Programmation 1: sémantique, leçon 2

Jean Goubault-Larrecq
ENS Cachan
goubault@lsv.ens-cachan.fr

20 décembre 2022

1 Sémantique dénotationnelle, points fixes et dcpos

1.1 Points fixes et boucles

L'un des problèmes centraux en sémantique est de donner une sémantique aux *boucles*. Il semble intuitif que `while (b) e` devrait faire la même chose que le programme

```
if (b)
  { e; while (b) e }
```

qui teste d'abord si `b` est vrai, et si c'est le cas exécute `e` puis la boucle proprement dite. (On dit qu'on a *déroulé la boucle* un coup.) Ceci nous avait posé quelques problèmes dans la sémantique dénotationnelle de IMP, où nous souhaitions que l'équation suivante soit vraie :

$$[\![\texttt{while (b) e}]\!] = [\![\texttt{if (b) \{e; while (b) e \}}]\!]$$

Nous avions ensuite opté pour l'étude de sémantiques opérationnelles, qui évitaient le problème. Nous avions aussi provisoirement conclu que la principale question en sémantique *dénotationnelle* était de définir un opérateur lfp qui à toute fonctionnelle dans un certain domaine associait un de ses points fixes.

Le problème de l'existence de points fixes se pose aussi dans la définition de fonctions récursives. Considérons le calcul de factorielle 7 en Caml :

```
let rec fact n =
        if n=0
           then 1
        else n * fact (n-1)
in fact 7;;
```

On voit que `fact` est une fonction qui est définie en termes d'elle-même : la définition de `fact`, après le premier signe =, fait appel à `fact` (que l'on applique à `n-1`). (Le fait que `fact` soit définie en termes d'elle-même est la raison d'être du mot-clé `rec`, qui instruit le compilateur Caml de ce fait.) C'est peut-être un peu plus clair si l'on réécrit l'expression Caml ci-dessus en :

```
let rec fact =
        fun n -> if n=0
                    then 1
                 else n * fact (n-1)
in fact 7;;
```

où la construction `fun n ->` [...] dénote la fonction qui à `n` associe [...]. Remplaçons `fact` par l'inconnue x dans sa définition, et soit F la fonction qui au programme $\underline{x}$ associe le programme

```
fun n -> if n=0
            then 1
         else n * x (n-1)
```

On a donc défini `fact` comme étant un point fixe de F.

Les deux exemples des boucles `while` et des fonctions récursives ne sont pas si éloignés qu'il y paraît. La boucle `while (b) e` peut en effet se coder en CaML sous la forme :

```
let rec boucle () =
        if (b)
           then (e; boucle ())
        else ()
in
   boucle ();;
```

Rappelez-vous comment on calcule le point fixe d'une fonction contractante F de $\mathbb{R}$ dans $\mathbb{R}$: partant de n'importe quel point x_0, on calcule les itérés $F^n(x_0)$ (où $F^0(x) = x$, $F^{n+1}(x) = F(F^n(x))$), et la limite est l'unique point fixe de F. Nous ne pourrons pas supposer que nous sommes dans un espace métrique complet dans la suite, ni que la fonction F qui à un programme x associe `if (b) { e;` x`}` est contractante, mais nous tenterons de calculer la sémantique de la boucle comme une sorte de limite des itérés $F^n(x_0)$ pour une certaine valeur x_0. Observons que cela revient à dire que la sémantique de `while (b) e` est la "limite" de

$$x_0$$
$$\texttt{if (b) \{ e;}\ x_0\}$$
$$\texttt{if (b) \{ e; if (b) \{ e;}\ x_0\}\}$$
$$\texttt{if (b) \{ e; if (b) \{ e; if (b) \{ e;}\ x_0\}\}\}$$
$$\ldots$$

Le problème de base va être de trouver des catégories de domaines de valeurs où toute fonction (raisonnable) aura au moins un point fixe.

1.2 Treillis complets et théorème de Tarski

L'une des catégories les plus simples où l'on aura un théorème du point fixe est celle des treillis complets et des fonctions monotones.

Rappelons qu'une relation d'*ordre* $\leq$ sur un ensemble X est une relation binaire réflexive ($x \leq x$), antisymétrique ($x \leq y$ et $y \leq x$ impliquent $x = y$), et transitive ($x \leq y$ et $y \leq z$ impliquent $x \leq z$). Un *ensemble ordonné* est un couple $(X, \leq)$ formé d'un ensemble X et d'une relation d'ordre $\leq$ sur X. On notera souvent X au lieu de $(X, \leq)$, par abus de langage. Une fonction $f : X \to Y$ sera dite *monotone* si et seulement si elle préserve l'ordre : si $x \leq x'$, alors $f(x) \leq f(x')$.

Rappelons aussi qu'un *majorant* y d'une partie F de X est un élément supérieur ou égal à tout élément de X : $x \leq y$ pour tout $x \in X$. Un *minorant* y est inférieur ou égal à tout élément de X : $y \leq x$ pour tout $x \in X$. La borne supérieure de F est le plus petit des majorants de F dans X, si elle existe ; elle est alors nécessairement unique. De même, la borne inférieure de F est le plus grand des minorants de F dans X, si elle existe ; elle est alors nécessairement unique.

Définition 1 (Treillis complet) *Un* treillis complet *est un ensemble ordonné $(X, \leq)$ non vide tel que toute famille $F \subseteq X$ a une borne supérieure $\bigvee F$ et une borne inférieure $\bigwedge F$.*

Si $F = \emptyset$, $\bigwedge \emptyset$ est par définition le plus grand élément de X, et sera noté $\top$; $\bigvee \emptyset$ est le plus petit élément de X, et sera noté $\bot$. On notera aussi $\bigvee_{i \in I} x_i$ la borne supérieure de la famille $(x_i)_{i \in I}$, et de même $\bigwedge_{i \in I} x_i$ sa borne inférieure. On notera aussi $x \vee y = \bigvee\{x, y\}$ et $x \wedge y = \bigwedge\{x, y\}$. Pour faire court, on dira aussi "le sup" au lieu de "la borne supérieure", et "l'inf" pour "la borne inférieure".

Exercice 1 *Un* inf-demi-treillis complet *est un ensemble ordonné $(X, \leq)$ non vide tel que toute famille $F \subseteq X$ a une borne inférieure $\bigwedge F$. Montrer que toute famille $F \subseteq X$ a une borne supérieure $\bigvee F$, et que donc tout inf-demi-treillis complet est un treillis complet. (Indication : $\bigvee F$ s'il existe est le plus petit des majorants... comment peut-on donc le construire ?)*

Exercice 2 *Montrer que tout sup-demi-treillis complet (notion que vous définirez par analogie avec la précédente) est un treillis complet.*

Exercice 3 *Soit A un ensemble quelconque, $\mathbb{P}(A)$ l'ensemble de ses parties. Montrer que $(\mathbb{P}(A), \subseteq)$ est un treillis complet.*

Exercice 4 *Par l'exercice 2, l'ensemble des ouverts $\mathcal{O}$ d'un espace topologique $(X, \mathcal{O})$ est un treillis complet : le sup d'une famille F d'ouverts est juste leur union. Quel est son inf ?*

Le point important est que toute fonction monotone d'un treillis complet dans lui-même a un point fixe :

Théorème 1 (Tarski-Knaster) *Soit $(X, \leq)$ un treillis complet. Soit $f : X \to X$ une fonction monotone. Alors f a un plus petit point fixe $\mathrm{lfp}(f)$ et un plus grand point fixe $\mathrm{gfp}(f)$. De plus l'ensemble $Fix(f)$ de tous les points fixes de f, ordonnés par $\leq$, est treillis complet.*

Avant d'en faire la démonstration, considérons l'intuition donnée à la fin de la section 1.1. Partant de $\bot$, on va calculer $f(\bot)$, $f^2(\bot)$, ..., $f^n(\bot)$, ... Le sup de cette famille, $\bigvee_{n\in\mathbb{N}} f^n(\bot)$, vérifie :

$$\begin{aligned} f(\bigvee_{n\in\mathbb{N}} f^n(\bot)) &\geq f(f^n(\bot)) \qquad \text{(pour tout } n\text{)} \\ &= f^{n+1}(\bot) \end{aligned}$$

donc $f(\bigvee_{n\in\mathbb{N}} f^n(\bot)) \geq \bigvee_{n\in\mathbb{N}} f^n(\bot)$, mais l'inégalité inverse ne tient pas en général (voir exercice 8). On pourrait utiliser cette forme d'argument en itérant f de façon transfinie, mais ceci nécessiterait que nous parlions d'ordinaux... et ce n'est pas un cours de théorie des ensembles.

Démonstration. Considérons l'ensemble $Pre(f) = \{x \in X | f(x) \leq x\}$ des *pre-points fixes* de f. D'abord remarquons que $Pre(f)$ est non vide, car $\top$ est dans $Pre(f)$; ceci n'a pas en réalité beaucoup d'importance.

Puisque X est un treillis complet, $Pre(f)$ a une borne inférieure ; appelons-la x_0. Comme x_0 est un minorant de $Pre(f)$, $x_0 \leq x$ pour tout $x \in Pre(f)$. Comme f est monotone, $f(x_0) \leq f(x)$, et comme $x \in Pre(f)$, $f(x) \leq x$: donc $f(x_0) \leq x$. Ceci étant vrai pour tout $x \in Pre(f)$, $f(x_0)$ est un minorant de $Pre(f)$. Comme x_0 est le plus grand de tous ces minorants, $f(x_0) \leq x_0$. Mais ceci, par définition, signifie que x_0 est dans $Pre(f)$, et est donc le *plus petit pre-point fixe* de f.

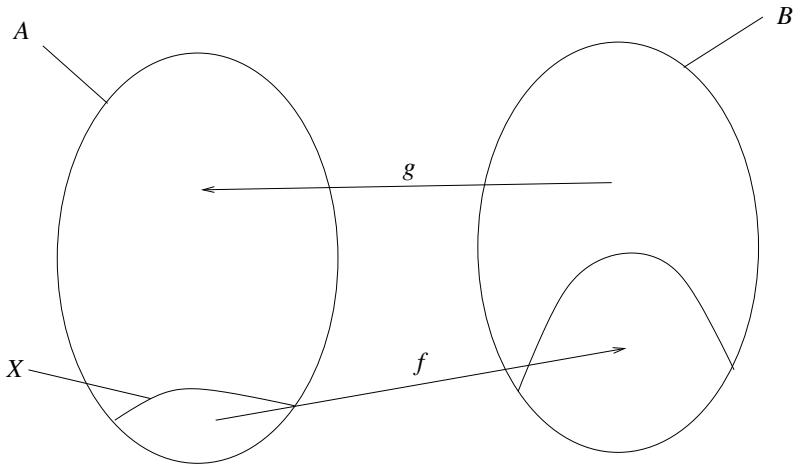
Revenons sur le fait que $f(x_0) \leq x_0$. Par monotonie de f encore, $f(f(x_0)) \leq f(x_0)$, donc $f(x_0)$ est aussi un pre-point fixe de f. Comme x_0 est le plus petit, $x_0 \leq f(x_0)$. Donc $x_0 = f(x_0)$ par antisymétrie. Ceci montre que x_0 est un point fixe de f.

C'est nécessairement le plus petit de tous les points fixes : tout autre point fixe x de f est clairement un pre-point fixe de f, et est donc supérieur ou égal au plus petit pre-point fixe x_0.

La fin de la démonstration est laissée en exercice. □

Exercice 5 *Terminez la démonstration du théorème de Knaster-Tarski. (Attention : le sup dans X d'une famille de points fixes de f n'est pas nécessairement un point fixe de f, et il faudra donc construire le sup* dans $Fix(f)$ *d'une famille de points fixes de f légèrement différemment ; indication : pensez pre-points fixes...)*

Exercice 6 *Démontrez le théorème de Cantor-Schröder-Bernstein en utilisant le théorème de Knaster-Tarski : si f est une injection de A dans B et g une injection de B dans A, alors il existe une bijection entre A et B. Indication : trouvez une partie X de A vérifiant certaines propriétés suggérées par la figure ci-dessous, en vous souvenant que $(\mathbb{P}(A), \subseteq)$ est un treillis complet (exercice 3).*



Exercice 7 *Soit $f : Y \times X \to X$ une fonction monotone, au sens où $y \leq y'$ et $x \leq x'$ impliquent $f(y,x) \leq f(y',x')$. Par le théorème de Knaster-Tarski, $\mathrm{lfp}(f(y,\_))$ existe pour tout y, où $f(y,\_)$ dénote la fonction monotone qui à x associe $f(y,x)$. Montrer que la fonction qui à y associe $\mathrm{lfp}(f(y,\_))$ est monotone de Y dans X. (Indication : utiliser la caractérisation du plus petit point fixe comme plus petit pre-point fixe.)*

Exercice 8 *Soit X l'intervalle $[0,1]$ de la droite réelle, muni de son ordre naturel $\leq$. Montrer que X est un treillis complet. Si f est une fonction monotone de X dans X, montrer que $f(\bigvee_{n\in\mathbb{N}} x_n) = \bigvee_{n\in\mathbb{N}} f(x_n)$ pour toute suite croissante $(x_n)_{n\in\mathbb{N}}$ si et seulement si f est continue à gauche sur $[0,1]$. En choisissant judicieusement f, en déduire que $f(\bigvee_{n\in\mathbb{N}} f^n(\bot))$ est en général différent de $\bigvee_{n\in\mathbb{N}} f^n(\bot)$.*

1.3 Dcpos, fonctions Scott-continues

Historiquement, Dana Scott avait proposé d'utiliser les treillis complets comme domaines de valeurs D servant à donner des sémantiques aux langages de programmation. Une notion importante découverte par D. Scott est que toutes les fonctions calculables de D dans D sont non seulement monotones mais encore *continues* (au sens de Scott, voir plus loin). Plus tard, d'autres, et notamment Gordon Plotkin, se sont aperçus que l'on pouvait se passer de treillis complets, et utiliser des ensembles ordonnés plus généraux : les *dcpos* ("directed-complete partial order").

Définition 2 (Famille dirigée, fonctions Scott-continues) *Soit $(X,\leq)$ un ensemble ordonné. Une famille $D \subseteq X$ est dite* dirigée *si et seulement si :*

- *D est non vide ;*
- *et pour tous x, y dans D, x et y ont un majorant dans D.*

Une fonction $f : X \to Y$ est Scott-continue *si et seulement si :*

- *f est monotone,*
- *et pour toute famille dirigée D qui a un sup $\bigvee D$, la famille $f(D)$ a un sup et $f(\bigvee D) = \bigvee f(D)$.*

Cette définition est une généralisation, qui s'est avérée naturelle, de ce que nous voulons vraiment représenter. Reprenons la construction du plus petit point fixe sous forme de $\bigvee_{n\in\mathbb{N}} f^n(\bot)$, construction qui avait échoué en section 1.2. La famille $(f^n(\bot))_{n\in\mathbb{N}}$ est dirigée, d'abord : c'est facile, et c'est le contenu de l'exercice 9 ci-dessous. Si cette famille a un sup $\bigvee_{n\in\mathbb{N}} f^n(\bot)$, la Scott-continuité de f est exactement l'hypothèse supplémentaire dont nous avions besoin pour montrer que $\bigvee_{n\in\mathbb{N}} f^n(\bot)$ est le plus petit point fixe de f : c'est le lemme 2.

