

Examen Programmation I (2017-18)

On insistera sur la correction et la clarté des réponses.

1 Théorie des domaines

Vous avez vu en TD qu'un dcpo X est *algébrique* si et seulement si pour tout $z \in X$, il existe une famille dirigée $(z_i)_{i \in I}$ d'éléments *finis* telle que $\sup_{i \in I} z_i = z$. Un élément z est *fini* si et seulement si pour toute famille dirigée $(y_j)_{j \in J}$ telle que $z \leq \sup_{j \in J} y_j$, alors il existe un $j \in J$ tel que $z \leq y_j$.

Un *inf-demi-treillis* est un ensemble ordonné dans lequel tout couple d'éléments x, y a une borne inférieure, que l'on notera $x \wedge y$.

On rappelle qu'un dcpo X est *conjonctif* si et seulement si c'est un inf-demi-treillis et pour tout $x \in X$, la fonction $y \mapsto x \wedge y$ est Scott-continue. Dans ce cas, la fonction $(x, y) \mapsto x \wedge y$ est Scott-continue.

1. (7 lignes) Montrer que tout inf-demi-treillis qui est aussi un dcpo algébrique est conjonctif. Indication : si z est fini et inférieur ou égal à $x \wedge \sup_{j \in J} y_j$, pourquoi z est-il inférieur ou égal à $\sup_{j \in J} (x \wedge y_j)$?

On montre l'indication. Soit z fini, $z \leq x \wedge \sup_{j \in J} y_j$. Alors $z \leq x$ et $z \leq \sup_{j \in J} y_j$, donc $z \leq y_j$ pour un certain $j \in J$. Ceci implique $z \leq x \wedge y_j \leq \sup_{j \in J} (x \wedge y_j)$.

Soit $(z_i)_{i \in I}$ une famille dirigée d'éléments finis de sup égal à $x \wedge \sup_{j \in J} y_j$. Ceci existe par algébricité. L'indication implique qu'ils sont tous $\leq \sup_{j \in J} (x \wedge y_j)$. On prend le sup et on obtient $x \wedge \sup_{j \in J} y_j \leq \sup_{j \in J} (x \wedge y_j)$.

En présence de monotonie, ceci suffit pour montrer que $y \mapsto x \wedge y$ est Scott-continue.

2 Real PCF⁻

On redonne les sémantiques dénotationnelle et opérationnelle de Real PCF⁻ dans l'annexe (Section 4). On rappelle que, pour tout $u : \tau$, $\llbracket u \rrbracket$ est une fonction bien définie, Scott-continue de $Env \stackrel{\text{def}}{=} \prod_{x_\sigma \text{ variable}} \llbracket \sigma \rrbracket$ vers $\llbracket \tau \rrbracket$.

2. (14 lignes) Montrer que la construction $u > 0$ de RealPCF⁻ est redondante. De façon explicite, proposer une définition d'une expression Real PCF⁻ **nonzero**, de type $\mathbb{I} \rightarrow \mathbf{unit}$, qui n'utilise pas d'expression de la forme $u > 0$, et dont la sémantique $\llbracket \mathbf{nonzero} \rrbracket \rho$

soit la fonction qui à 0 associe \perp et à tout $a \in \mathcal{I}$ non nul associe \top . On demande la preuve de cette affirmation.

```

rec nonzero =fn mI.
    pif m > 1/2 then *
    else nonzero(tl0m)

```

Sa sémantique est le plus petit point fixe de la fonction F qui à $\phi \in [\mathcal{I} \rightarrow \mathbb{S}]$ associe la fonction qui à $a \in \mathcal{I}$ associe \top si $a > 1/2$, $\varphi(\max(2a, 1)) = \varphi(2a)$ sinon. (Le \wedge est trivial ici.)

Les itérés de Kleene sont $\phi_0 = \perp$, puis ϕ_1 qui associe \top exactement aux $a > 1/2$, puis ϕ_2 qui associe \top exactement aux a tels que $a > 1/2$ ou $2a > 1/2$ (c'est-à-dire aux $a > 1/4$).

