

Examen, λ -calcul, 2023

7 juin 2023

Notes préliminaires. Comme d'habitude, vos réponses devront être claires. Toute affirmation doit être justifiée. Aucune référence en avant n'est autorisée. Toute réponse à une question doit être lisible indépendamment des autres. N'inventez pas vos propres notations, et utilisez celles de l'énoncé.

1 Types rékursifs

OCaml dispose de types *rékursifs*, c'est-à-dire définis en termes d'eux-mêmes. On va considérer une discipline de types pour le λ -calcul qui permet de définir des types rékursivement, comme OCaml.

Les types sont donnés par l'extension suivante des types simples :

$$F ::= \alpha \mid F \Rightarrow G \mid \mu\alpha.F$$

où les α sont des variables de type, et où la variable de type α est liée dans $\mu\alpha.F$. On supposera que l' α -renommage est appliqué aux types implicitement.

On considère un λ -calcul étendu par deux opérations **fold** et **unfold**, que l'on appellera le **λ fold-calcul**.

La syntaxe du **λ fold-calcul** est (modulo α -renommage lui aussi) :

$s, t, u, v, \dots ::= x, y, z, \dots$	variables
uv	applications
$\lambda x.u$	λ -abstractions
fold u	repliage
unfold u	dépliage.

Les règles de réduction sont :

$$\begin{aligned} (\beta) \quad & (\lambda x.u)v \rightarrow u[x := v] \\ (\text{fold}) \quad & \text{unfold}(\text{fold } u) \rightarrow u. \end{aligned}$$

Elles s'appliquent sous tout contexte.

Les règles de typage sont :

$$\frac{}{\Gamma, x : F \vdash x : F} (Var)$$

$$\frac{\Gamma \vdash u : F_1 \Rightarrow F_2 \quad \Gamma \vdash v : F_1}{\Gamma \vdash uv : F_2} (App) \qquad \frac{\Gamma, x : F_1 \vdash u : F_2}{\Gamma \vdash \lambda x \cdot u : F_1 \Rightarrow F_2} (Abs)$$

$$\frac{\Gamma \vdash u : F[\alpha := \mu\alpha.F]}{\Gamma \vdash \mathbf{fold} \ u : \mu\alpha.F} (Fold) \qquad \frac{\Gamma \vdash u : \mu\alpha.F}{\Gamma \vdash \mathbf{unfold} \ u : F[\alpha := \mu\alpha.F]} (Unfold)$$

Ces règles de typage forment le *système* μ . On admettra que le $\lambda\mathbf{fold}$ -calcul a la propriété d'auto-réduction pour le système μ .

Le *type paradoxal* est $\pi \stackrel{\text{def}}{=} \mu\alpha.(\alpha \Rightarrow \alpha)$. Un contexte de typage est *paradoxal* s'il est de la forme $x_1 : \pi, \dots, x_n : \pi$; son *domaine* est $\{x_1, \dots, x_n\}$.

Question 1 Définir une fonction φ de traduction des λ -termes ordinaires (sans \mathbf{fold} ni \mathbf{unfold} , et non typés) vers les $\lambda\mathbf{fold}$ -termes, ayant les propriétés suivantes :

- (a) pour tout λ -terme u , $\varphi(u)$ a les mêmes variables libres que u ;
- (b) pour tout λ -terme u , pour tout contexte de typage paradoxal Γ dont le domaine contient les variables libres de u , on peut typer $\Gamma \vdash \varphi(u) : \pi$ en système μ (oui, même sans supposer u typé, dans aucune discipline de types);
- (c) pour tous λ -termes u et u' tels que $u \rightarrow u'$, on a $\varphi(u) \rightarrow^+ \varphi(u')$.

Je n'ai pas besoin de preuve des propriétés (a), (b) et (c) : je saurai si vous avez compris en voyant votre définition de φ .

Question 2 En déduire qu'il existe des $\lambda\mathbf{fold}$ -termes qui, bien que typables, ne sont pas fortement normalisables, et en fait pas normalisables du tout.

