qui envoie les valeurs $\texttt{e3}_0$, $\texttt{e3}_1$ et $\texttt{e3}_2$ respectivement vers	❐ $\texttt{e3}_0$, $\texttt{e3}_1$ et $\texttt{e3}_2$ ☑ vr, fa et vr ❐ $\texttt{e3}_0$, $\texttt{e3}_1$ et $\texttt{e3}_0$
2. Pour coder la permutation circulaire qui envoie les valeurs $\texttt{e3}_0$, $\texttt{e3}_1$ et $\texttt{e3}_2$ respectivement vers $\texttt{e3}_1$, $\texttt{e3}_2$ et $\texttt{e3}_0$, on peut employer (2 bonnes réponses)	☑ $\lambda e.\, e\ \texttt{e3}_1\ \texttt{e3}_2\ \texttt{e3}_0$ ❐ $(\lambda x.y)\,(\lambda y.z)\,(\lambda z.x)$ ❐ $\lambda e.\, e\,(\lambda x.y)\,(\lambda y.z)\,(\lambda z.x)$ ☑ $\lambda e.\lambda x\,y\,z.\, e\,y\,z\,x$
3. Pour coder la permutation circulaire qui envoie les valeurs $\texttt{bpt}_0$, $\texttt{bpt}_1$, $\texttt{bpt}_2$, $\texttt{bpt}_3$ et $\texttt{bpt}_4$ respectivement vers $\texttt{bpt}_1$, $\texttt{bpt}_2$ $\texttt{bpt}_3$, $\texttt{bpt}_4$ et $\texttt{bpt}_0$, on peut employer (1 bonne réponse)	❐ $\lambda s.\, s\ \texttt{bpt}_1\ \texttt{bpt}_2\ \texttt{bpt}_3\ \texttt{bpt}_4\ \texttt{bpt}_0$ ☑ $\lambda s.\, s\,(\lambda x.\, x\ \texttt{bpt}_1\ \texttt{bpt}_2)(\lambda y.\, y\ \texttt{bpt}_3\ \texttt{bpt}_4\ \texttt{bpt}_0)$ ❐ $(\lambda x.y)\,(\lambda y.z)\,(\lambda z.t)\,(\lambda t.u)\,(\lambda u.x)$ ❐ $\lambda s.\, s\ \texttt{bpt}_2\ \texttt{bpt}_3\ \texttt{bpt}_4\ \texttt{bpt}_0\ \texttt{bpt}_1$
4. Pour coder la fonction qui envoie $\texttt{bfb}_0$, $\texttt{bfb}_1$, $\texttt{bfb}_2$ et $\texttt{bfb}_3$ respectivement vers $\texttt{bpb}_0$, $\texttt{bpb}_1$, $\texttt{bpb}_2$ et $\texttt{bpb}_3$ on peut employer (2 bonnes réponses)	☑ $\lambda p.\lambda k_1\, k_2.\, p(\lambda b.b\, k_1\, k_2)$ ☑ $\lambda p.\, p(\lambda ab.\, a\,(b\ \texttt{bpb}_0\ \texttt{bpb}_1)\,(b\ \texttt{bpb}_2\ \texttt{bpb}_3))$ ❐ $\lambda p.\, p\ \texttt{bpb}_0\ \texttt{bpb}_1\ \texttt{bpb}_2\ \texttt{bpb}_3$ ❐ $\lambda k_1\, k_2.\,(k_1 \texttt{bpb}_0\ \texttt{bpb}_1)\ k_2\,(\texttt{bpb}_2\ \texttt{bpb}_3)$

Partie 3 : logarithme en base 2 (8 pts)

Indiquer (en l'entourant) le type choisi nn : pbool × pnat ou pnat + pnat.

3.1

Donner le λ-terme cont qui extrait l'entier de Church contenu dans une valeur de type nn.

$\texttt{cont} \stackrel{\text{def}}{=} \lambda x.\, x\,(\lambda b\, n.\, n)$ $\qquad$ $\texttt{cont} \stackrel{\text{def}}{=} \lambda x.\, x\,(\lambda n.n)\,(\lambda n.n)$

3.2

Donner le λ-terme usd qui code l'incrémentation une fois sur deux de l'entier contenu dans une valeur de type nn.

$\texttt{usd} \stackrel{\text{def}}{=} \lambda x\, k.\, x\,(\lambda b\, n.\, b\,(k\ \texttt{fa}\ n)\,(k\ \texttt{vr}\ (\text{pS}\ n)))$ $\qquad$ $\texttt{usd} \stackrel{\text{def}}{=} \lambda x k_1\, k_2.\, x\,(\lambda n. k_2\, n)\,(\lambda n. k_1\,(\text{pS}\ n))$

3.3

En combinant usd et cont (ou en les supposant connus) donner le λ-terme half qui rend la division euclidienne d'un entier de Church par 2.

$\texttt{half} \stackrel{\text{def}}{=} \lambda n.\, \texttt{cont}\,(n\ \texttt{usd}\,(\lambda k.\, k\ \texttt{vr}\ \text{p0}))$ $\qquad$ $\texttt{half} \stackrel{\text{def}}{=} \lambda n.\, \texttt{cont}\,(n\,\texttt{usd}\,(\lambda k_1\, k_2.\, k_1\ \text{p0}))$

3.4

Donner la fonctionnelle flog2 associée à la fonction log2 qui appelle récursivement half tant que son argument est plus grand que 0. Coder ensuite la fonction log2 au moyen de Y.

$\texttt{flog2} \stackrel{\text{def}}{=} \lambda f\, n.\, \texttt{tzer}\ n\ \text{p0}\ (\text{pS}\ (f\ (\texttt{half}\ n))$ $\qquad$ $\texttt{log2} \stackrel{\text{def}}{=} \texttt{Y}\ \texttt{flog2}$