

Examen Programmation I (2022-23)

Tous documents écrits autorisés. Pas d'ordinateur, de tablette, de smartphone.

On insistera sur la correction et la clarté des réponses.

Toute affirmation devra être justifiée explicitement, par un théorème du cours, par un numéro de question, par une référence à une règle.

N'inventez pas vos propres notations, et réutilisez les miennes, même si elles ne vous plaisent pas.

Je veux une copie au propre, pas un brouillon : pas de rature, notamment. L'orthographe, et plus généralement la présentation, est importante.

Je corrigerai vraisemblablement toutes les copies en corrigeant d'abord toutes les questions 1 de tout le monde, puis toutes les questions 2, et ainsi de suite : ne comptez pas sur le fait que je me souvienne de ce que vous avez écrit dans une question précédente.

J'appellerai *fonction* ce qu'on appelle plus communément une application, c'est-à-dire une fonction totale. On ne les confondra pas avec les *noms de fonction* (voir plus bas).

★ ★ ★

Le sujet porte sur un langage permettant le calcul sur des *réels exacts*. Contrairement aux nombres en virgule flottante, les calculs sur ces réels ne font aucune erreur d'arrondi. Une représentation agréable de ces réels (entre -1 et 1) est sous forme de liste infinie (paresseuse) $x_1 : x_2 : \dots : x_k : \dots$ de nombres dans $\{-1, 0, 1\}$. Une telle liste représente $\sum_{k=1}^{+\infty} x_k/2^k$; un nombre est en général représenté par plusieurs listes infinies : la représentation est redondante.

On peut coder toutes les fonctions usuelles sur les réels sur cette représentation. Si ça vous intéresse, je vous recommande : Weng Kin Ho, *Exact Real Calculator for Everyone*, Proceedings of the 1st Asian Technology Conference in Mathematics (ATCM'13), Mumbai, India, 2013, Yang, W.-C. and Majwesi, M. and Alwis, T. and Rana, I. K., editors.

On va étudier un sous-langage de Haskell qui sera suffisant pour écrire ce genre de programmes : $\text{Hask}\mathbb{R}$. Nous en donnerons deux variantes : une version sans aucune annotation de type (Section 1), et une version où toutes les variables, tous les noms de fonctions ont un type unique (Section 2).

1 Le langage $\text{Hask}\mathbb{R}$: typage

Les types de $\text{Hask}\mathbb{R}$ sont I et int (rien d'autre). Les termes sont décrits, avec les conventions usuelles, par :

M, N, P, \dots		termes
$::= x, y, z, \dots$		variables
$ f(M_1, M_2, \dots, M_n)$	application de nom de fonction ($n \in \mathbb{N}$)	
$ \dot{n}$	constantes entières ($n \in \mathbb{Z}$)	
$ M \dot{+} N$	addition entière	
$ M : N$	constructeur de liste	
$ \text{hd } M$	premier élément de liste	
$ \text{tl } M$	reste de liste.	

Le constructeur de liste $M : N$ serait noté en $M :: N$ en OCaml; nous choisissons ici la syntaxe Haskell. Les symboles f des applications sont pris dans un ensemble Fun , supposé disjoint de l'ensemble Var des variables x, y, z, \dots , et appelé l'ensemble des *noms de fonctions*.

Un *programme* π est intuitivement une liste de déclarations de fonctions (récurives, et même mutuellement récurives). Formellement, π est une fonction d'un sous-ensemble fini de Fun vers des *définitions* de fonctions : pour chaque nom de fonction f dans le domaine de π , $\pi(f)$ est un couple $\langle (x_1, x_2, \dots, x_n), M \rangle$ où M est un terme et où x_1, x_2, \dots, x_n sont des variables distinctes qui contiennent toutes les variables qui apparaissent dans M .