On répare ce problème en restreignant la construction de point fixe $\mu\alpha.F$ de sorte que α n'apparaisse que *positivement* dans F . (Dans le type paradoxal π , α apparaît *négativement* dans $\alpha \Rightarrow \alpha$, via son occurrence à gauche de \Rightarrow .) Les ensembles de variables *apparaissant positivement* $pos(F)$ et *apparaissant négativement* $neg(F)$ dans un type F sont définis par :

$$\begin{aligned} pos(\alpha) &\stackrel{\text{def}}{=} \{\alpha\} & neg(\alpha) &\stackrel{\text{def}}{=} \emptyset \\ pos(F \Rightarrow G) &\stackrel{\text{def}}{=} neg(F) \cup pos(G) & neg(F \Rightarrow G) &\stackrel{\text{def}}{=} pos(F) \cup neg(G) \\ pos(\mu\alpha.F) &\stackrel{\text{def}}{=} pos(F) \setminus \{\alpha\} & neg(\mu\alpha.F) &\stackrel{\text{def}}{=} neg(F) \setminus \{\alpha\}. \end{aligned}$$

On n'aura en fait pas besoin de la définition explicite ; seule une propriété de monotonie, que l'on admettra plus loin, sera utile.

Le système μ^+ est la restriction du système μ où, dans tout type $\mu\alpha.F$ (ou sous-expression $\mu\alpha.F$ d'un type), on a $\alpha \notin \text{neg}(F)$.

Dans la suite, nous aurons besoin du théorème de Tarski. Au cas où vous ne le connaîtriez pas déjà, le voici. Un *treillis complet* est un ensemble ordonné (A, \leq) dans lequel toute famille $F \subseteq A$ admet une borne inférieure (la borne inférieure $\inf F$ de F , si elle existe, est le plus grand de ses minorants ; un minorant de F est un élément $a \in A$ tel que $a \leq x$ pour tout $x \in F$). Une fonction $f: A \rightarrow A$ est *monotone* si et seulement si elle préserve l'ordre : $a \leq a'$ implique $f(a) \leq f(a')$. Un point fixe de f est un élément $a \in A$ tel que $f(a) = a$. Un plus petit point fixe de f est un point fixe de f qui est inférieur ou égal à tout point fixe de f . Alors :

Theorem 1 (Tarski) *Pour tout treillis complet (A, \leq) , toute fonction monotone f de A dans A admet un plus petit point fixe $\text{lfp}(f)$.*

On définit les candidats de réductibilité pour le λfold -calcul exactement comme pour le λ -calcul, à l'aide des propriétés **(CR1)**, **(CR2)**, **(CR3)**, à la différence près qu'un terme *neutre* désigne maintenant un terme ne commençant ni par λ ni par fold . SN est l'ensemble des λfold -termes fortement normalisables. Explicitement, un *candidat de réductibilité* est un ensemble S de λfold -termes tel que :

(CR1) $S \subseteq SN$;

(CR2) pour tout $u \in S$, si $u \rightarrow u'$ alors $u' \in S$;

(CR3) pour tout λfold -terme neutre u , si tous les réduits en une étape u' de u sont dans S , alors u est lui-même dans S .

On admettra :

Lemme 2 *SN est un candidat de réductibilité.*

On notera CR l'ensemble de tous les candidats de réductibilité pour notre nouveau calcul, ordonné par l'ordre d'inclusion \subseteq .

Question 3 Montrer que (CR, \subseteq) est un treillis complet. (Attention au cas de la famille vide de candidats, dont on donnera explicitement la borne inférieure.)

On définit :

$$S \Rightarrow S' \stackrel{\text{def}}{=} \{u \text{ } \lambda\text{fold}\text{-terme} \mid \forall v \in S, uv \in S'\}.$$

Question 4 Montrer que pour tous candidats de réductibilité S et S' , $S \Rightarrow S'$ vérifie **(CR3)**.

Il est facile de voir que $S \Rightarrow S'$ vérifie aussi **(CR1)** et **(CR2)**, d'où le résultat suivant, similaire au cas du λ -calcul, et que l'on admettra.

Lemme 3 Pour tous candidats de réductibilité S et S' , $S \Rightarrow S'$ est un candidat de réductibilité.

On définit aussi :

$$\text{Fold}(S) \stackrel{\text{def}}{=} \{u \text{ } \lambda\text{fold-terme} \mid \mathbf{unfold} \ u \in S\}.$$

Question 5 Montrer que pour tout candidat de réductibilité S , $\text{Fold}(S)$ est un candidat de réductibilité.