Exercice 9 *Soit $(x_n)_{n\in\mathbb{N}}$ une suite croissante quelconque dans $(X, \leq)$. Autrement dit, $x_n \leq x_{n+1}$ pour tout $n \in \mathbb{N}$. Montrer que $(x_n)_{n\in\mathbb{N}}$ forme une famille dirigée. Si $f : X \to X$ est une fonction monotone, montrer que $(f^n(\bot))_{n\in\mathbb{N}}$ est une suite croissante, et forme donc une famille dirigée.*

Lemme 2 *Soit f une fonction Scott-continue de X dans X, et supposons que X a un plus petit élément $\bot$. Si $\bigvee_{n\in\mathbb{N}} f^n(\bot)$ existe, c'est le plus petit point fixe de f.*

Démonstration. C'est un point fixe :

$$\begin{aligned} f(\bigvee_{n\in\mathbb{N}} f^n(\bot)) &= \bigvee_{n\in\mathbb{N}} f(f^n(\bot)) \qquad \text{(par Scott-continuité)} \\ &= \bigvee_{n\in\mathbb{N}} f^{n+1}(\bot) = \bigvee_{n\in\mathbb{N}} f^{n+1}(\bot) \vee \bot \qquad \text{(trivialement)} \\ &= \bigvee_{n\in\mathbb{N}} f^n(\bot) \end{aligned}$$

C'est le plus petit : supposons en effet que x est un point fixe, et commençons par montrer que $f^n(\bot) \leq f^n(x)$ pour tout n, par récurrence sur n ; c'est évident si $n = 0$, et par la monotonie de f dans le cas récurrent. On en déduit que $\bigvee_{n\in\mathbb{N}} f^n(\bot) \leq \bigvee_{n\in\mathbb{N}} f^n(x)$, or $\bigvee_{n\in\mathbb{N}} f^n(x) = \bigvee_{n\in\mathbb{N}} x = x$ puisque $f(x) = x$. Donc $\bigvee_{n\in\mathbb{N}} f^n(\bot) \leq x$. □

L'intuition derrière la notion de Scott-continuité est qu'un programme *calcule* une valeur, et qu'en particulier une boucle devrait *calculer* sa valeur en tant que point fixe. La construction du théorème de Knaster-Tarski est non constructive. L'intérêt de la construction du lemme 2 est que, en identifiant sup et limite, le plus petit point fixe est construit comme une limite d'itérés finis $f^n(\bot)$ de la fonction f, ce qui correspond à l'intuition que nous avons donnée à la fin de la section 1.1 sur la façon dont on pouvait imaginer calculer des points fixes.

Maintenant, $\bigvee_{n\in\mathbb{N}} f^n(\bot)$ existe toujours dans un treillis complet. Mais, alors que dans un treillis complet, les sups de n'importe quelle famille existent toujours, ici nous n'avons besoin que de sups de familles dirigées.

Définition 3 (Dcpo) *Un* ordre partiel complet*, ou* dcpo *("directed-complete partial order") est un ensemble ordonné $(X, \leq)$ dans lequel toute famille dirigée a un sup. Un* dcpo pointé *est un dcpo ayant un plus petit élément $\bot$.*

Nous avons maintenant notre opérateur lfp :

Corollaire 3 *Toute fonction continue f d'un dcpo pointé X dans lui-même a un plus petit point fixe, qui est égal au sup de la famille dirigée $(f^n(\bot))_{n\in\mathbb{N}}$.*

On notera en général $\mathrm{lfp}(f) = \sup_{n\in\mathbb{N}} f^n(\bot)$.

On pourra s'interroger sur l'opportunité d'appeler une fonction f continue lorsqu'elle préserve les sups (vus comme limites). Il se trouve qu'il s'agit réellement de la notion de continuité usuelle vue en topologie générale.

Définition et lemme 4 (Topologie de Scott) *Soit $(X, \leq)$ un ensemble ordonné. Soit $\mathcal{O}$ l'ensemble des parties O de X telles que :*

- *O est* clos par le haut *: pour tout x dans O, si $x \leq y$ alors y est dans O ;*
- *et O est* inaccessible par le bas *: pour toute famille dirigée D de X dont le sup existe et est dans O, D rencontre O, c'est-à-dire $D \cap O \neq \emptyset$.*

Alors $\mathcal{O}$ est une topologie sur X, appelée la topologie de Scott. *Les éléments de $\mathcal{O}$ sont appelés les* ouverts de Scott *de X.*

La propriété d'inaccessibilité par le bas dit, dans le cas où D est une suite croissante $(x_n)_{n\in\mathbb{N}}$, que si le sup (la "limite") des x_n est dans D, alors l'un des x_n est dans D. Par contraposée, toute suite croissante hors de O a un sup (limite) hors de O, ce qui est une façon de dire que le complémentaire de O est fermé, dans un sens intuitif.

Démonstration. D'abord, la partie vide et X lui-même sont dans $\mathcal{O}$, clairement.

Ensuite, si $(O_i)_{i\in I}$ est une famille (possiblement infinie) d'ouverts de Scott, on prétend que $\bigcup_{i\in I} O_i$ est un ouvert de Scott : $\bigcup_{i\in I} O_i$ est clairement clos par le haut, et si D est une famille dirigée dont le sup existe et est dans $\bigcup_{i\in I} O_i$, alors $\bigvee D$ est dans un O_i, donc D rencontre O_i, ce qui entraîne que D rencontre $\bigcup_{i\in I} O_i$

Finalement, si O_1 et O_2 sont deux ouverts de Scott, montrons que leur intersection en est encore un. Clairement $O_1 \cap O_2$ est clos par le haut. Pour toute famille dirigée D dont le sup existe et telle que $\bigvee D$ est dans $O_1 \cap O_2$, alors $\bigvee D$ est dans O_1 et dans O_2, donc il existe deux éléments $x_1 \in D \cap O_1$ et $x_2 \in D \cap O_2$. Comme D est dirigée, x_1 et x_2 ont un majorant x dans D, qui est dans $O_1 \cap O_2$ parce que O_1 et O_2 sont clos par le haut. Donc D rencontre $O_1 \cap O_2$. □

Exercice 10 *Pour tout x dans X, notons $\downarrow x$ l'ensemble de tous les $x' \leq x$ dans X. Montrer que $\downarrow x$ est un fermé de Scott. (On rappelle qu'un* fermé *est par définition le complémentaire d'un ouvert, et non une partie non ouverte.)*

Exercice 11 *Soit f une fonction monotone de l'ensemble ordonné $(X, \leq)$ dans l'ensemble ordonné $(Y, \leq)$. Démontrer que f est Scott-continue si et seulement si f est continue pour la topologie de Scott, autrement dit si et seulement si l'image réciproque de tout ouvert de Scott est encore un ouvert de Scott. (Pour la direction seulement si, on montrera d'abord que l'image d'une famille dirigée par une fonction monotone est dirigée. Pour la direction si, plus difficile, on considérera l'ensemble F, intersection des $\downarrow y$ lorsque y parcourt l'ensemble des majorants de $f(D)$, et on démontrera que F est un fermé tel que $f^{-1}(F)$ contient D.)*

Dans la suite, on ne dira plus "Scott-continu", mais simplement "continu", l'exercice 25 montrant qu'il n'y a en fait aucune ambiguïté.

Exercice 12 *Montrez que la topologie de Scott sur $(X, \leq)$ n'est en général pas séparée (ou Hausdorff, ou T_2 : pour tous $x \neq x'$, il existe un ouvert O contenant x et un ouvert O' contenant x' d'intersection vide), en ce sens que si elle est séparée, alors tous les éléments de X sont incomparables pour $\leq$. Montrez en revanche que toute topologie de Scott est T_0, c'est-à-dire que pour tous $x \neq x'$, il existe un ouvert de Scott contenant x mais pas x', ou bien contenant x' mais pas x. (Utilisez l'exercice 10.)*

Exercice 13 *Étant donné un espace topologique $(X, \mathcal{O})$, son* préordre de spécialisation $\preceq$ *est défini par : pour tous x, x' dans X, $x \preceq x'$ si et seulement si, pour tout ouvert O contenant x, O contient x'. Montrer qu'il s'agit bien d'un préordre, c'est-à-dire d'une relation réflexive et transitive. Si de plus la topologie $\mathcal{O}$ est T_0, alors il s'agit d'une relation d'ordre.*

Exercice 14 *Montrer que l'ordre de spécialisation (cf. exercice 13) de la topologie de Scott d'un ensemble ordonné $(X, \leq)$ quelconque est l'ordre $\leq$ lui-même. (Pour l'un des deux sens de l'implication à démontrer, pensez à utiliser l'exercice 10.)*

L'exercice suivant montre en quoi les sups sont une bonne façon de parler de limites dans les dcpos.

Exercice 15 *On dit que x est une* limite *de la famille dirigée D dans l'espace topologique X si et seulement si, pour tout ouvert O contenant x, il existe y dans D tel que pour tout $z \geq y$ dans D, z est dans O. Montrer que le sup de D s'il existe est une limite de D, pour la topologie de Scott. En vous aidant de l'exercice 14, montrer que le sup de D s'il existe est en fait la* plus grande limite *de D. (On rappelle à ceux que cette formulation étonnerait que le théorème d'unicité des limites n'est valable que dans les espaces séparés...)*

Exercice 16 *Soit Y un ensemble ordonné possédant la* condition de chaîne croissante *: il n'y a pas de suite strictement croissante infinie $y_0 < y_1 < \cdots < y_n < \cdots$. Montrer que Y est un dcpo, et qu'en fait, pour tout famille dirigée $D = (y_i)_{i \in I}$, $\sup_{i \in I} y_i$ est un élément de D. (Attention à la différence entre suite croissante et famille dirigée.)*

Pour tout dcpo X, montrer que les fonctions continues de X vers Y sont exactement les fonctions monotones.

Dans la suite, nous aurons besoin de la construction d'un dcpo des fonctions continues. Montrons tout de suite que ceci a un sens. L'ordre entre fonctions de X vers Y est l'ordre *point à point*, défini par $f \leq g$ si et seulement si, pour tout $x \in X$, $f(x) \leq g(x)$. (Oui, cet ordre ne dépend que de l'ordre de Y, pas de celui de X.)

Lemme 5 *Soit X, Y deux dcpos. L'ensemble $[X \to Y]$ des fonctions continues de X vers Y, muni de l'ordre point à point, est un dcpo. Les sups dirigés sont eux aussi calculés point à point, autrement dit $(\sup_{i \in I} f_i)(x) = \sup_{i \in I}(f_i(x))$ pour toute famille dirigée $(f_i)_{i \in I}$ d'éléments de $[X \to Y]$ et pour tout $x \in X$. De plus, si Y est pointé, alors $[X \to Y]$ aussi.*

Démonstration. Soit $(f_i)_{i\in I}$ une famille dirigée d'éléments de $[X \to Y]$. Posons $f\colon X \to Y$ la fonction qui à tout $x \in X$ associe $f(x) = \sup_{i\in I} f_i(x)$. On va montrer que : (i) f est une fonction continue, et (ii) f est le sup de $(f_i)_{i\in I}$. (Donc $(\sup_{i\in I} f_i)(x) = \sup_{i\in I}(f_i(x))$, ce qui peut sembler être une tautologie, mais doit être démontré.)

(i). Si $x \leq x'$, alors pour tout $i \in I$, $f_i(x) \leq f_i(x') \leq f(x')$ par monotonie de f_i et le fait que $f_i \leq f$. En prenant le sup sur $i \in I$, $f(x) = \sup_{i\in I} f_i(x) \leq f(x')$, donc f est monotone. Soit maintenant $(x_j)_{j\in J}$ une famille dirigée d'éléments de X. On veut montrer que $f(\sup_{j\in J} x_j) = \sup_{j\in J} f(x_j)$. Le côté gauche vaut $\sup_{i\in I} f_i(\sup_{j\in J} x_j) = \sup_{i\in I}\sup_{j\in J} f_i(x_j)$ car f_i est continue pour tout i, et le côté droit vaut $\sup_{j\in J}\sup_{i\in I} f_i(x_j)$. Ces deux quantités sont égales, parce que $\sup_{i\in I}\sup_{j\in J} A_{ij} = \sup_{j\in J}\sup_{i\in I} A_{ij}$ pour n'importe quelle famille d'éléments A_{ij} : les deux côtés de l'équation ont les mêmes majorants.

(ii) Démontrer que f est le sup de $(f_i)_{i\in I}$, c'est démontrer que f est un majorant des f_i (évident), et que c'est le plus petit. Si g est un autre majorant, alors $f_i(x) \leq g(x)$ pour tout $i \in I$, donc $f(x) = \sup_{i\in I} f_i(x) \leq g(x)$, et on a fini.

Finalement, si Y est pointé, soit $\bot$ son plus petit élément : alors la fonction constante qui à tout x associe $\bot$ est le plus petit élément de $[X \to Y]$. □

On a un résultat similaire pour les produits. Étant donnés deux dcpos X et Y, leur produit $X \times Y$ est l'ensemble des couples (x, y) avec $x \in X$ et $y \in Y$. On l'ordonne par l'ordre point à point : $(x, y) \leq (x', y')$ si et seulement si $x \leq x'$ (dans X) et $y \leq y'$ (dans Y).

Lemme 6 *Soit X, Y deux dcpos. Le produit $X \times Y$ est un dcpo. Les sups dirigés sont calculés point à point : $\sup_{i\in I}(x_i, y_i) = (\sup_{i\in I} x_i, \sup_{i\in I} y_i)$ pour toute famille dirigée $(x_i, y_i)_{i\in I}$. De plus, si X et Y sont pointés, alors $X \times Y$ l'est aussi.*

Démonstration. Soit $(x_i, y_i)_{i\in I}$ une famille dirigée dans $X \times Y$. D'abord, $(x_i)_{i\in I}$ est elle aussi dirigée, mais dans X. En effet, elle est non vide, et pour tout couple d'indices $i, j \in I$, on peut trouver un $k \in I$ tel que $(x_i, y_i), (x_j, y_j) \leq (x_k, y_k)$, donc $x_i, x_j \leq x_k$. De façon similaire, $(y_i)_{i\in I}$ est dirigée. Donc $\sup_{i\in I} x_i$ et $\sup_{i\in I} y_i$ sont bien définis.

Pour démontrer que $\sup_{i\in I}(x_i, y_i) = (\sup_{i\in I} x_i, \sup_{i\in I} y_i)$, il suffit de démontrer que les deux côtés ont les mêmes majorants. Il s'agit, effectivement, dans les deux cas, de l'ensemble des couples (x, y) tels que x majore $(x_i)_{i\in I}$ et y majore $(y_i)_{i\in I}$.

Finalement, si X a un plus petit élément $\bot_X$ et Y a un plus petit élément $\bot_Y$, alors $(\bot_X, \bot_Y)$ est plus petit que tout élément de $X \times Y$, donc $X \times Y$ est pointé. □

2 Sémantique dénotationnelle de IMP

La sémantique dénotationnelle, et la théorie des dcpos, brille dans le cas des langages fonctionnels. Dans le cas de langages impératifs comme C ou IMP, elle est trop simple pour que l'on voie clairement l'intérêt de la théorie générale des dcpos. Néanmoins, nous commençons par IMP, où nos théorèmes de correction, d'adéquation et d'abstraction complète seront plus accessibles. Nous nous intéresserons à un langage fonctionnel en section 3.

Nous reprenons notre tentative de sémantique dénotationnelle pour IMP du poly précédent. Env et l'ensemble $Var \to \mathbb{Z}$ des environnements, vu comme un dcpo avec l'ordre d'égalité. On

équipe aussi Var et $\mathbb{Z}$ de l'ordre trivial d'égalité. À ce titre, Env est aussi le dcpo $[Var \to \mathbb{Z}]$ des fonctions continues de Var dans $\mathbb{Z}$... car toutes les fonctions de Var dans $\mathbb{Z}$ sont continues.

Pour tout dcpo X, nous définirons le *lifting* $X_\perp$ de X comme étant le dcpo union disjointe de X et de $\perp$, avec $\perp \leq x$ pour tout $x \in X$, l'ordre sur les éléments de $X_\perp$ qui sont dans X étant le même que l'ordre de X. On dit parfois que $X_\perp$ est obtenu en « ajoutant un nouvel élément $\perp$ en-dessous de X ».

