Examen, λ-calcul, 2023

6 juin 2023

Correction.

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 récursifs

OCaml dispose de types *récursifs*, 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 récursivement, 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 $\mathbf{fold}$ et $\mathbf{unfold}$, que l'on appellera le $\lambda\mathbf{fold}$-calcul.

La syntaxe du $\lambda\mathbf{fold}$-calcul est (modulo α-renommage lui aussi) :

$$\begin{array}{rr} s, t, u, v, \cdots ::= x, y, z, \cdots & \text{variables} \\ \mid uv & \text{applications} \\ \mid \lambda x.u & \lambda\text{-abstractions} \\ \mid \mathbf{fold}\ u & \text{repliage} \\ \mid \mathbf{unfold}\ u & \text{dépliage.} \end{array}$$

Les règles de réduction sont :

$$(\beta) \quad (\lambda x.u)v \to u[x := v]$$
$$(\mathbf{fold}) \quad \mathbf{unfold}(\mathbf{fold}\ u) \to u.$$

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 $x_1 : \pi, \cdots, x_n : \pi$; son *domaine* est $\{x_1, \cdots, x_n\}$.

Question 1 Définir une fonction φ de traduction des λ-termes ordinaires (sans **fold** ni **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 $u \to u'$, on a $\varphi(u) \to^+ \varphi(u')$.

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.

> *La définition de φ ressemble à la définition des modèles du λ-calcul non typé. On pose :*
>
> $$\varphi(x) \stackrel{def}{=} x$$
> $$\varphi(uv) \stackrel{def}{=} \mathbf{unfold}(\varphi(u))(\varphi(v))$$
> $$\varphi(\lambda x.u) \stackrel{def}{=} \mathbf{fold}(\lambda x.\varphi(u)).$$

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.

> *Il suffit de prendre $\varphi(u)$ où u est un λ-terme non normalisable, par exemple $\Omega = \delta\delta$. Le fait que $\varphi(u)$ n'ait pas de forme normale dans le nouveau calcul est dû à la propriété (c), le fait qu'il soit typable à la propriété (b).*

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 :

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

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 $\alpha \notin neg(F)$.

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 $f : A \to A$ 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* $\mathrm{lfp}(f)$.

On définit les candidats de réductibilité pour le $\lambda\mathbf{fold}$-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 $\mathbf{fold}$. SN est l'ensemble des $\lambda\mathbf{fold}$-termes fortement normalisables. Explicitement, un *candidat de réductibilité* est un ensemble S de $\lambda\mathbf{fold}$-termes tel que :

(CR1) $S \subseteq SN$;

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

(CR3) pour tout $\lambda\mathbf{fold}$-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 $\mathcal{C}R$ l'ensemble de tous les candidats de réductibilité pour notre nouveau calcul, ordonné par l'ordre d'inclusion $\subseteq$.

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

> *Il suffit de montrer que toute famille F de candidats de réductibilité a une borne inférieure.*
>
> *Si F est non vide, l'intersection de tous les $S \in F$ est un candidat. Le point délicat est de montrer **(CR1)** : comme $F \neq \emptyset$, on peut fixer $S \in F$; alors, si $u \in \bigcap F$, on a $u \in S$ donc $u \in SN$. **(CR2)** et **(CR3)** ne présentent aucune difficulté.*
>
> *Si F est vide, on cherche à montrer qu'il existe un plus grand minorant de la famille vide, c'est-à-dire un plus grand élément, c'est-à-dire un plus grand candidat de réductibilité : c'est SN, qui est un candidat de réductibilité par le lemme 2, et qui est le plus grand par **(CR1)**.*

On définit :