Nous avons tout ce qu'il nous faut pour entamer notre démonstration. Un *contexte de candidats* est une fonction C qui à chaque variable de type α associe un candidat de réductibilité. Pour tout type F du système μ^+ , pour tout contexte de candidats C , on définit RED_F^C par récurrence sur F comme suit :

$$\begin{aligned} RED_\alpha^C &\stackrel{\text{def}}{=} C(\alpha) \\ RED_{F \Rightarrow G}^C &\stackrel{\text{def}}{=} RED_F^C \Rightarrow RED_G^C \\ RED_{\mu\alpha.F}^C &\stackrel{\text{def}}{=} \text{lfp}(S \in CR \mapsto \text{Fold}(RED_F^{C[\alpha:=S]})). \end{aligned}$$

La notation $S \in CR \mapsto \text{Fold}(RED_F^{C[\alpha:=S]})$ désigne la fonction qui à tout S dans CR associe $\text{Fold}(RED_F^{C[\alpha:=S]})$.

Il se trouve que la fonction $S \in CR \mapsto \text{Fold}(RED_F^{C[\alpha:=S]})$ est monotone si $\alpha \notin \text{neg}(F)$. La définition de $RED_{\mu\alpha.F}^C$ a donc un sens, grâce au théorème de Tarski, pour les types du système μ^+ . Plus généralement, les résultats énoncés jusqu'ici permettent de démontrer le résultat suivant, que l'on admettra.

Lemme 4 Pour tout type F du système μ^+ , pour tout contexte de candidats C , RED_F^C est un candidat de réductibilité.

On admettra aussi :

Lemme 5 Pour tous types F et G du système μ^+ , pour toute variable de type α , pour tout contexte de candidats C , $RED_{F[\alpha:=G]}^C = RED_F^{C[\alpha \mapsto RED_G^C]}$.

Lemme 6 Pour tous types F et G du système μ^+ , pour tout contexte de candidats C , pour tout $\lambda\text{fold-terme}$ u , pour toute variable x , si $u[x := v] \in RED_G^C$ pour tout $v \in RED_F^C$, alors $\lambda x.u \in RED_{F \Rightarrow G}^C$.

Question 6 Démontrer que pour tout type F du système μ^+ , pour toute variable de type α , pour tout contexte de candidats C , pour tout $u \in RED_{\mu\alpha.F}^C$, $\mathbf{unfold} \ u$ est dans $RED_{F[\alpha:=\mu\alpha.F]}^C$.

Question 7 Démontrer que pour tout type F du système μ^+ , pour toute variable de type α , pour tout contexte de candidats C , pour tout $u \in RED_{F[\alpha:=\mu\alpha.F]}^C$, $\mathbf{fold} \ u$ est dans $RED_{\mu\alpha.F}^C$.

Question 8 En utilisant tout cela, montrer que tout $\lambda\text{fold-terme}$ typable dans le système μ^+ est fortement normalisable.

2 Une machine à environnements pour le λ -calcul

Nous allons étudier une machine à environnements pour le λ -calcul. Il s'agit d'une machine présentée, sans preuves, pendant l'avant-dernier cours.

On utilise pour ceci un nouveau calcul, le $\lambda\mathbf{x}$ -calcul, dont la grammaire des termes est :

$M, N, \dots ::= x, y, z, \dots$	variables
MN	applications
$\lambda x.M$	λ -abstractions
$M\langle x := N \rangle$	substitution explicite.

Clairement, tous les λ -termes sont des $\lambda\mathbf{x}$ -termes : ceux dont l'opérateur de substitution explicite $\langle _ := _ \rangle$ est absent.