La sémantique des expressions est donnée par la définition de $[\![e]\!]\,\rho \in \mathbb{Z}$ pour tout $\rho \in Env$. La définition est la même qu'en section 1.1 du poly précédent :

$$\begin{aligned}
[\![x]\!]\,\rho &= \rho(x) \\
[\![\dot{n}]\!]\,\rho &= n \\
[\![e_1 \dot{+} e_2]\!]\,\rho &= [\![e]\!]_1\,\rho + [\![e]\!]_2\,\rho \\
[\![\dot{-}e]\!]\,\rho &= -\,[\![e]\!]\,\rho.
\end{aligned}$$

On définit la sémantique des commandes $[\![c]\!]\,\rho$, pour $\rho \in Env$, comme un élément de $Env_\perp$, et non plus Env. Ceci est une modification (apparemment) très mineure de notre tentative avortée précédente. Ceci nous permet de poser :

$$\begin{aligned}
[\![x := e]\!]\,\rho &= \rho[x \mapsto [\![e]\!]\,\rho] \\
[\![\texttt{skip}]\!]\,\rho &= \rho \\
[\![c_1; c_2]\!]\,\rho &= \begin{cases} \perp & \text{si } [\![c_1]\!]\,\rho = \perp \\ [\![c_2]\!]\,([\![c_1]\!]\,\rho) & \text{sinon} \end{cases} \\
[\![\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2]\!]\,\rho &= \begin{cases} [\![c_1]\!]\,\rho & \text{si } [\![e]\!]\,\rho \neq 0 \\ [\![c_2]\!]\,\rho & \text{si } [\![e]\!]\,\rho = 0 \end{cases} \\
[\![\texttt{while}\ e\ \texttt{do}\ c]\!]\,\rho &= \mathrm{lfp}(F_{e,c})(\rho)
\end{aligned}$$

où $F_{e,c} \colon [Env \to Env_\perp] \to [Env \to Env_\perp]$ est la fonctionnelle suivante :

$$F_{e,c}(f)(\rho) = \begin{cases} \rho & \text{si } [\![e]\!]\,\rho = 0 \\ \perp & \text{si } [\![e]\!]\,\rho \neq 0 \text{ et } [\![c]\!]\,\rho = \perp \\ f([\![c]\!]\,\rho) & \text{si } [\![e]\!]\,\rho \neq 0 \text{ et } [\![c]\!]\,\rho \neq \perp \end{cases}$$

Les cas qui changent sont ceux de la séquence $c_1; c_2$, où l'on reconnaît que c_1 peut ne pas terminer ($[\![c_1]\!]\,\rho = \perp$), et surtout celui de la boucle `while`, qui est (enfin) bien défini. On notera notamment que l'on y calcule le plus petit point fixe d'une fonction $F_{e,c}$ de $[Env \to Env_\perp]$ dans lui-même. C'est bien défini, par le corollaire 3, parce que :

— $[Env \to Env_\perp]$ est un dcpo pointé. Son plus petit élément est la fonction constante qui à tout environnement ρ associe $\perp$ (la fonction qui ne termine sur aucun environnement de départ).

— Et parce que $F_{e,c}$ est continue... ce qui nous reste à prouver. Ce sera la proposition 7. Nous aurons besoin de quelques lemmes préliminaires.

Notons $[\![c]\!]$ la fonction qui à $\rho \in Env$ associe $[\![c]\!]\,\rho \in Env_\perp$.

Proposition 7 *Pour toute commande c, $[\![c]\!]$ est bien définie de Env vers $Env_\perp$. La fonction $F_{e,c}$ est continue.*

Démonstration. On démontre que $[\![c]\!]\,\rho$ est bien définie pour toute commande c et tout environnement ρ, par récurrence structurelle sur la commande c. Le seul cas intéressant est celui de la boucle `while`. Par hypothèse de récurrence, nous savons que $[\![c]\!]$ est bien définie.

Montrons d'abord que $F_{e,c}$ est monotone. Si $f \leq f'$, alors pour tout $\rho \in Env$, ou bien $[\![e]\!]\,\rho = 0$ et $F_{e,c}(f)(\rho) = \rho = F_{e,c}(f')(\rho)$; ou bien $[\![e]\!]\,\rho \neq 0$, $[\![c]\!]\,\rho = \perp$ et $F_{e,c}(f)(\rho) = \perp = F_{e,c}(f')$; ou bien $[\![e]\!]\,\rho \neq 0$, $[\![c]\!]\,\rho \neq \perp$ et $F_{e,c}(f)(\rho) = f([\![c]\!]\,\rho) \leq f'([\![c]\!]\,\rho) = F_{e,c}(f')(\rho')$. Dans tous les cas, $F_{e,c}(f)(\rho) \leq F_{e,c}(f')(\rho')$. Donc $F_{e,c}(f) \leq F_{e,c}(f')$, ce que l'on souhaitait démontrer.

Montrons la continuité de $F_{e,c}$. Soit $(f_i)_{i\in I}$ une famille dirigée d'éléments de $[Env \to Env_\perp]$, dont le sup est f. Nous devons démontrer que $F_{e,c}(f) = \sup_{i\in I} F_{e,c}(f_i)$, autrement dit que pour tout $\rho \in Env$, $F_{e,c}(f)(\rho) = \sup_{i\in I} F_{e,c}(f_i)(\rho)$. C'est évident si $[\![e]\!]\,\rho = 0$ (auquel cas tous les termes sont égaux à ρ), ou si $[\![e]\!]\,\rho \neq 0$ et $[\![c]\!]\,\rho = \perp$ (auquel cas tous les termes sont égaux à $\perp$). Dans le cas qui reste, $F_{e,c}(f)(\rho) = f([\![c]\!]\,\rho) = \sup_{i\in I} f_i([\![c]\!]\,\rho) = \sup_{i\in I} F_{e,c}(f_i)(\rho)$.

Comme $F_{e,c}$ est continue, $\mathrm{lfp}(F_{e,c})$ existe par le corollaire 3, et $[\![\texttt{while}\ e\ \texttt{do}\ c]\!]\,\rho$ est donc bien définie. □

On n'avait pas besoin de démontrer que $F_{e,c}$ était continue. Le fait qu'elle soit monotone suffit à démontrer l'existence de $\mathrm{lfp}(F_{e,c})$, par le théorème de Tarski 1. Nous aurons besoin de la continuité plus tard.

2.1 Correction, adéquation : IMP

La sémantique dénotationnelle est correcte vis-à-vis de la sémantique opérationnelle. (Pour les différentes sémantiques opérationnelles, voir le poly précédent.) Ceci peut prendre plusieurs formes. L'une des plus fortes est le fait que la sémantique dénotationnelle est un *invariant* des configurations au cours de l'exécution :

Lemme 8 *Si $(C, \rho) \to (C', \rho')$ dans la sémantique opérationnelle à petits pas (section 1.3 du poly précédent), alors $[\![C;]\!]\,\rho = [\![C';]\!]\,\rho'$.*

Démonstration. Par analyse de cas. Ceci ne présente aucune difficulté. Le seul cas intéressant est celui de la règle (8) :

$$(\texttt{while}\ e\ \texttt{do}\ c \cdot C, \rho) \quad \to \quad (c \cdot \texttt{while}\ e\ \texttt{do}\ c \cdot C, \rho) \qquad \text{si } [\![e]\!]\,\rho \neq 0$$

Démontrons d'abord que $[\![\texttt{while}\ e\ \texttt{do}\ c]\!]\,\rho = [\![c; \texttt{while}\ e\ \texttt{do}\ c]\!]\,\rho$. On a :

$$\begin{aligned} [\![\texttt{while}\ e\ \texttt{do}\ c]\!]\,\rho &= \mathrm{lfp}(F_{e,c})(\rho)) \\ &= F_{e,c}(\mathrm{lfp}(F_{e,c}))(\rho) \end{aligned}$$

parce que $\mathrm{lfp}(F_{e,c})$ est un point fixe de $F_{e,c}$. Comme $[\![e]\!]\,\rho \neq 0$, on n'a que deux cas à considérer. Si $[\![c]\!]\,\rho = \perp$, alors $F_{e,c}(\mathrm{lfp}(F_{e,c}))(\rho) = \perp = [\![c; \texttt{while}\ e\ \texttt{do}\ c]\!]\,\rho$. Sinon, $F_{e,c}(\mathrm{lfp}(F_{e,c}))(\rho) = \mathrm{lfp}(F_{e,c})([\![c]\!]\,\rho) = [\![\texttt{while}\ e\ \texttt{do}\ c]\!]\,([\![c]\!]\,\rho) = [\![c; \texttt{while}\ e\ \texttt{do}\ c]\!]\,\rho$.

On en déduit que $[\![\texttt{while}\, e\, \texttt{do}\, c; C']\!]\, \rho = [\![c; (\texttt{while}\, e\, \texttt{do}\, c; C')]\!]\, \rho$ en remarquant que la séquence est « sémantiquement associative » : pour toutes commandes c_1, c_2, c_3, $[\![c_1; (c_2; c_3)]\!] = [\![(c_1; c_2); c_3]\!]$. C'est une vérification pénible (on doit distinguer les trois cas $[\![c]\!]_1\, \rho = \bot$, $[\![c]\!]_1\, \rho \neq \bot$ et $[\![c]\!]_2\, \rho = \bot$, $[\![c]\!]_1\, \rho \neq \bot$ et $[\![c]\!]_2\, \rho \neq \bot$) mais élémentaire. Alors $[\![c; (\texttt{while}\, e\, \texttt{do}\, c; C')]\!]\, \rho = [\![(c; \texttt{while}\, e\, \texttt{do}\, c); C']\!]\, \rho$ vaut :

— soit $\bot$, si $[\![c; \texttt{while}\, e\, \texttt{do}\, c]\!]\, \rho = \bot$; on a alors $\bot = [\![\texttt{while}\, e\, \texttt{do}\, c; C']\!]\, \rho$;

— soit $[\![C']\!]\, ([\![c; \texttt{while}\, e\, \texttt{do}\, c]\!]\, \rho) = [\![C']\!]\, ([\![\texttt{while}\, e\, \texttt{do}\, c]\!]\, \rho) = [\![\texttt{while}\, e\, \texttt{do}\, c; C']\!]\, \rho$. □

Théorème 9 (Correction) *Si $\rho \vdash c \Rightarrow \rho'$ dans la sémantique opérationnelle à grands pas de* IMP*, alors $\rho' = [\![c]\!]\, \rho$.*

Démonstration. Deux démonstrations possibles. Soit on utilise le théorème 3 du poly précédent pour obtenir que $(c \cdot \epsilon, \rho) \to^* (\epsilon, \rho')$, puis l'on effectue une récurrence sur la longueur de cette dernière trace en invoquant le lemme 8 pour obtenir $[\![c; \epsilon']\!]\, \rho = [\![\epsilon']\!]\, \rho'$, qui est le résultat cherché. Soit on effectue directement une récurrence sur la dérivation donnée, π, de $\rho \vdash c \Rightarrow \rho'$. Le seul cas intéressant est de nouveau le cas de la boucle `while` lorsqu'elle ne s'arrête pas tout de suite, c'est-à-dire la règle (`while`). Comme dans la démonstration du lemme 8, le point fondamental à utiliser est que $[\![\texttt{while}\, e\, \texttt{do}\, c]\!]\, \rho$ se calcule comme un point fixe de $F_{e,c}$, appliqué à ρ. □

Ce n'est qu'une implication. Nous allons démontrer l'*adéquation*, c'est-à-dire l'implication réciproque. Ce n'est pas très difficile, mais il faudra maintenant utiliser que la sémantique dénotationnelle du `while` est le *plus petit* point fixe, pas n'importe quel point fixe. Nous aurons aussi besoin de la continuité de $F_{e,c}$, et ceci pour bénéficier de la formule explicite du plus petit point fixe donnée au corollaire 3.

Théorème 10 (Adéquation) *Si $[\![c]\!]\, \rho = \rho' \neq \bot$, alors $\rho \vdash c \Rightarrow \rho'$ est dérivable dans la sémantique opérationnelle à grands pas de* IMP.

Ceci est un résultat très fort. Il dit notamment que si $[\![c]\!]\, \rho \neq \bot$, alors le programme c doit nécessairement *terminer* en partant de ρ. C'est en fait le cœur de l'argument : le théorème d'adéquation est un théorème de terminaison.

Démonstration. On construit cette dérivation par récurrence structurelle sur c. Tous les cas sont faciles, sauf celui de la boucle `while`. La situation est la suivante : nous avons $[\![\texttt{while}\, e\, \texttt{do}\, c]\!]\, \rho = \rho'$, avec $\rho' \neq \bot$, et nous souhaitons produire une dérivation de $\rho \vdash \texttt{while}\, e\, \texttt{do}\, c \Rightarrow \rho'$. Nous savons aussi, par l'hypothèse de récurrence, que : (H) pour tout environnement ρ_1, si $[\![c]\!]\, \rho_1 \neq \bot$ alors $\rho_1 \vdash c \Rightarrow [\![c]\!]\, \rho_1$ est dérivable.

Par le corollaire 3, ρ' est égal au sup de la famille dirigée des $F^n_{e,c}(\overline{\bot})(\rho)$, $n \in \mathbb{N}$, où $\overline{\bot}$, le plus petit élément de $[Env \to Env_\bot]$, est la fonction constante de valeur $\bot$.

Comme $Env_\bot$ a la condition de chaîne croissante [1], ce sup est atteint pour une certaine valeur de n. Il ne reste plus qu'à montrer que pour tout $n \in \mathbb{N}$ tel que $\rho'' = F^n_{e,c}(\overline{\bot})(\rho)$ est différent de

1. Attention : on notera qu'en revanche, $[Env \to Env_\bot]$ n'a *pas* la condition de chaîne croissante, et $\mathrm{lfp}(F_{e,c})$ lui-même n'est en général atteint à aucune valeur de n. C'est $\mathrm{lfp}(F_{e,c})(\rho)$ qui est atteint, pour chaque ρ, à une valeur de n qui dépend en général de ρ.

$\bot$ (hypothèse que nous noterons (H')), il existe une dérivation de $\rho \vdash \texttt{while}\, e\, \texttt{do}\, c \Rightarrow \rho''$. Nous le montrons par récurrence sur n.

Pour $n = 0$, l'hypothèse (H') nous dit que $\overline{\bot}(\rho) \neq \bot$, ce qui est impossible. Du faux, on peut déduire n'importe quoi, en particulier la propriété qui nous intéresse.

Supposons donc $n \neq 0$. Nous exploitons le fait que $\rho'' = F^n_{e,c}(\overline{\bot})(\rho) = F_{e,c}(f)(\rho)$ avec $f = F^{n-1}_{e,c}(\overline{\bot})$, et examinons les trois cas de la définition de $F_{e,c}$. Si $[\![e]\!]\,\rho = 0$, $\rho'' = F_{e,c}(f)(\rho) = \rho$, on peut donc produire la dérivation :

$$\frac{}{\rho \vdash \texttt{while}\, e\, \texttt{do}\, c \Rightarrow \rho}\,(\texttt{while}_{fin})$$

Supposons maintenant $[\![e]\!]\,\rho \neq 0$. Il est impossible que $[\![c]\!]\,\rho = \bot$, sinon on aurait $\rho'' = F_{e,c}(f)(\rho) = \bot$, contredisant (H'). Donc $[\![c]\!]\,\rho \neq \bot$, ce qui implique l'existence d'une dérivation π_1 de $\rho \vdash c \Rightarrow \rho'$ avec $\rho' = [\![c]\!]\,\rho$, par (H) ((H) est mentionnée au début de la démonstration). De plus, (H') s'écrit $\rho'' = F^{n-1}_{e,c}(\overline{\bot})(\rho') \neq \bot$. On peut appliquer l'hypothèse de récurrence et en déduire une dérivation π_2 de $\rho' \vdash \texttt{while}\, e\, \texttt{do}\, c \Rightarrow \rho''$. On peut finalement produire la dérivation :

$$\frac{\begin{array}{cc}\vdots\,\pi_1 & \vdots\,\pi_2\\ \rho \vdash c \Rightarrow \rho' & \rho' \vdash \texttt{while}\, e\, \texttt{do}\, c \Rightarrow \rho''\end{array}}{\rho \vdash \texttt{while}\, e\, \texttt{do}\, c \Rightarrow \rho''}\,(\texttt{while})$$

□

2.2 Abstraction complète : IMP

Une notion classique en théorie des langages de programmation est celle d'*équivalence contextuelle* : deux programmes c et c' sont contextuellement équivalents si et seulement s'ils se comportent de façon identique dans tous les contextes C.

