Analyse des Programmes et Sémantique (6)

Prof[r] Jacques Malenfant

(cours de J.Malenfant modifié et enseigné en 2013 par Basile Starynkévitch)

janvier-avril 2013

MI030 - APS

Cours (en M1) du Professeur Jacques Malenfant
http://pagesperso-systeme.lip6.fr/Jacques.Malenfant/ Professeur en informatique au Laboratoire d'Informatique de Paris 6

Enseigné en 2013 par Basile Starynkevitch
http://starynkevitch.net/Basile/
basile@starynkevitch.net & basile.starynkevitch@cea.fr
ingénieur chercheur au CEA, LIST - http://www-list.cea.fr/
travaille sur gcc-melt.org

♠ Nota Bene

Les transparents à fond rose (pages numérotées ♠) et les mots ♠ signalés ainsi ♣ sont de Basile Starynkevitch (dont les opinions n'engagent que lui) ♠

La plupart des transparents sont [recopiés de ceux] de J.Malenfant 2012, que je remercie. Ces transparents sont disponibles sous http://starynkevitch.net/Basile/

Rappels

On a vu :

- Les domaines (avec un CPO = “ordre partiel complet”) où $x \preccurlyeq y$ est interprété comme le fait que y contient plus d’information que x
- Un catalogue de domaines discrets prédéfinis : un ensemble S (i.e. $\mathbb{N}$, etc...) de valeurs discrètes augmenté d’un $\perp_S$
- les domaines produits
- les domaines sommes (unions disjointes)
- les domaines séquences et les fonctions dessus

Domaines de fonctions I

Définition (fonction totale et partielle)

Une fonction $f : A \to B$ est *totale* si elle est telle que $f(x) \in B$ est définie pour tout $x \in A$, sinon elle est dite *partielle*.

Définition (domaines de fonctions)

Soient **A** et **B** des domaines munis d'ordre partiels complets, alors on peut définir le domaine $\mathbf{Fun}(\mathbf{A},\mathbf{B})$ des fonctions totales de **A** dans **B** muni de l'ordre partiel complet $\preccurlyeq_{\mathbf{Fun}(\mathbf{A},\mathbf{B})}$ tel que

$$\forall f, g \in \mathbf{Fun}(\mathbf{A},\mathbf{B}), f \preccurlyeq_{\mathbf{Fun}(\mathbf{A},\mathbf{B})} g \text{ si } f(x) \preccurlyeq_{\mathbf{B}} g(x), \forall x \in \mathbf{A}$$

- Des restrictions sont nécessaires pour éviter d'admettre des fonctions dont le comportement n'est pas réalisable, comme par exemple

$$f(x) = \begin{cases} 1 & \text{si } g(x) = \bot \\ 0 & \text{sinon} \end{cases}$$

qui n'est pas calculable puisque $\bot$ représente l'indéfinition de g ou encore sa non-terminaison.

Domaines de fonctions II

Définition (fonction monotone)

Une fonction f est *monotone* si $x \preccurlyeq_{\mathbf{A}} y \implies f(x) \preccurlyeq_{\mathbf{B}} f(y), \forall x, y \in \mathbf{A}$.

♠ on pourrait dire que f "commute" avec les ordres $\preccurlyeq$ ♣

Définition (fonction continue)

Une fonction f est *continue* si elle préserve les plus petites bornes supérieures, c'est-à-dire que pour toute chaîne ascendante $x_1 \preccurlyeq_{\mathbf{A}} x_2 \preccurlyeq_{\mathbf{A}} \ldots$, on a $f(lub(\{x_i, i \geq 1\}) = lub(\{f(x_i), i \geq 1\})$.

♠ on pourrait dire que f "commute" avec les *lub* ♣

Théorème

La relation $\preccurlyeq_{\mathbf{Fun}(\mathbf{A},\mathbf{B})}$ sur l'ensemble des fonctions monotones et continues de $\mathbf{Fun}(\mathbf{A}, \mathbf{B})$ est un *ordre partiel complet*.