Les règles de typage sont :

$$\begin{array}{c}
\frac{}{\Gamma, x: \tau \vdash x: \tau} \text{ (Var :)} \\
\\
\frac{}{\Gamma \vdash \dot{n}: \text{int}} \text{ (Cst :)} \\
\\
\frac{\Gamma \vdash M: \text{int} \quad \Gamma \vdash N: \text{int}}{\Gamma \vdash M \dot{+} N: \text{int}} \text{ (Plus :)} \\
\\
\frac{\Gamma \vdash M: \text{int} \quad \Gamma \vdash N: \text{I}}{\Gamma \vdash M: N: \text{I}} \text{ (Cons :)} \quad \frac{\Gamma \vdash M: \text{I}}{\Gamma \vdash \text{hd } M: \text{int}} \text{ (Hd :)} \quad \frac{\Gamma \vdash M: \text{I}}{\Gamma \vdash \text{tl } M: \text{I}} \text{ (Tl :)} \\
\\
\frac{\Gamma \vdash M_1: \tau_1 \cdots \Gamma \vdash M_n: \tau_n}{\Gamma \vdash f(M_1, \dots, M_n): \tau} \text{ (App :)} \\
\text{si } \Gamma \text{ contient } f: \tau_1, \dots, \tau_n \rightarrow \tau
\end{array}$$

Dans ces règles, $\tau, \tau_1, \dots, \tau_n$ sont des types, donc soit **int** soit **I**. Un contexte Γ est un ensemble de *liaisons* de la forme $x: \tau$ ($x \in \text{Var}$) ou $f: \tau_1, \dots, \tau_n \rightarrow \tau$ ($f \in \text{Fun}$), les mêmes variables ou noms de fonctions n'apparaissant qu'au plus une fois à gauche du symbole de typage $:$.

On dit qu'un programme π est *bien typé* si et seulement s'il existe un contexte Γ tel que, pour tout f dans le domaine de π , et si $\pi(f) = \langle (x_1, x_2, \dots, x_n), M \rangle$, alors Γ contient une liaison de la forme $f: \tau_1, \dots, \tau_n \rightarrow \tau$ (avec le même n) et l'on peut dériver :

$$\Gamma, x_1: \tau_1, \dots, x_n: \tau_n \vdash M: \tau$$

dans le système de typage ci-dessus. Un tel contexte Γ sera appelé un *typage* de π .

Question 1 Donner un algorithme efficace (en temps polynomial) permettant de décider si π est bien typé, et le cas échéant, en retourne un typage principal. Je n'ai pas besoin de preuve de correction, mais l'algorithme devra être correct.

2 Le langage $\text{Hask}\mathbb{R}$: sémantique

On suppose dorénavant tous les programmes et termes que nous manipulerons bien typés, tous dans le même contexte de typage Γ . Chaque variable x a donc un type unique σ , ce que l'on explicitera parfois en écrivant x_σ à la place de x ; chaque nom de fonction f a un type unique $\tau_1, \dots, \tau_n \rightarrow \tau$, ce que l'on explicitera parfois par la notation $f_{\tau_1, \dots, \tau_n \rightarrow \tau}$; et tout terme a lui aussi un type uniquement déterminé.

On définit une sémantique opérationnelle à petits pas de $\text{Hask}\mathbb{R}$ par les règles de la figure 1. Les configurations sont juste les termes du langage.

$$\begin{array}{c}
\frac{}{f(N_1, N_2, \dots, N_n) \rightarrow_\pi P[x_1 := N_1, x_2 := N_2, \dots, x_n := N_n]} \text{ (App)} \\
\text{où } \pi(f) = \langle (x_1, x_2, \dots, x_n), P \rangle \\
\\
\frac{M \rightarrow_\pi M'}{M \dot{+} N \rightarrow_\pi M' \dot{+} N} (\dot{+}L) \quad \frac{N \rightarrow_\pi N'}{M \dot{+} N \rightarrow_\pi M \dot{+} N'} (\dot{+}R) \quad \frac{}{\dot{m} \dot{+} \dot{n} \rightarrow_\pi \dot{p}} (\dot{+}) \\
\text{où } p = m + n \\
\\
\frac{M \rightarrow_\pi M'}{\text{hd } M \rightarrow_\pi \text{hd } M'} (\text{hd } L) \quad \frac{}{\text{hd } (M : N) \rightarrow_\pi M} (\text{hd}) \\
\\
\frac{M \rightarrow_\pi M'}{\text{tl } M \rightarrow_\pi \text{tl } M'} (\text{tl } L) \quad \frac{}{\text{tl } (M : N) \rightarrow_\pi N} (\text{tl})
\end{array}$$

FIGURE 1 – Sémantique opérationnelle de $\text{Hask}_{\mathbb{R}}$

$M[x_1 := N_1, x_2 := N_2, \dots, x_n := N_n]$ est le terme obtenu à partir de M en remplaçant simultanément chaque occurrence d'une variable x_i par l'expression N_i correspondante.