L'intuition est que C est un suite de commandes qui va servir à tester ce que font c et c'. On fait tourner c, ou c', mais on ne sait pas lequel des deux, dans le contexte C, et on regarde les résultats produits. Les programmes c et c' sont contextuellement équivalents si on n'arrivera jamais à voir la différence, avec aucun contexte C.

Quand je dis « on regarde les résultats produits », la convention ici est que l'on va demander des résultats faciles à interpréter : pas un environnement complet, juste un booléen. En l'occurrence, le booléen sera juste la réponse à la question : le programme termine-t-il ?

En IMP, les valeurs des variables sont importantes, mais ne font pas partie de la commande c. On définira donc la relation d'*équivalence contextuelle* entre couples (ρ, c) plutôt qu'entre commandes c comme suit. Au passage, on dira que $(c \cdot C, \rho)$ *termine* si et seulement si la trace maximale d'exécution partant de $(c \cdot C, \rho)$ dans la sémantique à petits pas est finie, ce qui est équivalent à demander que l'on peut dériver un jugement $\rho \vdash c; C^: \Rightarrow \rho'$ pour un certain ρ'.

On dit que (ρ, c) est contextuellement équivalent à (ρ', c'), en notation $(\rho, c) \cong (\rho', c')$, si et seulement si les contextes C pour lesquels $(c \cdot C, \rho)$ termine sont les mêmes que ceux pour lesquels $(c' \cdot C, \rho')$ termine.

La correction et l'adéquation donnent un critère simple d'équivalence contextuelle :

Lemme 11 *Si* $[\![c]\!]\,\rho = [\![c']\!]\,\rho'$*, alors* $(c, \rho) \cong (c', \rho')$.

Démonstration. Supposons que $(c \cdot C, \rho)$ termine. Il existe donc une dérivation de $\rho \vdash c; C^; \Rightarrow \rho'$ pour un certain environnement ρ'. Par la correction (théorème 9), $\rho' = [\![c; C^;]\!]\,\rho$ est différent de $\bot$. L'hypothèse implique que $[\![c'; C^;]\!]\,\rho' \neq \bot$. L'adéquation (théorème 10) implique que $(c' \cdot C, \rho')$ termine. La réciproque suit exactement le même raisonnement. □

Quand on a une implication, il est toujours intéressant de se demander si la réciproque est vraie. Cette propriété est l'*abstraction complète*, et IMP est complètement abstrait :

Théorème 12 (Abstraction complète) *Le langage* IMP *est complètement abstrait :* $[\![c]\!]\,\rho = [\![c']\!]\,\rho'$ *est équivalent à* $(c, \rho) \cong (c', \rho')$.

Démonstration. À la lumière du lemme 11, il reste à démontrer que si $[\![c]\!]\,\rho \neq [\![c']\!]\,\rho'$, alors on peut trouver un contexte C tel que $(c \cdot C, \rho)$ termine et $(c' \cdot C, \rho')$ ne termine pas, ou tel que $(c \cdot C, \rho)$ ne termine pas et $(c' \cdot C, \rho')$ termine.

Comme $\rho_\infty = [\![c]\!]\,\rho$ et $\rho'_\infty = [\![c']\!]\,\rho'$ sont différents, l'un des deux est différent de $\bot$, disons $\rho_\infty = [\![c]\!]\,\rho$. Si $\rho'_\infty \neq \bot$, il existe de plus une variable x telle que $\rho_\infty(x) \neq \rho'_\infty(x)$. Sinon, soit x une variable quelconque. Dans tous les cas, posons $n = \rho_\infty(x)$, et écrivons le contexte $C = c_{\text{test}} \cdot \epsilon$, où c_{test} est :

$$\texttt{if}\ x + (-\dot{n}) = 0\ \texttt{then skip else}\ \Omega,$$

Ω étant une commande quelconque qui ne termine jamais, par exemple `while` $\dot{1}$ `do skip`. Il est facile de voir que $[\![c; C^;]\!]\,\rho \neq \bot$ mais que $[\![c'; C^;]\!]\,\rho' = \bot$. Par adéquation et correction respectivement, $(c \cdot C, \rho)$ termine mais pas $(c' \cdot C', \rho)$. □

2.3 Plus faibles préconditions, logique de Hoare

Dans le même style d'idée, si nous nous donnons une commande c et une propriété P sur les environnements — c'est-à-dire fondamentalement juste une fonction de Env vers les booléens, ou de façon équivalente un sous-ensemble de Env — on peut se demander quels sont les environnements ρ tels que $[\![c]\!]\,\rho$ vérifie la propriété P. (Ce par quoi nous entendons que $[\![c]\!]\,\rho$ appartient à l'ensemble des environnements vérifiant P, et est donc un environnement : ceci implique donc notamment que c termine en partant de l'environnement ρ.)

Etant donné une propriété P, la propriété d'être un environnement ρ tel que $[\![c]\!]\,\rho$ vérifie P, c'est-à-dire tel que $P([\![c]\!]\,\rho)$ soit vraie, est appelée la *plus faible précondition* de c établissant P :

$$wp(c)(P) = (\rho \in Env \mapsto P([\![c]\!]\,\rho)).$$

où l'on note, en général, $(V \in D \mapsto g(V))$ la fonction qui à tout $V \in D$ associe $g(V)$. Il est équivalent de définir :

$$wp(c)(P)(\rho) = P([\![c]\!]\,\rho).$$

En général, $[\![c]\!]\,\rho$ peut être dans Env, auquel cas $P([\![c]\!]\,\rho)$ a un sens, ou bien valoir $\bot$, auquel cas nous conviendrons que $P(\bot)$ signifie « faux » ($\bot$ n'est pas dans l'ensemble des environnements vérifiant P : ce n'est même pas un environnement).

Il est facile de voir que l'on peut définir wp par récurrence sur la structure de c, directement, sans faire appel à la sémantique dénotationnelle de c :

$$\begin{array}{rcl}
wp(x := e)(P) & = & (\rho \in Env \mapsto P(\rho[x \mapsto [\![e]\!]\,\rho])) \\
wp(\texttt{skip})(P) & = & P \\
wp(c_1; c_2)(P) & = & wp(c_1)(wp(c_2)(P)) \\
wp(\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2)(P) & = & (\rho \in Env \mapsto \\
 & & \qquad ([\![e]\!]\,\rho \neq 0 \wedge wp(c_1)(P)(\rho)) \\
 & & \qquad \vee([\![e]\!]\,\rho = 0 \wedge wp(c_2)(P)(\rho)) \\
wp(\texttt{while}\ e\ \texttt{do}\ c)(P) & = & \mathrm{lfp}(\mathrm{Pre}^P_{e,c})
\end{array}$$

où $\mathrm{Pre}^P_{e,c}$ est la fonction, monotone, qui à chaque propriété Q associe la propriété :

$$\begin{array}{l}
(\rho \in Env \mapsto \\
\qquad ([\![e]\!]\,\rho = 0 \wedge P(\rho)) \\
\qquad \vee([\![e]\!]\,\rho \neq 0 \wedge wp(c)(Q)(\rho)))
\end{array}$$

On note $\wedge$ pour la conjonction (« et »), et $\vee$ pour la disjonction (« ou »). $\mathrm{Pre}_{e,c}$ définit une fonction des propriétés vers les propriétés. Les propriétés forment un treillis complet, isomorphe à $\mathbb{P}(Env)$, et l'on peut vérifier que $\mathrm{Pre}^P_{e,c}$ est une fonction continue de ce treillis dans lui-même. Le plus petit point fixe $\mathrm{lfp}(\mathrm{Pre}^P_{e,c})$ s'écrit donc aussi comme la fonction qui à tout $\rho \in Env$ associe $\bigvee_{n\in\mathbb{N}}(\mathrm{Pre}^P_{e,c})^n(\rho)$.

Exercice 17 *Démontrer qu'avec la définition de wp ci-dessus, on a effectivement $wp(c)(P)(\rho) = P([\![c]\!]\,\rho)$ pour toute commande c, toute propriété P sur les environnements, et tout environnement ρ.*

Démontrer aussi que $wp(\texttt{while}\ e\ \texttt{do}\ c)(P)$ est la plus faible *propriété Q sur les environnements telle que $Q(\rho) = P(\rho)$ si $[\![e]\!]\,\rho = 0$ et $Q(\rho) = wp(c)(Q)(\rho)$ sinon. On dit que Q est plus faible que Q' si et seulement si Q' implique Q, autrement dit si l'ensemble des ρ vérifiant Q contient celui des ρ vérifiant Q'.*

On appelle parfois les propriétés des *prédicats*, et si P est un prédicat, alors $wp(c)(P)$ est aussi un prédicat. On dit que $wp(c)$ est une sémantique (eh oui, encore une) par *transformateurs de prédicats* : pour un prédicat P sur les environnements en sortie de calcul de c, $wp(c)(P)$ est un prédicat sur les environnements en entrée du calcul de c. Les sémantiques par transformateurs de prédicats sont, pour cette raison, qualifiées de sémantiques *en arrière* (« backward »).

Si un prédicat est maintenant vu comme un ensemble d'environnements, alors $wp(c)(P)$ est l'ensemble des environnements ρ tels que si l'on exécute c en partant de ρ, (on termine et) on obtient un environnement dans l'ensemble P. C'est donc l'ensemble des *prédécesseurs* d'éléments de P pour la relation reliant les entrées et les sorties de P.

C'est dans cet esprit que nous avons nommé $\mathrm{Pre}^P_{e,c}$ le transformateur de prédicats utilisé dans la définition de $wp(\texttt{while}\, e\ \texttt{do}\ c)(P)$ plus haut. La disjonction infinie $\bigvee_{n\in\mathbb{N}}(\mathrm{Pre}^P_{e,c})^n(\rho)$ est souvent notée $(\mathrm{Pre}^P_{e,c})^*(\rho)$ (notation de *Kleene*), et est appelée l'ensemble des *prédécesseurs itérés* de ρ. Cette relation est fondamentale dans l'étude des systèmes de transition, notamment en model-checking.

On peut donner une syntaxe à ces constructions. On utilise des formules F d'une logique suffisamment expressive, par exemple l'arithmétique de Peano du second ordre. Notons $\rho \models F$ pour dire que $\rho \in Env$ vérifie la formule F. (On ne donnera pas de définition. A titre d'indication, si $\rho(x) = 3$, alors ρ vérifiera la formule $x \doteq \dot{3}$ mais pas la formule $x \dot{+} \dot{1} = \dot{0}$.) On définit la formule $\langle c\rangle F$, pour toute commande c et toute formule F, de sorte que $\rho \models \langle c\rangle F$ si et seulement si $[\![c]\!]\,\rho \neq \bot$ et $[\![c]\!]\,\rho \models F$. De façon équivalente, si l'on pose $[\![F]\!] = \{\rho \in Env \mid \rho \models F\}$, on définit $\langle c\rangle F$ de sorte que $[\![\langle c\rangle F]\!] = wp(c)([\![F]\!])$. On vérifie que l'on peut poser :

$$\begin{array}{rcl}
\langle x := e\rangle F & = & F[x := e] \\
\langle \texttt{skip}\rangle F & = & F \\
\langle c_1; c_2\rangle F & = & \langle c_1\rangle\langle c_2\rangle F \\
\langle \texttt{if}\, e\, \texttt{then}\, c_1\, \texttt{else}\, c_2\rangle F & = & (e \dot{\neq} \dot{0} \wedge \langle c_1\rangle F) \vee (e \doteq \dot{0} \wedge \langle c_2\rangle F) \\
\langle \texttt{while}\, e\, \texttt{do}\, c\rangle F & = & \forall A \cdot (\forall \vec{x} \cdot e \doteq \dot{0} \wedge F \Rightarrow A(\vec{x})) \wedge \\
 & & \qquad (\forall \vec{x} \cdot e \dot{\neq} \dot{0} \wedge \langle c\rangle A(\vec{x}) \Rightarrow A(\vec{x})) \\
 & & \Rightarrow A(\vec{x})
\end{array}$$

où la notation $F[x := e]$ représente la substitution de x par e dans F. La quantification $\forall A$ porte sur tous les prédicats A d'arité k, où k est le nombre de variables apparaissant dans $\texttt{while}\, e\, \texttt{do}\, c$, et $\vec{x}$ est le k-uplet de ces variables.

Exercice 18 *Montrer la relation $[\![\langle c\rangle F]\!] = wp(c)([\![F]\!])$. Pour le cas du* `while`*, exploiter le fait que $\mathrm{lfp}(\mathrm{Pre}_{e,c})$ est le plus petit pré-point fixe de $\mathrm{Pre}_{e,c}$ (voir la preuve du théorème 1).*

On notera que $\langle \texttt{while}\, e\, \texttt{do}\, c\rangle F$ est un principe de récurrence : sous l'hypothèse $\langle \texttt{while}\, e\, \texttt{do}\, c\rangle F$, on peut démontrer une formule G arbitraire en démontrant d'abord $\forall \vec{x} \cdot e \doteq \dot{0} \wedge F \Rightarrow G$ (cas de base), et ensuite $\forall \vec{x} \cdot e \dot{\neq} \dot{0} \wedge \langle c\rangle G \Rightarrow G$ (cas de récurrence). En effet, en instanciant $A(\vec{x})$ par G, dans le côté droit de la définition de $\langle \texttt{while}\, e\, \texttt{do}\, c\rangle F$, on obtient :

$$\begin{array}{l}
(\forall \vec{x} \cdot e \doteq \dot{0} \wedge F \Rightarrow G) \wedge \\
(\forall \vec{x} \cdot e \dot{\neq} \dot{0} \wedge \langle c\rangle G \Rightarrow G) \\
\quad \Rightarrow G
\end{array}$$

On peut éviter l'utilisation de quantification $\forall A$, et par exemple se ramener à l'utilisation d'une logique du premier ordre, en utilisant les *triplets de Hoare*. L'exercice 19 en présente la variante pour la correction *totale* ; ce vocable recouvre le fait que $\{F\}\ c\ \{F'\}$ signifie que partant d'un environnement ρ satisfaisant F, c *termine* en produisant un environnements satisfaisant F'. S'y oppose la notion de correction *partielle*, où l'on omet la terminaison.

$$\frac{}{\{F[x := e]\}\ x := e\ \{F\}}\,(H :=) \qquad \frac{}{\{F\}\ \texttt{skip}\ \{F\}}\,(H\,\texttt{skip})$$

$$\frac{\{F \wedge e \doteq \dot{0}\}\ c_1\ \{F'\} \quad \{F \wedge e \dot{\neq} \dot{0}\}\ c_2\ \{F'\}}{\{F\}\ \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{F'\}}\,(H\,\texttt{if})$$

$$\frac{\{F\}\ c_1\ \{G\} \quad \{G\}\ c_2\ \{F'\}}{\{F\}\ c_1; c_2\ \{F'\}}\,(H;)$$

$$\frac{\{I \wedge e \dot{\neq} \dot{0} \wedge e' = x\}\ c\ \{I \wedge e' \prec x\}}{\{I\}\ \texttt{while}\ e\ \texttt{do}\ c\ \{I \wedge e \doteq \dot{0}\}}\,(H\,\texttt{while}_t)$$

$$\frac{\{G\}\ c\ \{G'\}}{\{F\}\ c\ \{F'\}}\,(H \Rightarrow) \quad \begin{array}{r}\text{si } \mathcal{L} \vdash F \Rightarrow G \\ \text{et } \mathcal{L} \vdash G' \Rightarrow F'\end{array}$$