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

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

> *La démonstration est comme dans le cours. Soit u un $\lambda\mathbf{fold}$-terme neutre dont tous les réduits en une étape sont dans $S \Rightarrow S'$.*
>
> *On montre que u est lui-même dans $S \Rightarrow S'$ en montrant que pour tout $v \in S$, uv est dans S', par récurrence sur $\nu(v)$. (La plus grande longueur de réduction $\nu(v)$ partant de v est bien définie car $v \in S \subseteq SN$, par **(CR1)**.) Les réduits en une étape de uv sont :*
>
> - *$u'v$ avec $u \to u'$; par hypothèse, $u' \in S \Rightarrow S'$, donc $u'v \in S'$;*
> - *uv' avec $v \to v'$; par **(CR2)** sur S, v' est dans S, et comme $\nu(v') < \nu(v)$, on peut utiliser l'hypothèse de récurrence, qui nous donne $uv' \in S'$.*
>
> *Il n'y a pas d'autre cas, car u est neutre, donc ne commence pas par λ. Comme uv est lui-même neutre, par **(CR3)** sur S' on obtient que $uv \in S'$.*

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 :

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

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

(CR1) *Soit $u \in \mathrm{Fold}(S)$. Par définition de* Fold, **unfold** u *est dans S, et est donc fortement normalisable par **(CR1)** sur S. Toute réduction infinie dans u induirait une réduction infinie dans* **unfold** u*, donc $u \in SN$.*

(CR2) *Soit $u \in \mathrm{Fold}(S)$, et supposons $u \to u'$. Alors* **unfold** $u \in S$ *par définition de* Fold, *et* **unfold** $u \to$ **unfold** u'*, donc* **unfold** $u' \in S$ *par **(CR2)** sur S. Par définition de* Fold, u' *est donc dans* $\mathrm{Fold}(S)$.

(CR3) *Soit u un λ***fold***-terme neutre, dont tous les réduits en une étape sont dans* $\mathrm{Fold}(S)$. *Alors les réduits en une étape de* **unfold** u *sont exactement les termes de la forme* **unfold** u' *avec $u \to u'$, puisque u, étant neutre, ne commence pas par* **fold**. *Par hypothèse, pour chacun, on a $u' \in \mathrm{Fold}(S)$, donc* **unfold** $u' \in S$. *On conclut que* **unfold** u *est dans S par **(CR3)** sur S. Donc u est dans* $\mathrm{Fold}(S)$.

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^C_F par récurrence sur F comme suit :

$$\begin{aligned} RED^C_\alpha &\stackrel{\text{def}}{=} C(\alpha) \\ RED^C_{F\Rightarrow G} &\stackrel{\text{def}}{=} RED^C_F \Rightarrow RED^C_G \\ RED^C_{\mu\alpha.F} &\stackrel{\text{def}}{=} \mathrm{lfp}(S \in \mathcal{C}R \mapsto \mathrm{Fold}(RED^{C[\alpha:=S]}_F)). \end{aligned}$$

La notation $S \in \mathcal{C}R \mapsto \mathrm{Fold}(RED^{C[\alpha:=S]}_F)$ désigne la fonction qui à tout S dans $\mathcal{C}R$ associe $\mathrm{Fold}(RED^{C[\alpha:=S]}_F)$.

Il se trouve que la fonction $S \in \mathcal{C}R \mapsto \mathrm{Fold}(RED^{C[\alpha:=S]}_F)$ est monotone si $\alpha \notin neg(F)$. La définition de $RED^C_{\mu\alpha.F}$ 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, $RED^C_{F[\alpha:=G]} = RED_F^{C[\alpha\mapsto RED_G^C]}$.*

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

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 $u \in RED^C_{\mu\alpha.F}$, $\mathbf{unfold}\ u$ est dans $RED^C_{F[\alpha:=\mu\alpha.F]}$.

> *Par définition, $RED^C_{\mu\alpha.F}$ est un point fixe de la fonction $S \in \mathcal{CR} \mapsto \mathrm{Fold}(RED_F^{C[\alpha:=S]})$. Donc, si $u \in RED^C_{\mu\alpha.F}$, alors u est dans $\mathrm{Fold}(RED_F^{C[\alpha:=RED^C_{\mu\alpha.F}]})$. Par définition de* Fold, **unfold** *u est donc dans $RED_F^{C[\alpha:=RED^C_{\mu\alpha.F}]}$, qui est égal à $RED^C_{F[\alpha:=\mu\alpha.F]}$ par le lemme 5.*

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 $u \in RED^C_{F[\alpha:=\mu\alpha.F]}$, $\mathbf{fold}\ u$ est dans $RED^C_{\mu\alpha.F}$.