Question 2 On dit qu'un terme M est *admissible* (pour le programme π) si et seulement si (il est bien typé et) s'il ne contient pas de variable, et si tous les noms de fonctions qui apparaissent dans M sont dans le domaine de π .

Une *p-valeur* dans notre cadre est un terme admissible M qui a terminé, c'est-à-dire telle que $M \rightarrow_\pi M'$ n'est dérivable pour aucun terme M' . Par exemple, n'importe quel terme admissible de la forme $M : N$ est une p-valeur.

Répondre aux deux sous-questions suivantes. Je n'ai pas besoin de justification, mais le résultat devra être correct.

- (a) Quelles sont les p-valeurs de type \mathbf{I} ?
- (b) Quelles sont les p-valeurs de type \mathbf{int} ?

On définit une sémantique dénotationnelle comme suit. On pose, pour les types :

$$[[\mathbf{int}]] = \mathbb{Z}_{\perp} \quad [[\mathbf{I}]] = [\mathbb{N} \rightarrow \mathbb{Z}_{\perp}].$$

On rappelle que \mathbb{Z}_{\perp} est le dcpo plat formé de tous les entiers relatifs, incomparables deux à deux, plus un élément \perp inférieur à tous les entiers. \mathbb{N} est

$$\begin{aligned}
\llbracket x \rrbracket \rho &= \rho(x) \\
\llbracket f(N_1, N_2, \dots, N_n) \rrbracket \rho &= \rho(f)(\llbracket N_1 \rrbracket \rho, \llbracket N_2 \rrbracket \rho, \dots, \llbracket N_n \rrbracket \rho) \\
\llbracket \dot{n} \rrbracket \rho &= n \\
\llbracket M \dot{+} N \rrbracket \rho &= \begin{cases} \llbracket M \rrbracket \rho + \llbracket N \rrbracket \rho & \text{si } \llbracket M \rrbracket \rho \neq \perp \text{ et } \llbracket N \rrbracket \rho \neq \perp \\ \perp & \text{sinon} \end{cases} \\
\llbracket M : N \rrbracket \rho &= \text{push}(\llbracket M \rrbracket \rho, \llbracket N \rrbracket \rho) \\
\llbracket \mathbf{hd} M \rrbracket \rho &= \text{hd } \llbracket M \rrbracket \rho \\
\llbracket \mathbf{tl} M \rrbracket \rho &= \text{tl } \llbracket M \rrbracket \rho
\end{aligned}$$

FIGURE 2 – Sémantique dénotationnelle de $\text{Hask}\mathbb{R}$

équipé de l'ordre $=$, donc $[\mathbb{N} \rightarrow \mathbb{Z}_\perp]$ n'est rien d'autre que le depo des suites infinies d'éléments de \mathbb{Z}_\perp , ordonnées composante par composante.

La sémantique elle-même est donnée en figure 2. Dans cette sémantique, on utilise des environnements ρ . Ce sont des fonctions qui à chaque variable x_σ associe une valeur de $[\sigma]$ et à chaque nom de fonction $f_{\sigma_1, \sigma_2, \dots, \sigma_n \rightarrow \tau}$ associe une fonction Scott-continue de $[\sigma_1] \times [\sigma_2] \times \dots \times [\sigma_n]$ vers $[\tau]$.

Pour une liste infinie $\ell \in [\mathbb{I}]$, on notera $\text{hd } \ell$ l'élément $\ell(0)$ de \mathbb{Z}_\perp , et $\text{tl } \ell$ la liste définie par $\text{tl } \ell(n) = \ell(n+1)$ ($n \in \mathbb{N}$).

On notera aussi $\text{push}(m, \ell)$ la liste dont l'élément à l'indice 0 vaut m , et dont l'élément à l'indice $n+1$ est $\ell(n)$, pour tout $n \in \mathbb{N}$.

Question 3 Soit M un terme fixé. On souhaite montrer que la fonction qui à $\rho \in \text{Env}$ associe $\llbracket M \rrbracket \rho$ est Scott-continue. On notera cette fonction $\llbracket M \rrbracket$, plus brièvement.