FIGURE 1 – La logique de Hoare pour la correction totale

Exercice 19 (Logique de Hoare, correction totale) *On se fixe une logique $\mathcal{L}$ disposant des expressions atomiques $e \doteq \dot{0}$, $e \dot{\neq} \dot{0}$ pour chaque expression e de* IMP*, de la disjonction et de la conjonction. (Noter qu'on n'aura pas besoin de la quantification du second ordre $\forall A \ldots$ ni même d'aucune quantification. On n'a pas besoin de négation non plus, même si nous allons utiliser l'implication $\Rightarrow$ plus bas.)*

Un triplet de Hoare est une expression de la forme $\{F\}\ c\ \{F'\}$ où F, F' sont des formules de $\mathcal{L}$ et c une commande. Il est valide pour la correction totale *si et seulement si $F \Rightarrow \langle c \rangle F'$ est prouvable dans la logique $\mathcal{L}$. Dans la règle $(H;)$, G est une formule arbitraire (à deviner, si l'on part de la conclusion). Dans la règle $(H\,\texttt{while}_t)$, I (l'*invariant *de la boucle) est aussi une formule arbitraire, e' est une expression arbitraire de la logique (le* variant *; c'est une expression de la logique, qui peut inclure davantage de constructions que les expressions de programmes), et x est une variable fraîche (n'apparaissant pas dans I, e, e'). De plus, $\prec$ est une relation binaire définissable dans la logique $\mathcal{L}$, et que l'on peut montrer bien fondée, c'est-à-dire qu'il n'existe aucune chaîne infinie décroissante $v_1 \succ v_2 \succ \cdots \succ v_n \succ \cdots$. La règle $(H \Rightarrow)$ est la règle de* conséquence*. On notera $\mathcal{L} \vdash F \Rightarrow G$ pour dire que G est conséquence logique de F dans la logique $\mathcal{L}$.*

Montrer que les règles de la figure 1, qui définissent la logique de Hoare *(pour la correction totale), sont correctes au sens où toute dérivation fondée sur cet ensemble de règles a comme conclusion un triplet valide pour la correction totale.*

Exercice 20 (Logique de Hoare, correction partielle) *On obtient la logique de Hoare pour la*

correction partielle *en omettant les variants dans la règle* $(H\,\texttt{while}_t)$*, qui devient :*

$$\frac{\{I \wedge e \doteq \dot{0}\}\ c\ \{I\}}{\{I\}\ \texttt{while}\, e\, \texttt{do}\ c\ \{I \wedge e \dot{\neq} \dot{0}\}}\ (H\,\texttt{while}_p)$$

Montrer que les règles de la logique de Hoare pour la correction partielle sont toujours correctes.

Montrer qu'elles sont aussi correctes dans le sens suivant : si $\{F\}\ c\ \{F'\}$ *est dérivable en logique de Hoare pour la correction partielle, alors pour tout environnement* ρ *satisfaisant* F*, si* $\rho \vdash c \Rightarrow \rho'$ *alors* ρ' *satisfait* F'*. (Remarquez que cette dernière condition dit que si* c *termine, alors* ρ' *vérifie* F'*, mais qu'on ne demande pas à* c *de terminer.)*

2.4 Sémantique par passage à la continuation

C'est encore une fois la même idée que pour wp... mais en s'apercevant que nos « prédicats » peuvent retourner bien autre chose qu'un booléen.

On se donne un dcpo pointé Ans arbitraire. Un élément de Ans est traditionnellement appelé une *réponse*. Ans était le domaine des booléens jusqu'ici, avec faux plus petit que vrai. Une fonction de Env vers Ans est une *continuation* : elle dit ce qu'il faudra donner comme réponse pour chaque environnement que nous pourrons calculer.

On reproduit la définition de $wp(c)$ par récurrence sur c, et on la note $C\,[\![c]\!]$: c'est une fonction prenant une continuation κ et un environnement ρ, et retournant une réponse. On notera comme avant $C\,[\![c]\!]\,\kappa$ la fonction qui à ρ associe $C\,[\![c]\!]\,\kappa\rho$.

$$\begin{aligned}
C\,[\![x := e]\!]\,\kappa\rho &= \kappa(\rho[x \mapsto [\![e]\!]\,\rho]) \\
C\,[\![\texttt{skip}]\!]\,\kappa\rho &= \kappa(\rho) \\
C\,[\![c_1; c_2]\!]\,\kappa\rho &= C\,[\![c_1]\!]\,(C\,[\![c_2]\!]\,\kappa)(\rho) \\
C\,[\![\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2]\!]\,\kappa\rho &= \begin{cases} C\,[\![c_1]\!]\,\kappa\rho & \text{si } [\![e]\!]\,\rho \neq 0 \\ C\,[\![c_2]\!]\,\kappa\rho & \text{si } [\![e]\!]\,\rho = 0 \end{cases} \\
C\,[\![\texttt{while}\, e\, \texttt{do}\ c]\!]\,\kappa\rho &= \left(\sup_{n\in\mathbb{N}} Pre^n_{e,c}(\kappa)\right)(\rho)
\end{aligned}$$

où $Pre_{e,c}(\kappa)$ est la continuation qui à tout $\rho \in Env$ associe $\kappa(\rho)$ si $[\![e]\!]\,\rho = 0$, $C\,[\![c]\!]\,\kappa\rho$ sinon. Le sup est pris dans le dcpo pointé $[Env \to Ans]$ des continuations (qui sont toutes continues, ici).

Jusqu'ici, la sémantique à continuations n'est qu'une réécriture de ce dont nous disposions déjà. On peut faire bien mieux avec des continuations ! Par exemple, on peut définir la sémantique d'un langage *avec exceptions*. Il suffit pour cela de passer à la fonction sémantique non plus une continuation, mais une continuation κ traitant de ce qu'il faudra faire si le programme termine normalement (sans lancer d'exception), plus une liste κ_* de continuations κ_E, une pour chaque nom d'exception E. (La liste κ_* sera codée comme une fonction de l'ensemble des noms d'exception E vers les continuations, et $\kappa_E = \kappa_*(E)$.)

Sans développer tous les détails, on pourra facilement enrichir notre sémantique à continuations de définitions pour de nouvelles constructions, comme :

$$\begin{aligned} C\,[\![\texttt{try}\,c\,\texttt{with}\,E \Rightarrow c']\!]\,(\kappa \mid \kappa_*) &= C\,[\![c]\!]\,(\kappa \mid \kappa_*[E \mapsto [\![c']\!]\,(\kappa \mid \kappa_*)]) \\ C\,[\![\texttt{throw}\,E]\!]\,(\kappa \mid \kappa_*) &= \kappa_E \end{aligned}$$

où la dernière ligne se lit peut-être plus facilement sous la forme :

$$C\,[\![\texttt{throw}\,E]\!]\,(\kappa \mid \kappa_*)\rho = \kappa_E(\rho).$$

La sémantique de la phase 2 du projet utilise des exceptions. Nous n'utiliserons pas ce style de sémantique dans ce cadre. Le format des sémantiques à continuation est déroutant au départ... mais concis !

On peut décrire bien plus que des exceptions avec des continuations, et notamment certaines constructions pour lesquelles les continuations semblent indispensables. Nous citerons l'exemple des *coroutines* (aussi appelés *processus coopératifs*) : des programmes qui s'exécutent chacun pendant un temps avant de passer la main au suivant. Je ne décrirai pas formellement comment ceci s'effectue, et me contenterai de donner une idée. Pour une réalisation concrète de cette idée (et bien plus), voir la thèse [1] de Gabriel Kerneis sur le langage CPC (*continuation-passing C*).

En tout état de cause, voici quelques pistes pour une définition d'une sémantique dénotationnelle d'une extension de IMP avec coroutines, et notamment avec une instruction `yield` passant la main à la coroutine suivante. On sauvegarde l'état de chaque coroutine sous forme de sa *continuation courante*, c'est-à-dire de la fonction qu'il faudra exécuter pour accomplir le reste de l'exécution de la coroutine. La sémantique maintient une liste circulaire de continuations courantes, en plus des paramètres usuels κ et ρ. La primitive `yield` récupère la continuation courante κ de la coroutine courante π, l'ajoute dans la liste, dépile la continuation suivante dans la liste, et l'évalue, provoquant ainsi l'exécution du code de la coroutine suivante (jusqu'à ce qu'elle-même appelle `yield`).

3 Sémantique dénotationnelle de PCF

La sémantique dénotationnelle, et la théorie des dcpos, brille dans le cas des langages fonctionnels.

Introduisons donc un tel langage : un fragment de Caml, disposant des fonctions (d'ordre supérieur, c'est-à-dire pouvant porter elles-mêmes sur d'autres fonctions), avec fonctions récursives (`letrec`), un peu d'arithmétique entière, et tests. Et surtout, ce langage, appelé *PCF* (Programming language for Computable Functions, introduit et étudié par Gordon Plotkin en 1977 [2]), est typé.

Les types sont :

$$\begin{aligned} \sigma, \tau, \ldots &::= \texttt{int} \\ &\mid \sigma \to \tau \end{aligned}$$

La sémantique (dénotationnelle) des types est donnée par un dcpo pointé $[\![\tau]\!]$ pour chaque type τ :

— $[\![\texttt{int}]\!]$ est le dcpo $\mathbb{Z}_\perp$ des entiers relatifs plus le symbole spécial $\perp$, avec la relation d'ordre telle que $\perp \leq n$ pour tout $n \in \mathbb{Z}$; pour tous $m, n \in \mathbb{Z}$ avec $m \neq n$, m et n sont *incomparables* dans l'ordre : on dit que le domaine $\mathbb{Z}_\perp$ est *plat*.
— $[\![\sigma \to \tau]\!]$ est le dcpo $[[\![\sigma]\!] \to [\![\tau]\!]]$ des fonctions continues de $[\![\sigma]\!]$ vers $[\![\tau]\!]$, avec l'ordre point à point. Rappelons que celui-ci est défini par $f \leq g$ si et seulement si $f(x) \leq g(x)$ pour tout $x \in [\![\sigma]\!]$.

L'ensemble des *expressions de type* τ est défini, pour chaque type τ simultanément, comme le plus petit tel que :

— Toute variable x_τ, y_τ, z_τ, ..., est une expression de type τ (on suppose un nombre infini dénombrable de variables avec l'indice τ, pour chaque type τ) ;
— Pour tout $n \in \mathbb{Z}$, $\dot{n}$ est une expression de type `int` ;
— Si u est une expression de type $\sigma \to \tau$ et v est une expression de type σ, alors uv est une expression de type τ ;
— si u est une expression de type τ et v est une expression de type λ, alors $\texttt{letrec } f_{\sigma\to\tau}(x_\sigma) = u \texttt{ in } v$ est une expression de type λ ;
— si u et v sont deux expressions de type `int`, alors $u \dot{+} v$ est une expression de type `int` ;
— si u est une expression de type `int`, alors $\dot{-}u$ est une expression de type `int` ;
— si u est une expression de type `int`, v et w sont deux expressions de type τ, alors $\texttt{if } u = 0 \texttt{ then } v \texttt{ else } w$ est une expression de type τ.

Appelons *environnement* ρ toute fonction (totale) qui à chaque variable x_τ associe un élément de $[\![\tau]\!]$. On notera Env le dcpo des environnements, avec l'ordre *produit*[2] : $\rho \leq \rho'$ si et seulement si pour toute variable x_τ, $\rho(x_\tau) \leq \rho'(x_\tau)$.

On peut alors définir la sémantique d'une expression u de type τ comme une valeur $[\![u]\!]\,\tau$ dans le dcpo $[\![\tau]\!]$, via la définition de la figure 2. Dans l'évaluation des fonctions `letrec`, on note $(V \in D \mapsto g(V))$ la fonction qui à tout $V \in D$ associe $g(V)$. La règle du `letrec` est sans doute plus facile à lire si on la décompose :

— d'abord, on définit une fonctionnelle $F^\rho_{f_{\sigma\to\tau}, x_\sigma, u}$ qui à chaque fonction φ (candidate courante pour la valeur de la fonction $f_{\sigma\to\tau}$) associe la fonction $(V \in [\![\sigma]\!] \mapsto [\![u]\!]\,(\rho[f_{\sigma\to\tau} \mapsto \varphi, x_\sigma \mapsto V]))$, qui est une meilleure approximation que φ de la sémantique de $f_{\sigma\to\tau}$;
— on en calcule le plus petit point fixe $\mathrm{lfp}(F^\rho_{f_{\sigma\to\tau}, x_\sigma, u})$, ce qui nous donne la sémantique de $f_{\sigma\to\tau}$; pour clarifier ce point, j'ai introduit une construction de langage fictive $\texttt{rec } f_{\sigma\to\tau}(x_\sigma) = u$, dénotant $\ll$ la fonction $f_{\sigma\to\tau}$ qui, appliquée x_σ, retourne $u \gg$, et défini sa sémantique comme étant $\mathrm{lfp}(F^\rho_{f_{\sigma\to\tau}, x_\sigma, u})$;
— on évalue ensuite v dans l'environnement obtenu à partir de ρ en associant à $f_{\sigma\to\tau}$ la fonction que nous venons de définir.

Il n'est pas du tout immédiat que cette sémantique soit bien définie. En cause, le fait que lfp ne calcule le plus petit point *que* des fonctions continues. Rien que pour cela, nous devons vérifier que la sémantique $[\![u]\!]\,\rho$ de u est bien définie *et continue* pour tout u.

Nous aurons besoin de quelques lemmes auxiliaires. Pour une famille dirigée $(a_i)_{i\in I}$, on dit qu'une sous-famille $(a_i)_{i\in J}$ (avec $J \subseteq I$) est *cofinale* si et seulement si pour tout $i \in I$, il existe

2. Si l'on identifie Env à un dcpo de fonctions des variables vers les valeurs, c'est une variante infime de l'ordre point à point.

$$
\begin{array}{rcl}
[\![x_\tau]\!]\,\rho & = & \rho(x_\tau) \\
[\![\dot{n}]\!]\,\rho & = & n \\
[\![uv]\!]\,\rho & = & [\![u]\!]\,\rho([\![v]\!]\,\rho) \\
[\![\texttt{letrec}\ f_{\sigma\to\tau}(x_\sigma) = u\ \texttt{in}\ v]\!] & = & [\![v]\!]\,(\rho[f_{\sigma\to\tau} \mapsto [\![\texttt{rec}\ f_{\sigma\to\tau}(x_\sigma) = u]\!]\,\rho]) \\
[\![\texttt{rec}\ f_{\sigma\to\tau}(x_\sigma) = u]\!]\,\rho & = & \mathrm{lfp}(F^\rho_{f_{\sigma\to\tau},x_\sigma,u}) \\
 & & \text{où } F^\rho_{f_{\sigma\to\tau},x_\sigma,u}(\varphi) = (V \in [\![\sigma]\!] \mapsto [\![u]\!]\,(\rho[f_{\sigma\to\tau} \mapsto \varphi, x_\sigma \mapsto V])) \\
[\![u\dot{+}v]\!]\,\rho & = & \left\{ \begin{array}{ll} [\![u]\!]\,\rho + [\![v]\!]\,\rho & \text{si } [\![u]\!]\,\rho \neq \bot, [\![v]\!]\,\rho \neq \bot \\ \bot & \text{sinon} \end{array} \right. \\
[\![\dot{-}u]\!]\,\rho & = & \left\{ \begin{array}{ll} -\,[\![u]\!]\,\rho & \text{si } [\![u]\!]\,\rho \neq \bot \\ \bot & \text{sinon} \end{array} \right. \\
[\![\texttt{if}\ u = 0\ \texttt{then}\ v\ \texttt{else}\ w]\!]\,\rho & = & \left\{ \begin{array}{ll} [\![v]\!]\,\rho & \text{si } [\![u]\!]\,\rho = 0 \\ [\![w]\!]\,\rho & \text{si } [\![u]\!]\,\rho \neq \bot, 0 \\ \bot & \text{si } [\![u]\!]\,\rho = \bot \end{array} \right.
\end{array}
$$

FIGURE 2 – Sémantique dénotationnelle de PCF

un $j \in J$ tel que $a_i \leq a_j$.