> *Par définition, $RED^C_{\mu\alpha.F}$ est un point fixe de la fonction $S \in \mathcal{CR} \mapsto \mathrm{Fold}(RED_F^{C[\alpha:=S]})$. Il suffit donc de démontrer que* **fold** *u est dans $\mathrm{Fold}(RED_F^{C[\alpha:=RED^C_{\mu\alpha.F}]})$, ou de façon équivalente, dans $\mathrm{Fold}(RED^C_{F[\alpha:=\mu\alpha.F]})$, grâce au lemme 5.*
>
> *Pour ceci, il suffit de démontrer que $\mathbf{unfold}(\mathbf{fold}\ u)$ est dans $RED^C_{F[\alpha:=\mu\alpha.F]}$. Or $\mathbf{unfold}(\mathbf{fold}\ u)$ est neutre, et on peut donc utiliser **(CR3)**.*
>
> *On effectue une récurrence sur $\nu(u)$ (car u est dans $RED^C_{F[\alpha:=\mu\alpha.F]}$, qui est un candidat de réductibilité, donc dans SN). Les réduits en une étape de $\mathbf{unfold}(\mathbf{fold}\ u)$ sont :*
>
> — *u, par la règle $(\mathbf{fold})$, qui est dans $RED^C_{F[\alpha:=\mu\alpha.F]}$ par hypothèse ;*
>
> — *$\mathbf{unfold}(\mathbf{fold}\ u')$ où $u \to u'$; par **(CR2)**, u' est dans $RED^C_{F[\alpha:=\mu\alpha.F]}$, et $\nu(u') < \nu(u)$, donc on peut appliquer l'hypothèse de récurrence et déduire que $\mathbf{unfold}(\mathbf{fold}\ u')$ est dans $RED^C_{F[\alpha:=\mu\alpha.F]}$.*

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

> *On démontre que pour toute dérivation de typage d'un jugement $\Gamma \vdash u : F$ en système μ^+ (où, disons, Γ s'écrit $x_1 : F_1, \cdots, x_n : F_n$), pour tout contexte de candidats C, pour toute substitution parallèle θ dans $\underline{RED}_\Gamma$, c'est-à-dire qui envoie chaque x_i vers un élément de $RED^C_{F_i}$, $u\theta$ est dans RED^F_C.*
>
> *C'est une récurrence sur la dérivation. Le cas où u est une variable est immédiat. Celui d'une application est par définition de $RED^C_{F\Rightarrow G}$, celui d'une λ-abstraction par le lemme 6. Les deux nouveaux cas sont les règles $(Fold)$ et $(Unfold)$.*
>
> — *$(Fold)$: par hypothèse, $u\theta \in RED^C_{F[\alpha:=\mu\alpha.F]}$ pour tout contexte de candidats C, pour toute $\theta \in \underline{RED}_\Gamma$. Par la **Question 7**, $\mathbf{fold}(u\theta) = (\mathbf{fold}\ u)\theta$ est dans $RED^C_{\mu\alpha.F}$.*
>
> — *$(Unfold)$: similairement, si $u\theta \in RED^C_{\mu\alpha.F}$, alors $\mathbf{unfold}(u\theta) = (\mathbf{unfold}\ u)\theta$ est dans $RED^C_{F[\alpha:=\mu\alpha.F]}$ par la **Question 6**.*
>
> *Ceci étant démontré, on prend pour C le contexte de candidats qui à toute variable associe SN, pour θ la substitution identité (qui est dans $\underline{RED}_\Gamma$ car toute variable est dans tout candidat de réductibilité, grâce à **(CR3)**), et on obtient que u est dans RED^C_F ; donc u est dans SN, par **(CR1)**.*