Par récurrence sur n , on voit que ϕ_n associe \top exactement aux $a > 1/2^n$. En effet, ϕ_{n+1} envoie tout $a > 1/2$ vers \top , et tout $a \leq 1/2$ vers $\phi_n(2a)$, c'est-à-dire vers \top si $2a > 1/2^n$ (i.e., $a > 1/2^{n+1}$) et vers \perp sinon.

Le plus petit point fixe envoie donc toujours 0 vers \perp , mais tout nombre $a > 0$ vers \top puisqu'il existe un $n \in \mathbb{N}$ à partir duquel $a > 1/2^n$.

Une autre solution est :

```

rec nonzero =fn mI. 1.mI > 1/2

```

qui était bien plus simple à justifier.

- (3 lignes) Montrer que l'avant-avant-dernière règle de la sémantique opérationnelle est correcte, au sens où $\llbracket \text{pif } u \text{ then } v \text{ else } * \rrbracket \rho = \llbracket v \rrbracket \rho$ pour tout $\rho \in \text{Env}$.

*Si $\llbracket u \rrbracket \rho > 1/2$, le côté gauche vaut $\llbracket v \rrbracket \rho$. Sinon, il vaut $\llbracket v \rrbracket \rho \wedge \llbracket * \rrbracket \rho = \llbracket v \rrbracket \rho$ puisque $\llbracket * \rrbracket \rho = \top$ est le plus grand élément de \mathbb{S} (et que tout se passe dans \mathbb{S} vu les contraintes de typage).*

- (5 lignes) On aimerait montrer l'adéquation. Bien entendu, c'est trop difficile et je ne le demande pas. Mais voici un argument qui pourrait faire partie de la preuve. On considère un programme Real PCF⁻ de la forme **letrec** $x_\sigma = u$ **in** v , de type **unit**. Montrer que si $\llbracket \text{letrec } x_\sigma = u \text{ in } v \rrbracket \rho \neq \perp$, alors il existe un entier $n \in \mathbb{N}$ tel que :

$$\llbracket \text{letrec } x_\sigma = u \text{ in } v \rrbracket \rho = g(f^n(\perp)),$$

où l'on a utilisé les abréviations $g(V) = \llbracket v \rrbracket (\rho[x_\sigma \mapsto V])$ et $f(V) = \llbracket u \rrbracket (\rho[x_\sigma \mapsto V])$. (Le \perp en argument de f^n est celui de $\llbracket \sigma \rrbracket$.) Ceci exprime qu'une définition récursive (de x_σ) utilisée dans un calcul terminant (v) de type **unit** ne sera « dépliée » que n fois.

Par définition, $\llbracket \text{letrec } x_\sigma = u \text{ in } v \rrbracket \rho = g(\text{lfp } f)$. En utilisant la formule de Kleene, et la Scott-continuité de g , ceci vaut $\sup_{n \in \mathbb{N}} g(f^n(\perp))$. Le dcpo $\llbracket \text{unit} \rrbracket = \mathbb{S}$ est plat, donc ce sup est atteint pour un certain n .

Attention : on doit utiliser la Scott-continuité de g . Il n'existe aucun $n \in \mathbb{N}$ tel que $f^n(\perp) = \text{lfp } f$ en général, comme le montre la question suivante.

- (2 lignes, avec contre-exemple) Pourquoi l'argument de la question précédente ne fonctionne-t-il pas si **letrec** $x_\sigma = u$ **in** v est de type **I** ?

I n'a pas la propriété de chaîne croissante. Par exemple, la définition de $\hat{1}$ produit une telle chaîne croissante infinie.