Lemme 13 *Si $(a_i)_{i\in J}$ est une sous-famille cofinale d'une famille dirigée $(a_i)_{i\in I}$ d'un dcpo X, elle est elle-même dirigée, et pour toute fonction monotone f de X vers un dcpo Y, $\sup_{i\in I} f(a_i) = \sup_{i\in J} f(a_i)$.*

Démonstration. D'abord, la famille $(a_i)_{i\in J}$ est non vide : prendre $i \in I$ quelconque, puis un $j \in J$ tel que $a_i \leq a_j$.

Ensuite, elle est dirigée : si $i, j \in J$, on prend $k \in I$ tel que $a_i, a_j \leq a_k$, puis un $k' \in J$ tel que $a_k \leq a_{k'}$ par cofinalité.

On a $\sup_{i\in I} f(a_i) \geq \sup_{i\in J} f(a_j)$ car I contient J. Réciproquement, pour tout $i \in I$, il existe un $j \in J$ tel que $a_i \leq a_j$, donc $f(a_i) \leq f(a_j)$; donc $f(a_i) \leq \sup_{j\in J} f(a_j)$; on en déduit $\sup_{i\in I} f(a_i) \leq \sup_{j\in J} f(a_j)$ en prenant les sups sur les $i \in I$. □

Lemme 14 *Si $(a_i)_{i\in I}$ est une famille dirigée dans un dcpo X, et f une fonction monotone de $X \times X$ vers un dcpo Y, alors $\sup_{i,j\in I} f(a_i, a_j) = \sup_{k\in I} f(a_k, a_k)$.*

Démonstration. Il suffit d'appliquer le lemme 13, en remarquant que la famille des (a_k, a_k), $k \in I$, est cofinale dans celle des (a_i, a_j), $i, j \in I$: étant donné (a_i, a_j), on utilise le fait que la famille est dirigée pour trouver un $k \in I$ tel que $a_i, a_j \leq a_k$, donc $(a_i, a_j) \leq (a_k, a_k)$. □

Lemme 15 *L'opérateur* $\mathrm{lfp}\colon [X \to X] \to X$ *sur un dcpo pointé X est lui-même une fonction continue.*

Démonstration. Rappelons que $\mathrm{lfp}(f) = \sup_{n\in\mathbb{N}} f^n(\bot)$ pour tout $f \in [X \to X]$ (corollaire 3).

Si $f \leq f'$, alors $\mathrm{lfp}(f) = \sup_{n\in\mathbb{N}} f^n(\bot) \leq \sup_{n\in\mathbb{N}} f'^n(\bot) = \mathrm{lfp}(f')$, donc lfp est monotone.

Et si $(f_i)_{i\in I}$ est une famille dirigée dans $[X \to X]$ de sup f, on note que pour tout $n \in \mathbb{N}$, $f^n(\bot) = \sup_{i\in I} f_i^n(\bot)$: c'est par récurrence sur n, le cas récurrent étant $f^{n+1}(\bot) = f(f^n(\bot)) = \sup_{i,j\in I} f_i(f_j^n(\bot))$ (par hypothèse de récurrence) $= \sup_{i\in I} f_i(f_i^n(\bot)) = \sup_{i\in I} f_i^{n+1}(\bot)$ par le lemme 14. Donc $\mathrm{lfp}(f) = \sup_{n\in\mathbb{N}} f^n(\bot) = \sup_{n\in\mathbb{N}} \sup_{i\in I} f_i^n(\bot) = \sup_{i\in I} \sup_{n\in\mathbb{N}} f_i^n(\bot) = \sup_{i\in I} \mathrm{lfp}(f_i)$. □

Notons $[\![u]\!]$ la fonction qui à tout environnement ρ associe $[\![u]\!]\,\rho$.

Proposition 16 *Pour toute expression u de type τ, $[\![u]\!]$ est bien définie et continue de Env vers $[\![\tau]\!]$.*

Démonstration. Par récurrence structurelle sur u. C'est évident pour les variables et les constantes. Pour les additions et opposés, on note que $\mathbb{Z}_\bot$ étant plat, il a la condition de chaîne croissante. Par l'exercice 16, il suffit de montrer que $[\![u \dot{+} v]\!]$, $[\![\dot{-}u]\!]$ sont monotones. C'est facile si l'on considère que, dans $\mathbb{Z}_\bot$, $a \leq b$ si et seulement si $a = b$, ou bien $a = \bot$ et $b \in \mathbb{Z}$. Attention de ne pas confondre l'ordre (plat) avec l'ordre usuel de $\mathbb{Z}$!

Pour les tests, on doit d'abord montrer la monotonie. Autrement dit, nous devons montrer que si $\rho \leq \rho'$ alors $[\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho \leq [\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho'$. Par la remarque que l'on vient de faire, il suffit de considérer les cas $[\![u]\!]\,\rho = [\![u]\!]\,\rho'$ (alors $[\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho = [\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho'$) et $[\![u]\!]\,\rho = \bot$, $[\![u]\!]\,\rho' \in \mathbb{Z}$ (alors $[\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho = \bot$ est inférieur ou égal à toute valeur). Pour la continuité proprement dite, soit $(\rho_i)_{i\in I}$ une famille dirigée d'environnements de sup ρ. Si $[\![u]\!]\,\rho_i = \bot$ pour tout $i \in I$, alors par continuité de $[\![u]\!]$, $[\![u]\!]\,\rho = \bot$, et donc $[\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho = \bot = \sup_{i\in I} \bot = \sup_{i\in I} [\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho_i$. Sinon, il existe un élément j_0 dans l'ensemble $J \subseteq I$ des indices i tels que $[\![u]\!]\,\rho_i \neq \bot$. La famille $(\rho_j)_{j\in J}$ est alors cofinale dans $(\rho_i)_{i\in I}$: pour tout $i \in I$, soit $j \in I$ tel que $\rho_i, \rho_{j_0} \leq \rho_j$; comme $j_0 \in J$ et $[\![u]\!]$ est croissante, $[\![u]\!]\,\rho_j \geq [\![u]\!]\,\rho_{j_0} \neq \bot$, donc $j \in J$. On peut donc appliquer le lemme 13 : $\sup_{i\in I} [\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho_i = \sup_{j\in J} [\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho_j$. Or la famille des $[\![u]\!]\,\rho_j$, $j \in J$, est une famille dirigée (car $[\![u]\!]$ est monotone) de $\mathbb{Z}$ (ne contenant pas $\bot$), elle est donc réduite à un seul élément. Si cet élément est non nul, $[\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho_j = [\![v]\!]\,\rho_j$ pour tout j, donc $\sup_{i\in I} [\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho_i = \sup_{j\in J} [\![v]\!]\,\rho_j = \sup_{i\in I} [\![v]\!]\,\rho_i$ (par le lemme 13 de nouveau) $= [\![v]\!]\,\rho = [\![\mathtt{if}\ u = 0\ \mathtt{then}\ v\ \mathtt{else}\ w]\!]\,\rho$. S'il est nul, le raisonnement est similaire, passant par $[\![w]\!]$ plutôt que $[\![v]\!]$.

Pour les applications uv, disons que u est de type $\sigma \to \tau$. D'abord, $[\![uv]\!]\,\rho$ est une fonction croissante de ρ : si $\rho \leq \rho'$, alors $[\![uv]\!]\,\rho = [\![u]\!]\,\rho([\![v]\!]\,\rho) \leq [\![u]\!]\,\rho'([\![v]\!]\,\rho)$ (parce que $[\![u]\!]$ est croissante) $\leq [\![u]\!]\,\rho'([\![v]\!]\,\rho')$ (parce que $[\![v]\!]\,\rho \leq [\![v]\!]\,\rho'$, $[\![v]\!]$ étant croissante, et parce que $[\![u]\!]\,\rho' \in [\![\sigma \to \tau]\!]$ est une fonction croissante). Ensuite, soit $(\rho_i)_{i\in I}$ une famille dirigée d'environnements de sup ρ. Sachant que $[\![u]\!]$, $[\![v]\!]$ sont continues, $[\![uv]\!]\,\rho = \sup_{i\in I} [\![u]\!]\,\rho_i(\sup_{j\in I} [\![v]\!]\,\rho_j)$, et sachant que $[\![u]\!]\,\rho_i$ est une fonction continue pour tout $i \in I$, $[\![uv]\!]\,\rho = \sup_{i,j\in I} [\![u]\!]\,\rho_i([\![u]\!]\,\rho_j)$. Par le lemme 14, ceci est égal à $\sup_{i\in I} [\![u]\!]\,\rho_i([\![v]\!]\,\rho_i) = \sup_{i\in I} [\![uv]\!]\,\rho_i$.

Finalement, pour `letrec`, on vérifie de façon élémentaire que la fonction $F^\rho_{f_{\sigma\to\tau},x_\sigma,u}(\varphi)$ est bien définie et continue pour toute fonction continue φ, que la fonctionnelle $F^\rho_{f_{\sigma\to\tau},x_\sigma,u}$ est elle-même continue, et dépend de ρ de façon continue. Par le lemme 15, on en déduit que

$\mathrm{lfp}(F^{\rho}_{f_{\sigma\to\tau},x_\sigma,u})$ dépend continument de ρ. En utilisant le lemme 14, on en déduit que $[\![v]\!]\,(\rho[f_{\sigma\to\tau} \mapsto \mathrm{lfp}(F^{\rho}_{f_{\sigma\to\tau},x_\sigma,u})])$ est définie et continue en ρ, donc qu'il en est de même de $[\![\texttt{letrec}\ f_{\sigma\to\tau}(x_\sigma) = u\ \texttt{in}\ v]\!]\,\rho$ (exercice). □

3.1 Correction : PCF$_V$

On peut se donner aussi une sémantique opérationnelle, et la première question à se poser une fois que nous l'aurons fait est celle du rapport entre les deux sémantiques.

Nous donnons une sémantique à grands pas. Elle retourne non pas des valeurs au sens sémantique du terme, mais ce que nous appellerons des *P-valeurs* (des valeurs au sens de Plotkin) :

— Les valeurs de type `int` sont les constantes $\dot{n}$, $n \in \mathbb{Z}$.

— Les valeurs de type $\sigma \to \tau$ sont les *clôtures*, c'est-à-dire des couples $\langle \texttt{rec}\ f_{\sigma\to\tau}(x_\sigma) = u, E\rangle$, où u est une expression de type τ, et E est un *P-environnement*, c'est-à-dire une fonction (partielle) de domaine fini $\mathrm{dom}\ E$ qui à chaque variable x_τ dans son domaine associe une P-valeur de type τ. On demande de plus que $\mathrm{dom}\ E$ contienne $\mathrm{fv}(u) \setminus \{x_\sigma, f_{\sigma\to\tau}\}$.

On a besoin d'expliquer plusieurs choses ici. D'abord, le principe d'une clôture est que c'est fondamentalement le corps d'une définition récursive de fonction par `letrec`, avec les noms de la fonction définie ($f_{\sigma\to\tau}$) et du paramètre formel (x_σ).

Ce qui est important, c'est qu'on l'associe avec un P-environnement, qui donne les P-valeurs de toutes les variables libres. Par exemple, E devra donner une P-valeur à $y_{\texttt{int}}$ dans $\langle \texttt{rec}\ f_{\texttt{int}\to\texttt{int}}(x_{\texttt{int}}) = x_{\texttt{int}} \dot{+} y_{\texttt{int}}, E\rangle$.

Ceci permettra à notre langage d'implémenter la *liaison lexicale*.

Formellement, l'ensemble $\mathrm{fv}(u)$ des variables libres est défini par récurrence structurelle sur u par :

$$\mathrm{fv}(x_\tau) = \{x_\tau\} \qquad \mathrm{fv}(\dot{n}) = \emptyset \qquad \mathrm{fv}(uv) = \mathrm{fv}(u) \cup \mathrm{fv}(v)$$
$$\mathrm{fv}(\texttt{letrec}\ f_{\sigma\to\tau}(x_\sigma) = u\ \texttt{in}\ v) = (\mathrm{fv}(u) \setminus \{f_{\sigma\to\tau}, x_\sigma\}) \cup (\mathrm{fv}(v) \setminus \{f_{\sigma\to\tau}\})$$
$$\mathrm{fv}(u \dot{+} v) = \mathrm{fv}(u) \cup \mathrm{fv}(v) \qquad \mathrm{fv}(\dot{-}u) = \mathrm{fv}(u)$$
$$\mathrm{fv}(\texttt{if}\ u = 0\ \texttt{then}\ v\ \texttt{else}\ w) = \mathrm{fv}(u) \cup \mathrm{fv}(v) \cup \mathrm{fv}(w)$$

La définition que nous avons donnée des P-valeurs utilise celle des P-environnements, qui elle-même se réfère aux P-valeurs : notre définition est donc récursive. Il y a trois façons (équivalentes) de résoudre le problème. La première est de définir l'ensemble des P-valeurs (avec leur type associé) comme le *plus petit* ensemble vérifiant les conditions données ci-dessus. Ceci ne montre pas clairement que cet ensemble existe. La deuxième est de définir le fait d'être une P-valeur p (de type τ), resp., un P-environnement, par des jugements $\vdash p : \tau$, resp., $\vdash E\ \mathrm{Env}$, et de dire quels sont les jugements dérivables dans le système de déduction de la figure 3. La troisième façon est de dire qu'une valeur est un arbre fini : ses feuilles sont des valeurs $\dot{n}$ entières, ses nœuds internes sont des clôtures $\langle \texttt{rec}\ f_{\sigma\to\tau}(x_\sigma) = u, [x_1 \mapsto p_1, \cdots, x_n \mapsto p_n]\rangle$ ayant n fils, un pour chaque valeur p_i, $1 \le i \le n$.