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 :

$$\begin{array}{rr} M, N, \cdots ::= x, y, z, \cdots & \text{variables} \\ \mid MN & \text{applications} \\ \mid \lambda x.M & \lambda\text{-abstractions} \\ \mid M\langle x := N\rangle & \text{substitution explicite.} \end{array}$$

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

$\mathrm{fv}(M)$ des variables libres d'un λx-terme M est défini par :

$$\mathrm{fv}(x) \stackrel{\text{def}}{=} \{x\} \qquad \mathrm{fv}(MN) \stackrel{\text{def}}{=} \mathrm{fv}(M) \cup \mathrm{fv}(N)$$
$$\mathrm{fv}(\lambda x.M) \stackrel{\text{def}}{=} \mathrm{fv}(M) \setminus \{x\} \qquad \mathrm{fv}(M\langle x := N\rangle) \stackrel{\text{def}}{=} (\mathrm{fv}(M) \setminus \{x\}) \cup \mathrm{fv}(N).$$

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

— M est un λx-terme ;

— env est une liste $[\langle x_1 := N_1\rangle; \cdots; \langle x_k := N_k\rangle]$ de substitutions explicites ($k \in \mathbb{N}$) ; les variables $x_1, \ldots, x_k$ ne sont *pas* supposées distinctes ;

— $args$ est une liste de λx-termes.

Lorsque $env = [\langle x_1 := N_1\rangle; \cdots; \langle x_k := N_k\rangle]$, on notera $M\ env$ le λx-terme $M\langle x_1 := N_1\rangle \cdots \langle x_k := N_k\rangle$. 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 :

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

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

$$[\![x]\!] \stackrel{\text{def}}{=} x$$
$$[\![MN]\!] \stackrel{\text{def}}{=} [\![M]\!][\![N]\!]$$
$$[\![\lambda x.M]\!] \stackrel{\text{def}}{=} \lambda x.[\![M]\!]$$
$$[\![M\langle x := N\rangle]\!] \stackrel{\text{def}}{=} [\![M]\!][x := [\![N]\!]]$$

Dans un deuxième temps, on pose :

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

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 $[\![M]\!] \neq_\beta [\![N, env, args]\!]$. À titre d'indication, le problème est un problème de capture de variables dans la règle (β).

> *Il s'agit d'un problème de funarg classique. Par exemple,* $M \stackrel{def}{=} (\lambda x.\lambda y.y)yx$*, où* x *et* y *sont deux variables distinctes, mène à :*
>
> $$\begin{aligned} M, [], [] &\rightsquigarrow^2 \lambda x.\lambda y.y, [], [y; x] && (\textit{explore-app}) \\ &\rightsquigarrow \lambda y.y, [\langle x := y\rangle], [x] && (\beta) \\ &\rightsquigarrow y, [\langle y := x\rangle; \langle x := y\rangle], [] && (\beta) \\ &\rightsquigarrow x, [\langle x := y\rangle], [] && (\textit{subst-var}) \\ &\rightsquigarrow y, [], [] && (\textit{subst-var}). \end{aligned}$$
>
> *On pose* $N \stackrel{def}{=} y$*,* $env \stackrel{def}{=} []$*,* $args \stackrel{def}{=} []$*.* $[\![M]\!]$ *a comme forme* β*-normale* x*; or* $[\![N, env, args]\!] = y$*, qui est en forme normale. Comme* $x \neq y$*, par* confluence $[\![M]\!] \neq_\beta [\![N, env, args]\!]$*.*

Le *domaine* $\mathrm{dom}(env)$ 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* $\mathrm{fv}(env)$ d'un tel environnement est $\bigcup_{i=1}^{k}(\mathrm{fv}(N_i) \setminus \{x_{i+1}, \cdots, x_k\})$. L'ensemble des variables libres $\mathrm{fv}(args)$ d'une liste $args \stackrel{\text{def}}{=} [P_1; \cdots; P_m]$ est $\bigcup_{j=1}^{m} \mathrm{fv}(P_j)$. L'ensemble des variables libres $\mathrm{fv}(M, env, args)$ d'une configuration $M, env, args$ est $(\mathrm{fv}(M) \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(env) \cup \mathrm{fv}(args)$. 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*,* $\mathrm{fv}(N\ env) = (\mathrm{fv}(N) \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(env)$.