6. (5 lignes) On rappelle que $\dot{0} \stackrel{\text{def}}{=} \text{letrec } x_{\text{I}} = 0.x_{\text{I}} \text{ in } x_{\text{I}}$. Montrer qu'il n'existe aucune dérivation dans la sémantique opérationnelle de :

$$\text{tl}_0(\text{pif } \dot{0} > 1/2 \text{ then } 1.\dot{0} \text{ else } 0.1.1.\dot{0}) > 1/2 \rightarrow^* *.$$

On pourra poser $Z \stackrel{\text{def}}{=} \text{letrec } x_{\text{I}} = 0.x_{\text{I}} \text{ in } 0.x_{\text{I}}$.

Les expressions $1.\dot{0}$ et $0.1.1.\dot{0}$ sont en forme normale (ne se réécrivent pas) parce $1._$ et $0._$ ne sont pas des contextes.

En fait, on ne peut que commencer par réécrire $\dot{0} > 1/2$ en $Z > 1/2$, puis en $0.Z > 1/2$, ce qui donne $\text{tl}_0(\text{pif } 0.Z > 1/2 \text{ then } 1.\dot{0} \text{ else } 0.1.1.\dot{0}) > 1/2$. Mais il n'y a plus aucune règle applicable à cette expression.

7. (8 lignes) Que peut-on en conclure pour l'adéquation au type `unit`? Justifier.

Elle échoue. En effet, pour n'importe quel environnement ρ , comme $\llbracket 0.Z > 1/2 \rrbracket \rho = \text{add}_0(0) = 0$,

$$\begin{aligned} \llbracket \text{tl}_0(\text{pif } 0.Z > 1/2 \text{ then } 1.\dot{0} \text{ else } 0.1.1.\dot{0}) \rrbracket \rho &= \text{rem}_0(\llbracket 1.\dot{0} \rrbracket \rho \wedge \llbracket 0.1.1.\dot{0} \rrbracket \rho) \\ &= \text{rem}_0(\text{add}_1(0) \wedge \text{add}_0(\text{add}_1(\text{add}_1(0)))) \\ &= \text{rem}_0(1/2 \wedge 3/8) = \text{rem}_0(3/8) = 3/4 \end{aligned}$$

*Donc $\llbracket \text{tl}_0(\text{pif } 0.Z > 1/2 \text{ then } 1.\dot{0} \text{ else } 0.1.1.\dot{0}) > 1/2 \rrbracket \rho = *$, mais nous venons de voir que la sémantique opérationnelle ne progresse pas suffisamment loin pour atteindre $*$.*

8. (9 lignes) Des suggestions pour compléter la sémantique opérationnelle ?

On peut déjà ajouter les règles :

$$\begin{aligned} \text{tl}_a(\text{pif } u \text{ then } v \text{ else } w) &\rightarrow \text{pif } u \text{ then } \text{tl}_a v \text{ else } \text{tl}_a w \\ (\text{pif } u \text{ then } v \text{ else } w) > 1/2 &\rightarrow \text{pif } u \text{ then } v > 1/2 \text{ else } w > 1/2 \\ (\text{pif } u \text{ then } v \text{ else } w) > 0 &\rightarrow \text{pif } u \text{ then } v > 0 \text{ else } w > 0 \end{aligned}$$

pour $a \in \{0, 1\}$. La troisième forme n'est pas essentielle pour l'exemple, mais on voit bien qu'elle sera nécessaire en générale. On pourrait penser aussi à rajouter des règles comme $(\text{pif } u \text{ then } v \text{ else } w)t \rightarrow \text{pif } u \text{ then } vt \text{ else } wt$, mais ce n'est pas nécessaire, car avec l'adéquation au type `unit`, les fonctions ne jouent que peu de rôle.

3 Typage

On suppose dorénavant qu'une même variable Real PCF⁻ est toujours étiquetée avec le même type : si l'on rencontre x_σ et x_τ , alors $\sigma = \tau$. Ceci revient à dire que le nom x des variables suffit à les distinguer.

On considère le langage Real PCF⁻⁻, qui est juste Real PCF⁻ mais sans aucun indice de type. Par exemple, `fn $x.u$` et `letrec $x = u$ in v` sont les expressions Real PCF⁻⁻ correspondant à `fn $x_\sigma.u$` et à `letrec $x_\sigma = u$ in v` , respectivement.