Nous en donnons une sémantique à grands pas. Nous examinons la variante PCF$_V$ en *appel par valeur*. Le langage PCF de Plotkin [2] est originellement en appel par nom... et la sémantique

$$\frac{}{\vdash \dot{n} : \texttt{int}} \text{ si } n \in \mathbb{Z} \qquad \frac{\vdash E \text{ Env}}{\vdash \langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u, E\rangle : \sigma \to \tau} \text{ si } u \text{ est une expression de type } \tau, \text{ et } \mathrm{fv}(u) \setminus \{f_{\sigma\to\tau}, x_\sigma\} \subseteq \mathrm{dom}\ E$$

$$\frac{\vdash p_1 : \tau_1 \quad \dots \quad \vdash p_n : \tau_n}{\vdash [x_{1\,\tau_1} \mapsto p_1, \cdots, x_{n\,\tau_n} \mapsto p_n] \text{ Env}}$$

FIGURE 3 – Définition des clôtures et des P-environnements

$$\frac{}{E \vdash x_\tau \Rightarrow E(x_\tau)}\ (Var) \text{ si } x_\tau \in \mathrm{dom}\ E \qquad \frac{}{E \vdash \dot{n} \Rightarrow \dot{n}}\ (Cst)$$

$$\frac{E \vdash u \Rightarrow \langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u', E'\rangle \quad E \vdash v \Rightarrow p \quad E'[f_{\sigma\to\tau} \mapsto \langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u', E'\rangle, x_\sigma \mapsto p] \vdash u' \Rightarrow p'}{E \vdash uv \Rightarrow p'}\ (App)$$

$$\frac{E[f_{\sigma\to\tau} \mapsto \langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u, E\rangle] \vdash v \Rightarrow p}{E \vdash \texttt{letrec}\ f_{\sigma\to\tau}(x_\sigma) = u\ \texttt{in}\ v \Rightarrow p}$$

$$\frac{E \vdash u \Rightarrow \dot{n}_1 \quad E \vdash v \Rightarrow \dot{n}_2}{E \vdash u \dot{+} v \Rightarrow \dot{n}}\ (\dot{+}) \text{ où } n = n_1 + n_2 \qquad \frac{E \vdash u \Rightarrow \dot{n}}{E \vdash \dot{-}u \Rightarrow \dot{-}\dot{n}}\ (\dot{-})$$

$$\frac{E \vdash u \Rightarrow \dot{0} \quad E \vdash v \Rightarrow p}{E \vdash \texttt{if}\, u = 0\ \texttt{then}\ v\ \texttt{else}\ w \Rightarrow p}\ (\texttt{if}_0) \qquad \frac{E \vdash u \Rightarrow \dot{n} \quad E \vdash w \Rightarrow p}{E \vdash \texttt{if}\, u = 0\ \texttt{then}\ v\ \texttt{else}\ w \Rightarrow p}\ (\texttt{if}_1) \text{ si } n \neq 0$$

FIGURE 4 – La sémantique à grands pas de PCF_V

dénotationnelle que nous avons donnée de PCF est en appel par nom. Voir l'exercice 21, et l'exercice 22. Notons qu'on n'utilise pas des environnements dans les jugements, mais des P-environnements.

Quel rapport entre les deux sémantiques ? Nous aimerions démontrer un théorème de correction, à savoir que si $E \vdash u \Rightarrow p$ est dérivable, alors $[\![u]\!]\,\rho = p\ldots$ sauf que ceci n'est pas bien défini. Par exemple, p est une P-valeur, pas une valeur. De plus, que vaut ρ ? Moralement, ρ, c'est E, mais ce ne sont pas des objets de même nature.

La solution est simple : nous allons démontrer que la sémantique opérationnelle est correcte *modulo* un changement de représentation. Nous allons montrer que si $E \vdash u \Rightarrow p$ est dérivable, alors $[\![u]\!]\,\rho = [\![p]\!]$, où $\rho = [\![E]\!]$. La sémantique dénotationnelle des valeurs est définie par :

— $[\![\dot{n}]\!] = n$, $n \in \mathbb{Z}$;

— $[\![\langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u, E\rangle]\!] = \mathrm{lfp}(F^{\rho}_{f_{\sigma\to\tau},x_\sigma,u})$, où $\rho = [\![E]\!]$ et où $F^{\rho}_{f_{\sigma\to\tau},x_\sigma,u}$ est définie en figure 2.

La sémantique dénotationnelle des environnements est définie par :

— $[\![E]\!] = \rho_0[x_{1\,\tau_1} \mapsto [\![p_1]\!], \cdots, x_{n\,\tau_n} \mapsto [\![p_n]\!]]$, lorsque $E = [x_{1\,\tau_1} \mapsto p_1, \cdots, x_{n\,\tau_n} \mapsto p_n]$, et où ρ_0 est un environnement arbitraire. (On ne cherchera jamais à consulter les valeurs de variables dans ρ_0.)

Ces deux définitions sont mutuellement récursives. En réalité, il faut les voir comme une définition par récurrence structurelle sur les dérivations du système de la figure 3.

Proposition 17 (Correction) *Si $E \vdash u \Rightarrow p$ est dérivable, alors $[\![u]\!]\,\rho = [\![p]\!]$, où $\rho = [\![E]\!]$.*

Démonstration. Par récurrence sur la dérivation π donnée. Nous traitons uniquement du cas de la règle (App). Dans ce cas, π est une dérivation de $E \vdash uv \Rightarrow p'$, et nous avons trois dérivations plus petites. En reprenant les notations de la figure 4, l'hypothèse de récurrence nous donne :

— $[\![u]\!]\,\rho = [\![\langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u', E'\rangle]\!] = \mathrm{lfp}(F^{\rho'}_{f_{\sigma\to\tau},x_\sigma,u'})$, où $\rho' = [\![E']\!]$,

— $[\![v]\!]\,\rho = [\![p]\!]$,

— $[\![u']\!]\,(\rho'[f_{\sigma\to\tau} \mapsto [\![u]\!]\,\rho, x_\sigma \mapsto [\![p]\!]]) = [\![p']\!]$,

où $\rho = [\![E]\!]$. Nous devons démontrer que $[\![uv]\!]\,\rho = [\![p']\!]$. Comme $[\![u]\!]\,\rho = \mathrm{lfp}(F^{\rho'}_{f_{\sigma\to\tau},x_\sigma,u'})$ est un point fixe, on a $[\![u]\!]\,\rho = F^{\rho'}_{f_{\sigma\to\tau},x_\sigma,u'}([\![u]\!]\,\rho)$. En expansant la définition de $F^{\rho'}_{f_{\sigma\to\tau},x_\sigma,u'}$, on obtient :

$$[\![u]\!]\,\rho = (V \in [\![\sigma]\!] \mapsto [\![u']\!]\,(\rho'[f_{\sigma\to\tau} \mapsto [\![u]\!]\,\rho, x_\sigma \mapsto V])).$$

En particulier,

$$\begin{array}{rcl} [\![uv]\!]\,\rho & = & [\![u]\!]\,\rho([\![v]\!]\,\rho) = [\![u]\!]\,\rho([\![p]\!]) \\ & = & [\![u']\!]\,(\rho'[f_{\sigma\to\tau} \mapsto [\![u]\!]\,\rho, x_\sigma \mapsto [\![p]\!]] = [\![p']\!]\,. \end{array}$$

Les autres cas sont plus simples, et laissés au lecteur. □

3.2 Adéquation : PCF$_V$

Comme pour IMP, la correction n'est qu'une implication. On aimerait aussi pouvoir démontrer l'*adéquation*, c'est-à-dire que si $[\![u]\!]\,\rho = [\![p]\!]$, où $\rho = [\![E]\!]$, alors $E \vdash u \Rightarrow p$ serait dérivable.

Ceci est faux ! Et ce, pour plusieurs raisons.

— La première est ce qui se passe lorsque u est d'un type autre que `int`, c'est-à-dire d'un type fonction. Il y a énormément de clôtures p qui ont la même sémantique $[\![p]\!]$. Par exemple, on peut coder l'addition de x et de y par $x\dot{+}y$, par $y\dot{+}x$, par $(x\dot{+}(-\dot{1}))\dot{+}(\dot{1}\dot{+}y)$, etc. Comme la sémantique opérationnelle à grands pas est déterministe, il est exclu que l'on puisse dériver $E \vdash u \Rightarrow p$ pour *toutes les implémentations* p d'une même fonction sémantique $[\![p]\!]$.
— Au type `int`, le problème ne se pose pas : il n'y a qu'une valeur p telle que $[\![p]\!] = n$ pour tout $n \in \mathbb{Z}$, à savoir $\dot{n}$. Et pourtant, la sémantique opérationnelle de PCF$_V$ n'est pas adéquate, même au type `int` : voir l'exercice 21.

Exercice 21 *On considère l'expression PCF u suivante :*

$$\begin{array}{l}\texttt{letrec } f_{\texttt{int}\to\texttt{int}}(x_{\texttt{int}}) = \dot{3} \texttt{ in}\\ \texttt{letrec } g_{\texttt{int}\to\texttt{int}}(x_{\texttt{int}}) = g_{\texttt{int}\to\texttt{int}}(x_{\texttt{int}}) \texttt{ in}\\ \qquad f_{\texttt{int}\to\texttt{int}}(g_{\texttt{int}\to\texttt{int}}\dot{0}).\end{array}$$

Montrer que $[\![u]\!]\,\rho = 3$ dans n'importe quel environnement ρ. Montrer que pourtant, on ne peut pas dériver $E \vdash u \Rightarrow 3$ pour aucun P-environnement E : en fait, il n'y a pas de dérivation de $E \vdash u \Rightarrow p$ pour aucun E et aucun p.

En déduire que la sémantique opérationnelle de PCF$_V$ n'est pas adéquate par rapport à la sémantique dénotationnelle de PCF.

Le problème ici, c'est que la sémantique dénotationnelle de PCF est en appel *par nom*. On peut corriger cela de deux façons différentes :
— En définissant une sémantique opérationnelle pour PCF en appel par nom. C'est ce que fait Plotkin [2] (modulo quelques changements de syntaxe). On obtient ainsi l'adéquation, par une technique de *relations logiques*.
— En définissant une sémantique dénotationnelle spécialisée au cas de l'appel par valeur. On aura de nouveau l'adéquation, par des techniques similaires. Voir l'exercice 23.

La sémantique dénotationnelle de PCF$_V$, c'est-à-dire en appel *par valeur*, est obtenue en opérant les modifications suivantes. D'abord, on change l'interprétation des types :

$$[\![\texttt{int}]\!]^V = \mathbb{Z} \qquad [\![\sigma \to \tau]\!]^V = [[\![\sigma]\!]^V \to [\![\tau]\!]] \qquad [\![\tau]\!] = [\![\tau]\!]^V_\perp$$

Chaque type τ a deux interprétations : $[\![\tau]\!]^V$ est le dcpo des *valeurs* de type τ, $[\![\tau]\!]$ est le dcpo des *calculs* de type τ. La seule différence est que $[\![\tau]\!]$ a un élément $\perp$ (non-terminaison) supplémentaire : $[\![\tau]\!]$ est le lifting de $[\![\tau]\!]^V$ (voir la section 2). L'ordre sur $[\![\texttt{int}]\!]^V = \mathbb{Z}$ est l'égalité. On notera bien que $[\![\sigma \to \tau]\!]^V$ a un plus petit élément, à savoir la fonction $(V \in [\![\sigma]\!]^V \mapsto \perp)$, que nous ne noterons *pas* $\perp$, pour la distinguer du plus petit élément de $[\![\sigma \to \tau]\!] = [\![\sigma \to \tau]\!]^V_\perp$.

Un environnement ρ est désormais une fonction totale qui à chaque variable x_τ associe un élément de $[\![\tau]\!]^V$ (et non de $[\![\tau]\!]$). La sémantique dénotationnelle de PCF$_V$ est donnée en figure 5, et définit $[\![u]\!]\,\rho$ pour toute expression de type τ comme un élément de $[\![\tau]\!]$ (pas $[\![\tau]\!]^V$: le calcul peut ne pas terminer).

Les seules petites différences entre les sémantiques dénotationnelles de PCF (figure 2) et de PCF$_V$ (figure 5) sont dans l'interprétation de l'application et du `letrec`. Nous réutilisons

$$
\begin{array}{rcl}
[\![x_\tau]\!]\,\rho & = & \rho(x_\tau) \\
[\![\dot{n}]\!]\,\rho & = & n \\
[\![uv]\!]\,\rho & = & \begin{cases} \bot & \text{si } [\![u]\!]\,\rho = \bot \text{ ou } [\![v]\!]\,\rho = \bot \\ [\![u]\!]\,\rho([\![v]\!]\,\rho) & \text{sinon} \end{cases} \\
[\![\texttt{letrec } f_{\sigma\to\tau}(x_\sigma) = u \texttt{ in } v]\!]\,\rho & = & [\![v]\!]\,(\rho[f_{\sigma\to\tau} \mapsto [\![\texttt{rec } f_{\sigma\to\tau}(x_\sigma) = u]\!]]) \\
[\![\texttt{rec } f_{\sigma\to\tau}(x_\sigma) = u]\!]\,\rho & = & \mathrm{lfp}^V(F^\rho_{f_{\sigma\to\tau},x_\sigma,u}) \\
 & & \text{où } F^\rho_{f_{\sigma\to\tau},x_\sigma,u}(\varphi) = (V \in [\![\sigma]\!]^V \mapsto [\![u]\!]\,(\rho[f_{\sigma\to\tau} \mapsto \varphi, x_\sigma \mapsto V])) \\
[\![u \dot{+} v]\!]\,\rho & = & \begin{cases} [\![u]\!]\,\rho + [\![v]\!]\,\rho & \text{si } [\![u]\!]\,\rho \neq \bot, [\![v]\!]\,\rho \neq \bot \\ \bot & \text{sinon} \end{cases} \\
[\![\dot{-} u]\!]\,\rho & = & \begin{cases} -\,[\![u]\!]\,\rho & \text{si } [\![u]\!]\,\rho \neq \bot \\ \bot & \text{sinon} \end{cases} \\
[\![\texttt{if } u = 0 \texttt{ then } v \texttt{ else } w]\!]\,\rho & = & \begin{cases} [\![v]\!]\,\rho & \text{si } [\![u]\!]\,\rho = 0 \\ [\![w]\!]\,\rho & \text{si } [\![u]\!]\,\rho \neq \bot, 0 \\ \bot & \text{si } [\![u]\!]\,\rho = \bot \end{cases}
\end{array}
$$

FIGURE 5 – Sémantique dénotationnelle de PCF_V

certaines notations, comme $F^\rho_{f_{\sigma\to\tau},x_\sigma,u}$ (qui est défini par la même formule, mais sur des espaces différents). Nous notons lfp^V l'opérateur « plus petit point fixe » dans le dcpo $[\![\sigma \to \tau]\!]^V$, pour le distinguer de l'opérateur de plus petit point fixe dans $[\![\sigma \to \tau]\!]$. On a $\mathrm{lfp}^V(F) = \sup_{n\in\mathbb{N}} F^n(\overline{\bot})$, où $\overline{\bot}$ est la fonction $V \in [\![\sigma]\!]^V \mapsto \bot$.

Exercice 22 *Montrer que ceci définit toujours une fonction qui est continue en ρ (adapter la démonstration de la proposition 16), et que la sémantique opérationnelle de PCF_V (figure 4) est correcte par rapport à la sémantique dénotationnelle de PCF_V (adapter la démonstration de la proposition 17).*

Donc en particulier, la sémantique opérationnelle de PCF_V est correcte à la fois pour la sémantique dénotationnelle de PCF et pour celle de PCF_V. L'adéquation ne fonctionne qu'avec PCF_V. En un sens, PCF « termine plus souvent » que PCF_V.

Exercice 23 (Adéquation, PCF_V) *Pour les plus courageux seulement ! On définit une famille de relations ternaires $\_ \models \_\ R_\tau\ \_$ et une famille de relations binaires $\_\ {R_\tau}^V\ \_$, indexées par les types. La relation $E \models u\ R_\tau\ V$ reliera des P-environnements E, des expressions u de type τ et des calculs $V \in [\![\tau]\!]$ de type τ. La relation $p\ {R_\tau}^V\ V$ reliera des P-valeurs p de type τ avec des valeurs $V \in [\![\tau]\!]^V$ de type τ. Voici la définition, par récurrence sur τ :*

- *$E \models u\ R_\tau\ V$ si et seulement si $V = \bot$, ou bien $V \in [\![\tau]\!]^V$ et il existe une P-valeur p de type τ telle que $E \vdash u \Rightarrow p$ est dérivable et $p\ R^V_\tau\ V$;*
- *$\dot{n}\ R^V_{\texttt{int}}\ m$ si et seulement si $m = n$;*

— $\langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u', E'\rangle\ R^V_{\sigma\to\tau}\ \varphi$ *(pour* $\varphi \in [[\![\sigma]\!]^V \to [\![\tau]\!]]$*) si et seulement si, pour toute P-valeur* p *de type* σ *et pour toute valeur* $V \in [\![\sigma]\!]^V$*, telles que* $p\ R^V_\sigma\ V$*, on a* $E'[f_{\sigma\to\tau} \mapsto \langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u', E'\rangle, x_\sigma \mapsto p] \models u'\ R_\tau\ \varphi(V)$.