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

> *On regarde chaque règle l'une après l'autre. Je reprends les notations propres à chaque règle.*
>
> — (*explore-app*) *: on a :*
>
> $$\begin{aligned} \mathrm{fv}(MN, env, args) = ((&\mathrm{fv}(M) \cup \mathrm{fv}(N)) \setminus \mathrm{dom}(env)) \\ &\cup \mathrm{fv}(env) \cup \mathrm{fv}(args). \end{aligned}$$

Si ceci est vide, alors $\mathrm{fv}(env) = \mathrm{fv}(args) = \emptyset$ *et* $\mathrm{fv}(M), \mathrm{fv}(N) \subseteq \mathrm{dom}(env)$.
On a alors $\mathrm{fv}(M, env, (N\ env) :: args) = (\mathrm{fv}(M) \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(N\ \ env) = \mathrm{fv}(N\ \ env)$ *(puisque* $\mathrm{fv}(M) \subseteq \mathrm{dom}(env)$*)* $= (\mathrm{fv}(N) \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(env)$ *(par le lemme 7)* $= \emptyset$*, puisque* $\mathrm{fv}(N) \subseteq \mathrm{dom}(env)$ *et* $\mathrm{fv}(env) = \emptyset$.
On peut aussi démontrer directement que $\mathrm{fv}(MN, env, args) = \mathrm{fv}(M, env, (N\ \ env) :: args)$*, sans aucune hypothèse sur aucun côté de l'égalité.*

— (*explore-sub*) *: supposons* $env = [\langle x_1 := N_1\rangle; \cdots; \langle x_k := N_k\rangle]$*, alors :*

$$\begin{aligned}
&\mathrm{fv}(M\langle x := N\rangle, env, args)\\
&= (((\mathrm{fv}(M) \setminus \{x\}) \cup \mathrm{fv}(N)) \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(env) \cup \mathrm{fv}(args)\\
&= (\mathrm{fv}(M) \setminus \{x, x_1, \cdots, x_k\}) \cup (\mathrm{fv}(N) \setminus \{x_1, \cdots, x_k\})\\
&\qquad \cup \bigcup_{i=1}^{k} (\mathrm{fv}(N_i) \setminus \{x_{i+1}, \cdots, x_k\}) \cup \mathrm{fv}(args)\\
&= (\mathrm{fv}(M) \setminus \mathrm{dom}(\langle x := N\rangle :: env)) \cup \mathrm{fv}(\langle x := N\rangle :: env) \cup \mathrm{fv}(args)\\
&= \mathrm{fv}(M, \langle x := N\rangle :: env, args).
\end{aligned}$$

— (β) *: on a :*

$$\begin{aligned}
\mathrm{fv}(\lambda x.M, env, N :: args) = ((\mathrm{fv}(M) \setminus \{x\}) \setminus \mathrm{dom}(env))\\
\cup \mathrm{fv}(env) \cup \mathrm{fv}(N) \cup \mathrm{fv}(args).
\end{aligned}$$

Ceci étant vide par hypothèse, on a $\mathrm{fv}(env) = \mathrm{fv}(N) = \mathrm{fv}(args) = \emptyset$ *et* $\mathrm{fv}(M) \subseteq \{x\} \cup \mathrm{dom}(env)$*. Par ailleurs, on a :*

$$\begin{aligned}
\mathrm{fv}(M, \langle x := N\rangle :: env, args) = (\mathrm{fv}(M) \setminus (\{x] \cup \mathrm{dom}(env)))\\
\cup (\mathrm{fv}(N) \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(env) \cup \mathrm{fv}(args),
\end{aligned}$$