Formellement, notons E la fonction d'effacement des types, définie par $E(\mathbf{letrec} x_\sigma = u \mathbf{in} v) \stackrel{\text{def}}{=} \mathbf{letrec} x = E(u) \mathbf{in} E(v)$, $E(\mathbf{fn} x_\sigma.u) \stackrel{\text{def}}{=} \mathbf{fn} x.E(u)$, etc.

On dira qu'une expression Real PCF⁻⁻ u , est *typable*, de type τ , si et seulement s'il existe une expression Real PCF⁻ u' , de type τ , telle que $E(u') = u$.

9. (1 ligne) Toutes les expressions Real PCF⁻⁻ sont-elles typables? Justifier.

Non. Par exemple, xx ne l'est pas, ou bien $ + \dot{0}$.*

10. (1 ligne) Le type d'une expression Real PCF⁻⁻ typable est-il unique? Justifier.

Non, par exemple $\mathbf{fn} x.x$ a tous les types $\sigma \rightarrow \sigma$.

On souhaite construire un algorithme de typage de Real PCF⁻⁻ à la Hindley. On supposera les expressions rectifiées (toute variable est liée à au plus un endroit). Etant donné une expression Real PCF⁻⁻ u_0 , on crée une variable de type α_t pour chaque sous-expression t de u_0 , et un ensemble d'équations à résoudre.

Par exemple, pour $t = uv$, on créera l'équation $\alpha_t = \alpha_u \rightarrow \alpha_{uv}$.

11. (1 ligne) Quelles équations créera-t-on pour $t = \mathbf{letrec} x = u \mathbf{in} v$?

$\alpha_x = \alpha_u$, et $\alpha_t = \alpha_v$.

12. (1 ligne) Et pour $t = \mathbf{pif} u \mathbf{then} v \mathbf{else} w$?

$\alpha_u = \mathbf{unit}$, $\alpha_v = \alpha_w$, et $\alpha_t = \alpha_v$.

13. (1 ligne) Une fois toutes les équations produites pour notre expression u_0 , comment pouvons-nous décider, en temps polynomial, si u_0 est typable?

Par unification.

4 Annexe

Les types de Real PCF⁻ :

$$\begin{array}{l} \sigma, \tau, \dots ::= \mathbf{unit} \\ \quad \quad \quad | \quad \mathbf{I} \\ \quad \quad \quad | \quad \sigma \rightarrow \tau \end{array}$$

$\mathbb{S} = \{\perp, \top\}$ avec $\perp < \top$. $\mathcal{I} = [0, 1]$, avec l'ordre usuel.

$$\llbracket \mathbf{unit} \rrbracket = \mathbb{S} \quad \llbracket \mathbf{I} \rrbracket = \mathcal{I} \quad \llbracket \sigma \rightarrow \tau \rrbracket = \llbracket \sigma \rrbracket \rightarrow \llbracket \tau \rrbracket.$$