Domaines de fonctions III

♠ à propos du théorème précédent : **Fun**(**A**, **B**) est un *ordre partiel complet*. ♣

- On peut montrer que la plus petite borne supérieure d'une chaîne ascendante de fonctions monotones et continues est une fonction monotone et continue.
- On peut aussi montrer que la composition de fonctions monotones et continues donne une fonction monotone et continue.
- Enfin, on peut montrer que les fonctions sur les domaines vus précédemment sont monotones et continues.

Retour sur les fonctions récursives

Soit la fonction

```
f(n) = if n = 0 then 5 else
         if n = 1 then f(n+2) else f(n-2)
```

- Quelle fonction peut être le dénotation de cette équation ?
- S'il y en a plus d'une, laquelle devrait-on choisir pour être *la* dénotation de cette équation ?
- Nous allons voir comment la calculer...

Définition de la fonctionnelle

Soit la fonctionnelle F telle que

$$F : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N})$$
$$F = \lambda f.\lambda n.\textbf{if } n = 0 \textbf{ then } 5 \textbf{ else if } n = 1 \textbf{ then } (f\ (+\ n\ 2))\ \textbf{else}\ (f\ (-\ n\ 2))$$

- Une fonction f satisfait l'équation originale si et seulement si elle est un point fixe de F, c'est-à-dire $(F\ f) = f$.
- Comment calculer cette fonction f ?

Théorème du point fixe pour les fonctions

Théorème (point fixe)

Soit $\mathbf{D}$ un domaine muni d'un ordre partiel complet $\preccurlyeq_{\mathbf{D}}$ et $g : \mathbf{D} \to \mathbf{D}$ une fonction monotone et continue sur $\mathbf{D}$, alors g a un plus petit point fixe dans $\mathbf{D}$.

Corollaire

Toute fonctionnelle continue $F : (\mathbf{A} \to \mathbf{B}) \to (\mathbf{A} \to \mathbf{B})$, où $\mathbf{A}$ et $\mathbf{B}$ sont des domaines, a un plus petit point fixe $f_{pppf} : \mathbf{A} \to \mathbf{B}$ qui peut être pris comme la dénotation de la définition récursive correspondant à F.

Calcul du plus petit point fixe

- Le calcul du point fixe va consister à poser une définition initiale de F, puis à itérer sur la fonction obtenue en résultat pour la rendre de mieux en mieux définie...
- Pour la fonctionnelle précédente, on va calculer la chaine ascendante $\bot \preccurlyeq F(\bot) \preccurlyeq F^2(\bot) \preccurlyeq F^3(\bot) \preccurlyeq \ldots$ telle que :

$$
\begin{aligned}
f_0 &= \lambda n.\bot \\
f_1 &= (F\ f_0) \\
&= \lambda n.\textbf{if } n = 0 \textbf{ then } 5 \textbf{ else if } n = 1 \textbf{ then } (f_0\ (+\ n\ 2)) \textbf{ else } (f_0\ (-\ n\ 2)) \\
&= \lambda n.\textbf{if } n = 0 \textbf{ then } 5 \textbf{ else } \bot \\
f_2 &= \lambda n.\textbf{if } n = 0 \textbf{ then } 5 \textbf{ else if } n = 1 \textbf{ then } (f_1\ (+\ n\ 2)) \textbf{ else } (f_1\ (-\ n\ 2)) \\
&= \lambda n.\textbf{if } n = 0 \textbf{ then } 5 \textbf{ else if } n = 1 \textbf{ then } \bot \textbf{ else if } n = 2 \textbf{ then } 5 \textbf{ else} \bot \\
\ldots &= \ldots
\end{aligned}
$$