qui est vide aussi car $\mathrm{fv}(M) \subseteq \{x\} \cup \mathrm{dom}(env)$*,* $\mathrm{fv}(N) = \emptyset$ *donc* $\mathrm{fv}(N) \setminus \mathrm{dom}(env) = \emptyset$*, et* $\mathrm{fv}(env) = \mathrm{fv}(args) = \emptyset$.
On pourrait en fait démontrer que $\mathrm{fv}(M\langle x := N\rangle, env, args) \subseteq \mathrm{fv}(\lambda x.M, env, N :: args)$*; on n'a pas l'égalité en général, mais l'inclusion s'obtient car* $\mathrm{fv}(N) \setminus \mathrm{dom}(env) \subseteq \mathrm{fv}(N)$.

— (*subst-var-drop*). *On a :*

$$\begin{aligned}
\mathrm{fv}(y, \langle x := N\rangle :: env, args) &= (\{y\} \setminus (\{x\} \cup \mathrm{dom}(env)))\\
&\quad \cup (\mathrm{fv}(N) \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(env) \cup \mathrm{fv}(args)\\
&= (\{y\} \setminus \mathrm{dom}(env))\\
&\quad \cup (\mathrm{fv}(N) \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(env) \cup \mathrm{fv}(args)
\end{aligned}$$

car $y \neq x$. Ceci étant vide, $y \in \mathrm{dom}(env)$ et $\mathrm{fv}(env) = \mathrm{fv}(args) = \emptyset$. Dans ces conditions,

$$\mathrm{fv}(y, env, args) = (\{y\} \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(env) \cup \mathrm{fv}(args)$$

est vide aussi.

— (*subst-var*). *On a :*

$$\mathrm{fv}(x, \langle x := N\rangle :: env, args) = (\{x\} \setminus (\{x\} \cup \mathrm{dom}(env))) \cup (\mathrm{fv}(N) \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(env) \cup \mathrm{fv}(args).$$

Ceci étant vide, on a $\mathrm{fv}(N) \subseteq \mathrm{dom}(env)$ et $\mathrm{fv}(env) = \mathrm{fv}(args) = \emptyset$. Donc :

$$\mathrm{fv}(N, env, args) = (\mathrm{fv}(N) \setminus \mathrm{dom}(env)) \cup \mathrm{fv}(env) \cup \mathrm{fv}(args)$$

est vide aussi.

On admettra le lemme suivant.

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

Question 11 Montrer que pour tout λx-terme M, $\mathrm{fv}([\![M]\!]) \subseteq \mathrm{fv}(M)$.

Par récurrence structurelle sur M. C'est évident pour tous les cas sauf peut-être pour les termes de la forme $M\langle x := N\rangle$. On a :

$$\begin{aligned}
\mathrm{fv}([\![M\langle x := N\rangle]\!]) &= \mathrm{fv}([\![M]\!][x := [\![N]\!]]) \\
&\subseteq (\mathrm{fv}([\![M]\!]) \setminus \{x\}) \cup \mathrm{fv}([\![N]\!]) && \textit{par le lemme 8} \\
&\subseteq (\mathrm{fv}(M) \setminus \{x\}) \cup \mathrm{fv}(N) && \textit{par hypothèse de récurrence} \\
&= \mathrm{fv}(M\langle x := N\rangle).
\end{aligned}$$

Question 12 Montrer que $\mathrm{fv}([\![M, env, args]\!]) \subseteq \mathrm{fv}(M, env, args)$.

En posant $env \stackrel{def}{=} [\langle x_1 := N_1\rangle; \cdots; \langle x_k := N_k\rangle]$, $args \stackrel{def}{=} [P_1; \cdots; P_m]$, on a :

$$\begin{aligned}
\mathrm{fv}([\![M, env, args]\!]) &= \mathrm{fv}([\![M]\!][x_1 := [\![N_1]\!]] \cdots [x_k := [\![N_k]\!]][\![P_1]\!] \cdots [\![P_m]\!]) \\
&= \mathrm{fv}([\![M]\!][x_1 := [\![N_1]\!]] \cdots [x_k := [\![N_k]\!]]) \cup \bigcup_{j=1}^{m} \mathrm{fv}([\![P_j]\!]).
\end{aligned}$$