(a) Sur quoi se fait la récurrence ?

On fait ensuite une analyse de cas. Par exemple, si M est de la forme $N : P$, l'argument repose sur le fait que push est Scott-continue de $\mathbb{Z}_\perp \times [\mathbb{N} \rightarrow \mathbb{Z}_\perp]$ vers $[\mathbb{N} \rightarrow \mathbb{Z}_\perp]$.

(b) Montrer que push est Scott-continue.

Dans la suite, nous admettrons tous les autres cas : la fonction $\llbracket M \rrbracket$ est Scott-continue.

Pour chaque nom de fonction $f_{\sigma_1, \sigma_2, \dots, \sigma_n \rightarrow \tau}$, on notera D_f le depo $[[\sigma_1] \times [\sigma_2] \times \dots \times [\sigma_n] \rightarrow [\tau]]$.

Pour un environnement ρ , et une définition de fonction $\pi(f) = \langle (x_1 \sigma_1, x_2 \sigma_2, \dots, x_n \sigma_n), P \rangle$, on notera $\text{Val}(f, \rho)$ la fonction qui à tout $(V_1, V_2, \dots, V_n) \in [\sigma_1] \times [\sigma_2] \times \dots \times [\sigma_n]$ associe $\llbracket P \rrbracket [x_1 \mapsto V_1, x_2 \mapsto V_2, \dots, x_n \mapsto V_n]$.

On note $Env = \prod_f \text{nom de fonction } D_f \times \prod_{x_\sigma} \text{variable } \llbracket \sigma \rrbracket$. C'est le dcpo des environnements. (C'est un dcpo car tout produit de dcpos est un dcpo. La démonstration se fait comme pour le cas des produits binaires, à savoir le lemme 6 de `prog1_sem2.pdf`, et on n'aura pas à le démontrer.) Étant donné un programme π , de domaine $\{f_1, f_2, \dots, f_p\}$, on pose, pour tout $\rho \in Env$:

$$F_\pi(\rho) \stackrel{\text{def}}{=} \rho[f_1 \mapsto Val(f_1, \rho), f_2 \mapsto Val(f_2, \rho), \dots, f_p \mapsto Val(f_p, \rho)].$$

On admettra que F_π est une fonction Scott-continue de Env vers Env . C'est une conséquence de la **Question 3**.

Question 4 Pourquoi F_π a-t-elle un plus petit point fixe ? On énoncera le théorème, et on expliquera pourquoi ses hypothèses sont vérifiées.

Par la suite, on notera $\rho_\pi = \text{lfp}(F_\pi)$ ce plus petit point fixe.

Question 5 On admettra le résultat suivant : pour tout environnement ρ , si $\rho(x_1) = \llbracket N_1 \rrbracket \rho$, $\rho(x_2) = \llbracket N_2 \rrbracket \rho$, \dots , $\rho(x_n) = \llbracket N_n \rrbracket \rho$, alors :

$$\begin{aligned} & \llbracket M[x_1 := N_1, x_2 := N_2, \dots, x_n := N_n] \rrbracket \rho \\ &= \llbracket M \rrbracket (\rho[x_1 \mapsto \llbracket N_1 \rrbracket \rho, x_2 \mapsto \llbracket N_2 \rrbracket \rho, \dots, \llbracket N_n \rrbracket \rho]). \end{aligned}$$

On souhaite montrer le résultat de correction suivant : si $M \rightarrow_\pi M'$ alors $\llbracket M \rrbracket \rho_\pi = \llbracket M' \rrbracket \rho_\pi$. Montrez-le dans le cas où $M \rightarrow_\pi M'$ est obtenu par la règle (*App*).

On admettra dans la suite que le résultat est vrai dans le cas des autres règles aussi.

Question 6 Pour un terme admissible M , on obtient le terme admissible $\text{Unfold}_\pi(M)$ en remplaçant chaque sous-terme de la forme $f(N_1, N_2, \dots, N_n)$ dans M par $P[x_1 := N_1, x_2 := N_2, \dots, x_n := N_n]$, où $\langle (x_1, x_2, \dots, x_n), P \rangle$ est la définition $\pi(f)$ de f .