♠ Ça fait penser à une “interprétation” progressive et partielle de `£` ♣

Observation

- Conceptuellement, qu'a-t'on fait ?
- On a d'abord construit la fonction f_0 qui n'est pas récursive et qui est indéfinie partout puisqu'elle a pour résultat $\perp$ pour toute entrée.
- On a ensuite construit la fonction f_1 qui répond 5 pour $n = 0$ puis appelle f_0 les autres cas.
- Et on a poursuivi pour la fonction f_2 qui sait répondre pour $n = 0$ et $n = 2$, mais appelle f_1 pour les autres cas.
- Et ainsi de suite, chaque fonction étant mieux définie que la précédente car capable de répondre pour plus de valeurs de n.
- Le point fixe étant la fonction totale f_∞ qui sait répondre pour tout n en utilisant les fonctions partielles qui la précèdent dans la chaine.

Récapitulons les étapes et reconnectons...

1. Définition récursive `f(...) = ... f ...`
2. Définition de la fonctionnelle $F\ f = \lambda \ldots f \ldots$
3. Calcul du plus petit point fixe de la fonctionnelle F, ce que l'on définit comme le résultat d'un opérateur

$$\textbf{fix} : (\mathbf{D} \to \mathbf{D}) \to \mathbf{D} \quad \text{où } \textbf{fix} = \lambda F.lub(\{F\ (\bot) | i \geq 0\})$$

avec la propriété

$$(F\ (\textbf{fix}\ F)) = (\textbf{fix}\ F)$$

4. **fix** est bel et bien l'opérateur Y du λ-calcul.
5. En réalité, il y a plusieurs opérateurs de point fixe :

$$\textbf{fix}_1 \equiv \lambda f.(\lambda x.(f\ (x\ x)))\ (\lambda x.(f\ (x\ x)))$$
$$\textbf{fix}_2 \equiv \lambda f.((\lambda x.(f\ (\lambda y.((x\ x)\ y))))\ (\lambda x.(f\ (\lambda y.((x\ x)\ y)))))$$

6. $\textbf{fix}_1$ est l'opérateur que nous avons étudié plus tôt, et qui fonctionne bien pour l'ordre normal.
7. $\textbf{fix}_2$ est un autre opérateur qui fonctionne pour l'ordre applicatif.

Revenons sur l'exemple du `while`

- Nous avons défini la dénotation du `while` de notre mini-langage impératif comme étant :
  $\mathit{execute}[\![\texttt{while}\ e\ i]\!]\rho\sigma = (\mathit{loop}\ \sigma)$
  where $\mathit{loop} = \mathbf{fix}\ \lambda f.\lambda\sigma.$
  if $\mathit{out}\mathbf{T}(\mathit{eval}[\![e]\!]\rho\sigma)$ **then** $(f\ \mathit{execute}[\![i]\!]\rho\sigma)$ **else** σ
- Que représente *loop* ?

Le point fixe pour *loop*

Si $loop_0 = \lambda\sigma.\bot$
et $loop_{n+1} = \lambda\sigma.\textbf{if } out\mathbf{T}(eval[\![e]\!]\rho\sigma) \textbf{ then } (loop_n\ execute[\![i]\!]\rho\sigma) \textbf{ else } \sigma$,
alors on peut voir intuitivement que :

- $loop_0$ représente les cas où la boucle ne s'exécute pas,
- $loop_1$ représente les cas où le test est faux dès le départ et retourne σ inchangée, ou alors n'est pas définie,
- $loop_2$ représente les cas où le test est vrai, puis où on exécute le corps de la boucle une fois pour constater que le test est maintenant faux, ou alors n'est pas définie,
- et ainsi de suite avec $loop_n$ qui représente le cas où le test est vrai $n-1$ fois puis faux, ou alors n'est pas définie.

Intuitivement, on voit que le point fixe de ce processus va bien donner la fonction $loop_\infty$ qui peut exécuter le corps de la boucle autant de fois que nécessaire pour rendre le test faux et alors s'arrêter.
On y reviendra en TD...