Il n'y a *pas* d'équivalence à α -renommage près pour ces termes. L'ensemble $\text{fv}(M)$ des variables libres d'un $\lambda\mathbf{x}$ -terme M est défini par :

$$\begin{aligned} \text{fv}(x) &\stackrel{\text{def}}{=} \{x\} & \text{fv}(MN) &\stackrel{\text{def}}{=} \text{fv}(M) \cup \text{fv}(N) \\ \text{fv}(\lambda x.M) &\stackrel{\text{def}}{=} \text{fv}(M) \setminus \{x\} & \text{fv}(M\langle x := N \rangle) &\stackrel{\text{def}}{=} (\text{fv}(M) \setminus \{x\}) \cup \text{fv}(N). \end{aligned}$$

On se donne une machine de Krivine dont les configurations sont de la forme $M, env, args$, où :

- M est un $\lambda\mathbf{x}$ -terme ;
- env est une liste $[\langle x_1 := N_1 \rangle; \dots; \langle x_k := N_k \rangle]$ de substitutions explicites ($k \in \mathbb{N}$) ; les variables x_1, \dots, x_k ne sont *pas* supposées distinctes ;
- $args$ est une liste de $\lambda\mathbf{x}$ -termes.

Lorsque $env = [\langle x_1 := N_1 \rangle; \dots; \langle x_k := N_k \rangle]$, on notera $M \text{ env}$ le $\lambda\mathbf{x}$ -terme $M\langle x_1 := N_1 \rangle \dots \langle x_k := N_k \rangle$. L'opération $::$ est comme en Caml l'opération d'ajout d'un élément a en tête d'une liste ℓ , donnant comme résultat $a :: \ell$. Les règles sont les suivantes :

$$\begin{aligned} \text{(explore-app)} \quad & MN, env, args \rightsquigarrow M, env, ((N \text{ env}) :: args) \\ \text{(explore-sub)} \quad & M\langle x := N \rangle, env, args \rightsquigarrow M, (\langle x := N \rangle :: env), args \\ \text{(\beta)} \quad & \lambda x.M, env, (N :: args) \rightsquigarrow M, (\langle x := N \rangle :: env), args \\ \text{(subst-var-drop)} \quad & y, (\langle x := N \rangle :: env), args \rightsquigarrow y, env, args \quad (\text{si } x \neq y) \\ \text{(subst-var)} \quad & x, (\langle x := N \rangle :: env), args \rightsquigarrow N, env, args \end{aligned}$$

Chaque configuration $M, env, args$ représente un λ -terme $\llbracket M, env, args \rrbracket$, défini en deux temps. Dans un premier temps, on définit le λ -terme $\llbracket M \rrbracket$ que représente chaque λx -terme M :

$$\begin{aligned}\llbracket x \rrbracket &\stackrel{\text{def}}{=} x \\ \llbracket MN \rrbracket &\stackrel{\text{def}}{=} \llbracket M \rrbracket \llbracket N \rrbracket \\ \llbracket \lambda x. M \rrbracket &\stackrel{\text{def}}{=} \lambda x. \llbracket M \rrbracket \\ \llbracket M \langle x := N \rangle \rrbracket &\stackrel{\text{def}}{=} \llbracket M \rrbracket [x := \llbracket N \rrbracket]\end{aligned}$$

Dans un deuxième temps, on pose :

$$\llbracket M, env, args \rrbracket \stackrel{\text{def}}{=} \llbracket M \rrbracket [x_1 := \llbracket N_1 \rrbracket] \cdots [x_k := \llbracket N_k \rrbracket] \llbracket P_1 \rrbracket \cdots \llbracket P_m \rrbracket$$

lorsque $env = [\langle x_1 := N_1 \rangle; \cdots; \langle x_k := N_k \rangle]$ et $args = [P_1; \cdots; P_m]$.

Question 9 Montrer que cette machine n'est pas correcte (au moins sans restriction supplémentaire) : exhiber un λ -terme (un λx -terme sans substitution explicite) M et une configuration $N, env, args$ tels que $M, [], [] \rightsquigarrow^+ N, env, arg$, mais $\llbracket M \rrbracket \neq_\beta \llbracket N, env, args \rrbracket$. À titre d'indication, le problème est un problème de capture de variables dans la règle (β) .

Le *domaine* $\text{dom}(env)$ d'un environnement $env \stackrel{\text{def}}{=} [\langle x_1 := N_1 \rangle; \cdots; \langle x_k := N_k \rangle]$ est l'ensemble $\{x_1, \cdots, x_k\}$. L'ensemble des variables *libres* $\text{fv}(env)$ d'un tel environnement est $\bigcup_{i=1}^k (\text{fv}(N_i) \setminus \{x_{i+1}, \cdots, x_k\})$. L'ensemble des variables libres $\text{fv}(args)$ d'une liste $args \stackrel{\text{def}}{=} [P_1; \cdots; P_m]$ est $\bigcup_{j=1}^m \text{fv}(P_j)$. L'ensemble des variables libres $\text{fv}(M, env, args)$ d'une configuration $M, env, args$ est $(\text{fv}(M) \setminus \text{dom}(env)) \cup \text{fv}(env) \cup \text{fv}(args)$. Une configuration est *close* si et seulement son ensemble de variables libres est vide.