$$\begin{aligned}
\llbracket x_\tau \rrbracket \rho &= \rho(x_\tau) \\
\llbracket uv \rrbracket \rho &= \llbracket u \rrbracket \rho (\llbracket v \rrbracket \rho) \\
\llbracket \text{fn } x_\sigma . u \rrbracket \rho &= (V \in \llbracket \sigma \rrbracket \mapsto \llbracket u \rrbracket (\rho[x_\sigma \mapsto V])) \\
\llbracket \text{letrec } x_\sigma = u \text{ in } v \rrbracket &= \llbracket v \rrbracket (\rho[x_\sigma \mapsto \llbracket \text{rec } x_\sigma = u \rrbracket \rho]) \\
\llbracket \text{rec } x_\sigma = u \rrbracket \rho &= \text{lfp}(V \in \llbracket \sigma \rrbracket \mapsto \llbracket u \rrbracket (\rho[x_\sigma \mapsto V])) \\
\llbracket 0.u \rrbracket \rho &= \text{add}_0(\llbracket u \rrbracket \rho) & \llbracket 1.u \rrbracket \rho &= \text{add}_1(\llbracket u \rrbracket \rho) \\
\llbracket \text{tl}_0 u \rrbracket \rho &= \text{rem}_0(\llbracket u \rrbracket \rho) & \llbracket \text{tl}_1 u \rrbracket \rho &= \text{rem}_1(\llbracket u \rrbracket \rho) \\
\llbracket \text{pif } u \text{ then } v \text{ else } w : \tau \rrbracket \rho &= \begin{cases} \llbracket v \rrbracket \rho & \text{si } \llbracket u \rrbracket \rho = \top \\ \llbracket v \rrbracket \rho \wedge \llbracket w \rrbracket \rho & \text{si } \llbracket u \rrbracket \rho = \perp \end{cases} \\
\llbracket u > 1/2 \rrbracket \rho &= \begin{cases} \top & \text{si } \llbracket u \rrbracket \rho > 1/2 \\ \perp & \text{sinon} \end{cases} & \llbracket u > 0 \rrbracket \rho &= \begin{cases} \top & \text{si } \llbracket u \rrbracket \rho > 0 \\ \perp & \text{sinon} \end{cases} \\
\llbracket * \rrbracket \rho &= \top,
\end{aligned}$$

où $V \in X \mapsto f(V)$ désigne la fonction qui à tout V de X associe $f(V)$, et où :

$$\begin{aligned}
\text{add}_0(a) &= a/2 & \text{add}_1(a) &= (a + 1)/2 \\
\text{rem}_0(a) &= \min(2a, 1) & \text{rem}_1(a) &= \max(2a - 1, 0)
\end{aligned}$$

Contextes (contraintes de types omises) :

$$\begin{aligned}
\mathcal{C} ::= & _ \\
& | \mathcal{C}v \\
& | \text{tl}_0 \mathcal{C} \\
& | \text{tl}_1 \mathcal{C} \\
& | \mathcal{C} > 1/2 \\
& | \mathcal{C} > 0 \\
& | \text{pif } \mathcal{C} \text{ then } v \text{ else } w \\
& | \text{pif } u \text{ then } \mathcal{C} \text{ else } w \\
& | \text{pif } u \text{ then } v \text{ else } \mathcal{C}
\end{aligned}$$

Sémantique opérationnelle. On n'applique une règle que sous un contexte \mathcal{C} de la forme ci-dessus, i.e., $u \rightarrow v$ si et seulement si $u = \mathcal{C}[\ell]$ et $v = \mathcal{C}[r]$, où \mathcal{C} est un contexte (les types étant

respectés), et $\ell \rightarrow r$ est l'une des règles ci-dessous.

$$\begin{array}{l}
(\text{fn } x_\sigma.u)v \rightarrow u[x_\sigma := v] \\
\text{letrec } x_\sigma = u \text{ in } v \rightarrow v[x_\sigma := \text{letrec } x_\sigma = u \text{ in } u] \\
\text{tl}_a(a.u) \rightarrow u \qquad (a \in \{0, 1\}) \\
\text{tl}_0(1.u) \rightarrow \dot{1} \\
\text{tl}_1(0.u) \rightarrow \dot{0} \\
(1.u) > 1/2 \rightarrow u > 0 \\
(1.u) > 0 \rightarrow * \\
(0.u) > 0 \rightarrow u > 0 \\
\text{pif } * \text{ then } v \text{ else } w \rightarrow v \\
\text{pif } u \text{ then } v \text{ else } * \rightarrow v \\
\text{pif } u \text{ then } 0.v \text{ else } 1.w \rightarrow 0.v \\
\text{pif } u \text{ then } a.v \text{ else } a.w \rightarrow a.(\text{pif } u \text{ then } v \text{ else } w) \qquad (a \in \{0, 1\})
\end{array}$$