On démontre que $\mathrm{fv}([\![M]\!][x_1 := [\![N_1]\!]] \cdots [x_k := [\![N_k]\!]]) \subseteq (\mathrm{fv}([\![M]\!]) \setminus \{x_1, \cdots, x_k\}) \cup \bigcup_{i=1}^{k} (\mathrm{fv}([\![N_i]\!]) \setminus \{x_{i+1}, \cdots, x_k\})$ *par récurrence sur* k*.*

— *Si* $k = 0$*, c'est évident.*

— *Sinon, on a :*

$$\begin{aligned}
&\mathrm{fv}([\![M]\!][x_1 := [\![N_1]\!]] \cdots [x_k := [\![N_k]\!]]) \\
&\subseteq (\mathrm{fv}([\![M]\!][x_1 := [\![N_1]\!]] \cdots [x_{k-1} := [\![N_{k-1}]\!]]) \setminus \{x_k\}) \cup \mathrm{fv}([\![N_k]\!]) \\
&\qquad \textit{par le lemme 8} \\
&\subseteq (((\mathrm{fv}([\![M]\!]) \setminus \{x_1, \cdots, x_{k-1}\}) \\
&\qquad \cup \bigcup_{i=1}^{k-1} (\mathrm{fv}([\![N_i]\!]) \setminus \{x_{i+1}, \cdots, x_{k-1}\})) \setminus \{x_k\}) \cup \mathrm{fv}([\![N_k]\!]) \\
&\qquad \textit{par hypothèse de récurrence} \\
&= (\mathrm{fv}([\![M]\!]) \setminus \{x_1, \cdots, x_k\}) \cup \bigcup_{i=1}^{k} (\mathrm{fv}([\![N_i]\!]) \setminus \{x_{i+1}, \cdots, x_k\}).
\end{aligned}$$

Ceci étant fait, on a :

$$\begin{aligned}
\mathrm{fv}([\![M, env, args]\!]) &\subseteq (\mathrm{fv}([\![M]\!]) \setminus \{x_1, \cdots, x_k\}) \\
&\qquad \cup \bigcup_{i=1}^{k} (\mathrm{fv}([\![N_i]\!]) \setminus \{x_{i+1}, \cdots, x_k\}) \cup \bigcup_{j=1}^{m} \mathrm{fv}([\![P_j]\!]) \\
&\subseteq (\mathrm{fv}(M) \setminus \{x_1, \cdots, x_k\}) \\
&\qquad \cup \bigcup_{i=1}^{k} (\mathrm{fv}(N_i) \setminus \{x_{i+1}, \cdots, x_k\}) \cup \bigcup_{j=1}^{m} \mathrm{fv}(P_j) \\
&= \mathrm{fv}(M, env, args),
\end{aligned}$$

en utilisant la ***Question 11*** *sur* M*, les* N_i*, et les* P_j*.*

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 $\mathrm{fv}([\![M, env, args]\!]) \subsetneq \mathrm{fv}(M, env, args)$.

Si $M \stackrel{def}{=} x$*,* $env \stackrel{def}{=} [\langle y := z \rangle]$ *(où* x*,* y*,* z *sont deux à deux distinctes),* $args \stackrel{def}{=} []$*, alors* $\mathrm{fv}([\![M, env, args]\!]) = \mathrm{fv}(x) = \{x\}$*. Mais* $\mathrm{fv}(M) = \{x\}$*,* $\mathrm{dom}(env) = \{y\}$*,* $\mathrm{fv}(env) = \{z\}$*,* $\mathrm{fv}(args) = \emptyset$*, donc* $\mathrm{fv}(M, env, args) = \{x, z\}$*, ce qui est différent.*

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 $[\![M, env, args]\!] \rightarrow_{\text{tf}}^{*} [\![M', env', args']\!]$. Il est facile de voir qu'on a en fait $[\![M, env, args]\!] = [\![M', env', args']\!]$ 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 :

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

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 $\rightarrow_{\text{tf}}$ consiste à réduire le rédex de tête, s'il y en a un, et s'il n'est pas sous un λ.