Une récurrence simple sur la longueur de env permet de démontrer le lemme suivant, qu'on admettra.

Lemme 7 Pour tout λx -terme N , pour tout environnement env , $\text{fv}(N \ env) = (\text{fv}(N) \setminus \text{dom}(env)) \cup \text{fv}(env)$.

Question 10 Vérifier que si $M, env, args \rightsquigarrow M', env', args'$, et si $\text{fv}(M, env, args) = \emptyset$, alors $\text{fv}(M', env', args') = \emptyset$. En d'autres termes, la relation \rightsquigarrow se restreint aux configurations closes.

On admettra le lemme suivant.

Lemme 8 Pour tous λ -termes u, v , pour toute variable x , $\text{fv}(u[x := v]) \subseteq (\text{fv}(u) \setminus \{x\}) \cup \text{fv}(v)$.

Question 11 Montrer que pour tout λx -terme M , $\text{fv}(\llbracket M \rrbracket) \subseteq \text{fv}(M)$.

Question 12 Montrer que $\text{fv}(\llbracket M, env, args \rrbracket) \subseteq \text{fv}(M, env, args)$.

Question 13 Montrer par un contre-exemple que l'inclusion de la question précédente peut être stricte ; autrement dit, trouver une configuration $M, env, args$ telle que $\text{fv}(\llbracket M, env, args \rrbracket) \subsetneq \text{fv}(M, env, args)$.

On admettra les lemmes suivants.

Lemme 9 Pour tous λ -termes s, u, v et pour toutes variables distinctes x et y , si y n'est pas libre dans u et si x n'est pas libre dans v , alors $s[x := u][y := v] = s[y := v][x := u]$ (le signe = s'interprétant comme la α -équivalence).

Lemme 10 Pour tous λ -termes s, v et pour toutes variables x et z , si z n'est pas libre dans s , alors $s[x := z][z := v] = s[x := v]$ (le signe = s'interprétant comme la α -équivalence).

Question 14 Montrer que la machine de Krivine donnée en début de section est correcte pour les réductions de tête faibles closes : si $M, env, args \rightsquigarrow M', env', args'$ et si $M, env, args$ est close, alors $\llbracket M, env, args \rrbracket \rightarrow_{\text{tf}}^* \llbracket M', env', args' \rrbracket$. Il est facile de voir qu'on a en fait $\llbracket M, env, args \rrbracket = \llbracket M', env', args' \rrbracket$ pour toute règle autre que (β) : je demande donc à ce que vous traitiez le cas de (β) *uniquement*. Vous utiliserez la même notation que moi, qui est de considérer l'instance suivante générique de la règle (β) close :

$$\begin{aligned} & \lambda x.M, [\langle x_1 := N_1 \rangle; \dots; \langle x_k := N_k \rangle], [P_1; \dots; P_m] \\ & \rightsquigarrow M, [\langle x := P_1 \rangle; \langle x_1 := N_1 \rangle; \dots; \langle x_k := N_k \rangle], [P_2; \dots; P_m], \end{aligned}$$

où $m \geq 1$, et où la configuration à gauche de \rightsquigarrow n'a pas de variable libre.

On rappelle que, contrairement aux λx -termes, les λ -termes sont interprétés modulo α -renommage. On rappelle aussi que la relation de réduction de tête faible \rightarrow_{tf} consiste à réduire le redex de tête, s'il y en a un, et s'il n'est pas sous un λ .

Question 15 Montrer que les configurations closes stoppées sont les configurations de la forme $\lambda x.M, env, []$. Citer les règles de réduction qui s'appliqueraient sur toute autre configuration, explicitement. Cette machine implémente donc exactement les réductions de tête faible, sans effectuer explicitement aucune opération de substitution ni d' α -renommage ; mais elle n'est correcte que pour les termes clos.