On peut montrer (et on l'admettra) que pour tout environnement ρ , $\llbracket \text{Unfold}_\pi(M) \rrbracket \rho = \llbracket M \rrbracket (F_\pi(\rho))$.

On pose ρ_\perp l'environnement qui à toute variable et à tout nom de fonction associe le plus petit élément du dcpo associé.

Montrer que pour tout terme admissible M de type `int` tel que $\llbracket M \rrbracket \rho_\pi$ est un entier $m \in \mathbb{Z}$ (donc différent de \perp), il existe un entier $n \in \mathbb{N}$ tel que $\llbracket \text{Unfold}_\pi^n(M) \rrbracket \rho_\perp = m$.

À titre d'indication, c'est un argument similaire à l'argument qui est au cœur de la preuve d'adéquation du langage `Imp` du cours.

Question 7 La conclusion de la question **Question 6** est-elle encore vraie lorsque M est de type **I**? Autrement dit, supposons que $\llbracket M \rrbracket_{\rho_{\pi}}$ soit une liste infinie $\ell = n_1 : n_2 : \dots : n_k : \dots$ d'entiers, tous différents de \perp : existe-t-il un entier $n \in \mathbb{N}$ tel que $\llbracket \text{Unfold}_{\pi}^n(M) \rrbracket_{\rho_{\perp}} = \ell$?

Fournir une justification rapide si c'est le cas, un contre-exemple sinon.

Question 8 On définit \rightarrow_{\perp} comme étant égale à la relation \rightarrow_{π} dans le cas où le programme π est vide. De façon équivalente, $M \rightarrow_{\perp} M'$ si et seulement si M se réécrit en M' en une étape par l'une des règles de la figure 1 autre que (*App*).

On souhaite montrer que pour tout terme N admissible de type **int** telle que $\llbracket N \rrbracket_{\rho_{\perp}} \neq \perp$, il existe un $n \in \mathbb{Z}$ tel que $N \rightarrow_{\perp}^* \dot{n}$. Ceci n'étant pas démontrable directement, nous allons démontrer simultanément que pour tout terme admissible N :

- (i) si N est de type **int** et $\llbracket N \rrbracket_{\rho_{\perp}} \neq \perp$, alors il existe un entier $n \in \mathbb{Z}$ tel que $N \rightarrow_{\perp}^* \dot{n}$;
- (ii) si N est de type **I**, alors pour tout entier $m \in \mathbb{N}$ tel que $\llbracket \text{hd } \text{tl}^m N \rrbracket_{\rho_{\perp}} \neq \perp$, il existe un entier $n \in \mathbb{Z}$ tel que $\text{hd } \text{tl}^m N \rightarrow_{\perp}^* \dot{n}$.

Les sous-questions auxquelles vous avez à répondre sont les suivantes.

- (a) Sur quoi se fait la récurrence ?
- (b) Traitez du cas où N est de la forme $P : Q$. Les autres cas seront admis.

Nous allons nous arrêter là. Nous ne sommes plus très loin de démontrer l'adéquation (si M est un terme admissible de type **int**, et que $\llbracket M \rrbracket_{\rho_{\pi}} = m$ avec $m \neq \perp$, alors $M \rightarrow_{\pi}^* \dot{m}$). Il ne resterait plus qu'à démontrer que pour tout π -dépliage M^* de M , si $M^* \rightarrow_{\perp}^* \dot{m}$, alors $M \rightarrow_{\pi}^* \dot{m}$. Un π -dépliage de M est obtenu à partir de M en itérant la procédure consistant à remplacer un sous-terme de la forme $f(N_1, N_2, \dots, N_n)$ dans M par $P[x_1 := N_1, x_2 := N_2, \dots, x_n := N_n]$, où $\langle (x_1, x_2, \dots, x_n), P \rangle$ est la définition $\pi(f)$ de f . Par la question **Question 6**, si $\llbracket M \rrbracket_{\rho_{\pi}} = m$ avec $m \neq \perp$, $\llbracket \text{Unfold}_{\pi}^n(M) \rrbracket_{\rho_{\perp}} = m$ pour un certain n , donc $\text{Unfold}_{\pi}^n(M) \rightarrow_{\perp}^* \dot{m}$ par la question **Question 8**. Or $\text{Unfold}_{\pi}^n(M)$ est un dépliage de M , ce qui permet de conclure.