On a :

$$
\begin{aligned}
&[\![\lambda x.M, [\langle x_1 := N_1\rangle; \cdots; \langle x_k := N_k\rangle], [P_1; \cdots; P_m]]\!] \\
&= [\![\lambda x.M]\!][x_1 := [\![N_1]\!]] \cdots [x_k := [\![N_k]\!]][\![P_1]\!] \cdots [\![P_m]\!] \\
&= (\lambda z.[\![M]\!][x := z])[x_1 := [\![N_1]\!]] \cdots [x_k := [\![N_k]\!]][\![P_1]\!] \cdots [\![P_m]\!] \\
&\qquad \textit{où } \underline{z \textit{ est fraîche}}\textit{, c'est-à-dire} \\
&\qquad \textit{différente des } x_i \textit{ et non libre dans } M \textit{ ni dans aucun } N_i\textit{,} \\
&\qquad \textit{donc dans aucun } [\![N_i]\!]\textit{, par la } \textbf{\textit{Question 11}} \\
&= (\lambda z.[\![M]\!][x := z][x_1 := [\![N_1]\!]] \cdots [x_k := [\![N_k]\!]])[\![P_1]\!] \cdots [\![P_m]\!] \\
&\qquad \textit{car } z \textit{ est fraîche, au sens de la ligne précédente} \\
&\to_{tf} [\![M]\!][x := z][x_1 := [\![N_1]\!]] \cdots [x_k := [\![N_k]\!]][z := [\![P_1]\!]][\![P_2]\!] \cdots [\![P_m]\!] \\
&= [\![M]\!][x := z][z := [\![P_1]\!]][x_1 := [\![N_1]\!]] \cdots [x_k := [\![N_k]\!]][\![P_2]\!] \cdots [\![P_m]\!] \\
&\qquad \textit{car } z \textit{ est fraîche (même sens)} \\
&\qquad \textit{et } \mathrm{fv}([\![P_1]\!]) \subseteq \mathrm{fv}(P_1) \textit{ (}\textbf{\textit{Question 11}}\textit{)} = \emptyset \textit{ (hypothèse)} \\
&\qquad \textit{ce qui permet d'utiliser le } \underline{\textit{lemme 9}}\ k \textit{ fois} \\
&= [\![M]\!][x := [\![P_1]\!]][x_1 := [\![N_1]\!]] \cdots [x_k := [\![N_k]\!]][\![P_2]\!] \cdots [\![P_m]\!] \\
&\qquad \textit{par le } \underline{\textit{lemme 10}}\textit{,} \\
&\qquad \textit{car } z \textit{ n'est pas libre dans } [\![M]\!] \\
&= [\![M, [\langle x := P_1\rangle, \langle x_1 := N_1\rangle; \cdots; \langle x_k := N_k\rangle], [P_2; \cdots; P_m]]\!].
\end{aligned}
$$

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.

Soit $M, env, args$ une configuration close stoppée. M ne peut pas être une application, sinon (explore-app) s'appliquerait. Si M est une λ-abstraction, $args$ doit être vide, sinon (β) s'appliquerait ; et donc la configuration est de la forme demandée.

Si M est une variable x, comme $\mathrm{fv}(M, env, args) = \emptyset$, on a nécessairement $\mathrm{fv}(M) \setminus \mathrm{dom}(env) = \emptyset$. Donc, si env est de la forme $[\langle x_1 := N_1\rangle; \cdots; \langle x_k := N_k\rangle]$, x est l'une des variables x_i. Soit i le plus petit indice tel que $x = x_i$. Si $i = 1$, la règle (subst-var) s'applique. Sinon, on a $i \geq 2$ et en particulier $k \neq 0$, donc (subst-var-drop) s'applique. Aucun de ces cas n'est possible, donc M n'est pas une variable.