On définira aussi la relation R *par* $E\ R\ \rho$ *si et seulement si pour toute variable* $x_\sigma \in \mathrm{dom}\ E$, $E(x_\sigma)\ R^V_\sigma\ \rho(x_\sigma)$*. Montrer que :*

— *la sémantique opérationnelle est déterministe : si* $E \vdash u \Rightarrow p_1$ *et* $E \vdash u \Rightarrow p_2$ *sont dérivables, alors* $p_1 = p_2$ *(ceci sert dans le point suivant) ;*

— $p\ R^V_\tau$*, qui est par définition l'ensemble des* $V \in [\![\tau]\!]^V$ *tels que* $p\ R^V_\tau\ V$*, est un fermé de Scott (voir l'exercice 10) ; de même,* $E \vdash u\ R_\tau$*, défini similairement, est aussi un fermé de Scott ;*

— $\langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u', E'\rangle\ R^V_{\sigma\to\tau}\ \overline{\perp}$*, pour toute clôture* $\langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u', E'\rangle$ *;*

— *en supposant que* $E'\ R\ \rho'$ *et que l'on sache que* $E'' \models u'\ R_\tau\ [\![u']\!]\,\rho''$ *pour tous* E'', ρ'' *tels que* $E''\ R\ \rho''$ *et* $\mathrm{fv}(u') \subseteq \mathrm{dom}\ E''$*, montrer que si* $\langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u', E'\rangle\ R^V_{\sigma\to\tau}\ \varphi$*, alors* $\langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u', E'\rangle\ R^V_{\sigma\to\tau}\ F^{\rho'}_{f_{\sigma\to\tau},x_\sigma,u'}(\varphi)$ *; en utilisant les remarques précédentes, en déduire que* $\langle \texttt{rec}\, f_{\sigma\to\tau}(x_\sigma) = u', E'\rangle\ R^V_{\sigma\to\tau}\ \mathrm{lfp}^V(F^{\rho'}_{f_{\sigma\to\tau},x_\sigma,u'})$ *;*

— *pour tout P-environnement* E*, pour tout environnement* ρ*, tels que* $E\ R\ \rho$*, pour toute expression* u *de type* τ *avec* $\mathrm{fv}(u) \subseteq \mathrm{dom}\ E$*,* $E \models u\ R_\tau\ [\![u]\!]\,\rho$*. Le cas le plus compliqué est celui du* `letrec`*, où l'on se servira de la remarque précédente.*

En déduire que si u *est une expression de type* `int` *sans variable libre et si* $[\![u]\!]\,\rho = n \neq \perp$ *(pour un* ρ *quelconque, noter que* u *n'ayant pas de variable libre,* $[\![u]\!]\,\rho$ *ne dépend pas de* ρ*), alors pour n'importe quel P-environnement* E*,* $E \vdash u \Rightarrow \dot{n}$ *est dérivable. C'est le résultat d'*adéquation *pour PCF*$_V$.

Qu'en est-il de l'abstraction complète ? Appelons contexte de type τ toute expression C de PCF de type `int`, avec une seule occurrence d'une variable spécifique notée $[]$, et de type τ. On note $C[u]$ l'expression obtenue en remplaçant $[]$ par l'expression u de type τ. Comme pour IMP, la correction et l'adéquation impliquent que si u et v sont deux expressions de type τ, sans variable libre, ayant la même sémantique dénotationnelle, alors ils sont contextuellement équivalents, au sens où $C[u]$ termine si et seulement si $C[v]$ termine.

Le langage est complètement abstrait si et seulement la réciproque est vrai elle aussi… et c'est **faux** tant pour PCF en appel par nom que pour PCF$_V$. Pour PCF en appel par nom, c'est encore Gordon Plotkin [2] qui a apporté la réponse :

— La clé est la fonction $por \in [\![\texttt{int} \to \texttt{int} \to \texttt{int}]\!]$ (pour la sémantique dénotationnelle en appel par nom, donc), appelée « ou parallèle », définie par :

$$por(1)(1) = 1 \quad por(0)(n) = por(n)(0) = 0 \text{ pour tout } n \in \mathbb{Z}_\perp$$
$$por(m)(n) = \perp \text{ dans tous les autres cas.}$$

On peut démontrer que por est effectivement une fonction continue, mais n'est pas définissable (il n'y a pas d'expression u sans variable libre de type $\texttt{int} \to \texttt{int} \to \texttt{int}$ en PCF telle que $[\![u]\!]\,\rho = por$), et qui n'est même pas sup de famille dirigée de définissables.

— En revanche, on peut définir un *goûteur de ou parallèle*, qui est une fonction $g : (\texttt{int} \to \texttt{int} \to \texttt{int}) \to \texttt{int}$ qui teste si son argument est un ou parallèle :

$$\texttt{letrec}\ g(f) = f\dot{1}\dot{1}\dot{-}\dot{1} = 0 \wedge f\dot{0}\Omega = 0 \wedge f\Omega\dot{0} = 0\ \texttt{in}\ g$$

$$
\begin{array}{rcl}
[\![x_\tau]\!]^\circ\,\rho\kappa & = & \kappa(\rho(x_\tau)) \\
[\![\dot{n}]\!]^\circ\,\rho\kappa & = & \kappa(n) \\
[\![uv]\!]^\circ\,\rho\kappa & = & [\![u]\!]^\circ\,\rho(f \in [\![\sigma \to \tau]\!]^\circ \mapsto [\![v]\!]^\circ\,\rho(f\kappa)) \\
[\![\texttt{letrec } f_{\sigma\to\tau}(x_\sigma) = u \texttt{ in } v]\!]^\circ\,\rho\kappa & = & [\![v]\!]^\circ\,(\rho[f_{\sigma\to\tau} \mapsto \mathrm{lfp}(T^\rho_{f_{\sigma\to\tau},x_\sigma,u})])\kappa \\
 & & \text{où } T^\rho_{f_{\sigma\to\tau},x_\sigma,u}(\varphi) = (\kappa \in [\![\tau]\!]^\perp \mapsto \\
 & & \qquad (V \in [\![\sigma]\!]^\circ \mapsto [\![u]\!]^\circ\,(\rho[f_{\sigma\to\tau} \mapsto \varphi, x_\sigma \mapsto V])\kappa)) \\
[\![u\dot{+}v]\!]^\circ\,\rho\kappa & = & [\![u]\!]^\circ\,\rho(m \in \mathbb{Z} \mapsto [\![v]\!]^\circ\,\rho(n \in \mathbb{Z} \mapsto \kappa(m+n))) \\
[\![\dot{-}u]\!]^\circ\,\rho\kappa & = & [\![u]\!]^\circ\,\rho(n \in \mathbb{Z} \mapsto \kappa(-n)) \\
[\![\texttt{if } u = 0 \texttt{ then } v \texttt{ else } w]\!]^\circ\,\rho\kappa & = & [\![u]\!]^\circ\,\rho(n \in \mathbb{Z} \mapsto \left\{ \begin{array}{ll} [\![v]\!]^\circ\,\rho\kappa & \text{si } n = 0 \\ [\![w]\!]^\circ\,\rho\kappa & \text{si } n \neq 0 \end{array} \right.)
\end{array}
$$

FIGURE 6 – La sémantique de PCF_V en passage à la continuation

On a omis les indices de types par souci de concision. On note aussi $a = 0 \wedge b = 0 \wedge c = 0$ pour $\texttt{if } a = 0 \texttt{ then if } b = 0 \texttt{ then if } c = 0 \texttt{ then } \dot{1} \texttt{ else } \Omega \texttt{ else } \Omega \texttt{ else } \Omega$, où Ω est un terme dont la sémantique est $\perp$. Par exemple, on peut définir Ω comme $\texttt{letrec } f_{\texttt{int}\to\texttt{int}}(x_{\texttt{int}}) = f_{\texttt{int}\to\texttt{int}}(x_{\texttt{int}}) \texttt{ in } f_{\texttt{int}\to\texttt{int}}(\dot{0}))$.

— On peut montrer que non seulement *por* n'est pas définissable, mais en fait le goûteur de ou parallèle est contextuellement équivalent à l'expression $\texttt{letrec } h(f) = h(f) \texttt{ in } h$ (qui ne termine jamais). Si *por* était définissable, disons par un terme `por`, le contexte $[]\ \texttt{por}$ distinguerait les deux.
En revanche, la sémantique dénotationnelle du goûteur de ou parallèle et celle de l'expression $\texttt{letrec } h(f) = h(f) \texttt{ in } h$ sont différentes : ce sont des fonctions qui, quand on les applique à *por*, retournent l'une 0, l'autre $\perp$. PCF n'est pas complètement abstrait.

Il en est de même pour PCF_V. L'idée pour le démontrer est que l'on peut coder l'appel par nom en appel par valeur. Les entiers paresseux de PCF deviennent des fonctions constantes retournant l'entier visé en PCF_V. Les détails sont laissés en exercice.

En revanche, PCF auquel on a ajouté l'opérateur `por` est complètement abstrait. C'est une démonstration un peu plus évoluée encore que celle de l'adéquation, et de nouveau, présentée en [2]. Le livre de Thomas Streicher [3] est une lecture recommandée sur ce sujet.

3.3 Sémantique par continuation

De même que pour IMP, on peut définir une sémantique de plus faibles préconditions, et de là, une sémantique par passage à la continuation. Le procédé est similaire. Nous allons donner directement le résultat… ou du moins une variante du résultat.

Ici, de nouveau, nous devrons prendre en compte les types, et une continuation au type τ sera un élément de $[[\![\tau]\!]^V \to Ans]$, où Ans est le dcpo pointé, fixé une fois pour toutes mais arbitraire, des réponses. Nous obtenons la sémantique de la figure 6, qui est toujours pour la variante en

appel par valeur PCF_V de PCF. Pour chaque expression u de type τ, chaque environnement ρ et chaque continuation $\kappa \in [[\tau]]^\circ \to Ans]$, $[\![u]\!]^\circ \rho\kappa$ sera un élément de Ans.

Attention, les types changent ! On a $[\![\texttt{int}]\!]^\circ = \mathbb{Z}$, mais $[\![\sigma \to \tau]\!]^\circ = [[\![\tau]\!]^\perp \to [\![\sigma]\!]^\perp]$, où $[\![\tau]\!]^\perp$, le domaine des continuations acceptant des objets de type τ, est égal à $[[\![\tau]\!]^\circ \to Ans]$. On implémente ainsi directement qu'une fonction de σ vers τ est un transformateur de prédicats (avec une sémantique en arrière), envoyant des prédicats sur les sorties, dans $[\![\tau]\!]^\perp$, vers des prédicats sur les entrées, dans $[\![\sigma]\!]^\perp$.

On peut lire la continuation κ comme « return ». Il sera utile aussi de lire les expressions comme $[\![u]\!]\,\rho(m \in \mathbb{Z} \mapsto [\![v]\!]\,\rho(n \in \mathbb{Z} \mapsto \kappa(m+n)))$ sous la forme intuitive :

> « Evaluer u et récupérer sa valeur m, puis évaluer v et récupérer sa valeur n, puis retourner $m+n$. »

L'exercice 24 propose une façon non standard de donner une sémantique à un langage tel que PCF avec des références. L'exercice 25 donne lui une application relativement standard des sémantiques par continuation à la gestion des exceptions.

Exercice 24 *On se donne pour chaque type τ un ensemble $Addr_\tau$ des adresses pointant vers des objets de type τ. On pose St le dcpo des* stores*, qui est celui des fonctions totales qui à chaque $a \in Addr_\tau$ associe un élément de $[\![\tau]\!]^\circ$, avec l'ordre usuel. Mini-Caml est le langage PCF_V plus deux constructeurs de type* `unit` *et* `ref` *et trois nouvelles constructions syntaxiques :*

- *pour chaque expression u de type* `ref` τ*, $!u$ est une expression de type τ ;*
- *pour toutes expressions u :* `ref` τ *et $v : \tau$, $u := v$ est une expression de type* `unit` *;*
- *pour toute expression $u : \tau$,* `ref` u *est une expression de type* `ref` τ*.*

On étend la sémantique par continuation de PCF_V en posant $[\![\texttt{unit}]\!]^\circ = \{\perp, \top\}$ avec $\perp < \top$, $[\![\texttt{ref}\ \tau]\!]^\circ = Addr_\tau$, et surtout en posant $Ans = [Mem \to Ans_0]$ pour un nouveau dcpo pointé Ans_0 de réponses dites pures. La sémantique des constructions déjà présentes en PCF_V reste définie par la figure 6, et la sémantique des nouvelles constructions est :

$$\begin{aligned}
[\![!u]\!]^\circ\,\rho\kappa &= [\![u]\!]^\circ\,\rho(\text{deref}\ \kappa) \\
[\![u := v]\!]^\circ\,\rho\kappa &= [\![u]\!]^\circ\,\rho(a \in Addr_\tau \mapsto [\![v]\!]^\circ\,\rho(\text{assign}\ a\ \kappa) \\
[\![\texttt{ref}\ u]\!]^\circ\,\rho\kappa &= [\![u]\!]^\circ\,\rho(\text{new}\ \kappa)
\end{aligned}$$

où :

$$\begin{aligned}
\text{deref}\ \kappa(a)(\mu) &= \kappa(\mu(a))(\mu) \\
\text{assign}\ a\ \kappa(V)(\mu) &= \kappa(\top)(\mu[a \mapsto V]) \\
\text{new}\ \kappa(V)(\mu) &= \begin{cases} \kappa(a)(\mu[a \mapsto V]) & \text{si } a = alloc_\tau(\mu) \neq \perp \\ \perp & \text{sinon} \end{cases}
\end{aligned}$$

où dans la dernière ligne $alloc_\tau : St \to Addr_{\tau\perp}$ est une fonction d'allocation, c'est-à-dire retournant ou bien $\perp$ ou bien une adresse de type τ hors du domaine du store passé en argument. On suppose ici $alloc_\tau$ continue, $Addr_\tau$ étant vu comme un dcpo avec l'ordre d'égalité.

Écrire une sémantique à grands pas de ce nouveau langage $\text{PCF}_V + St$, dans le style de celle de PCF_V, mais dont les jugements auront la forme $E, S \vdash u \Rightarrow p, S'$. La nouveauté est

$$
\begin{array}{rcl}
[\![x_\tau]\!]^\circ\,\rho(\kappa \mid \lambda) & = & \kappa(\rho(x_\tau)) \\
[\![\dot{n}]\!]^\circ\,\rho(\kappa \mid \lambda) & = & \kappa(n) \\
[\![uv]\!]^\circ\,\rho(\kappa \mid \lambda) & = & [\![u]\!]^\circ\,\rho(f \in [\![\sigma \to \tau]\!]^\circ \mapsto [\![v]\!]^\circ\,\rho(f(\kappa \mid \lambda) \mid \lambda) \mid \lambda) \\
[\![\texttt{letrec}\ f_{\sigma\to\tau}(x_\sigma) = u\ \texttt{in}\ v]\!]^\circ\,\rho(\kappa \mid \lambda) & = & [\![v]\!]^\circ\,(\rho[f_{\sigma\to\tau} \mapsto \mathrm{lfp}(T^\rho_{f_{\sigma\to\tau},x_\sigma,u})])(\kappa \mid \lambda) \\
 & & \text{où } T^\rho_{f_{\sigma\to\tau},x_\sigma,u}(\varphi) = ((\kappa \mid \lambda) \in [\![\tau]\!]^\perp \mapsto \\
 & & \qquad (V \in [\![\sigma]\!]^\circ \mapsto [\![u]\!]^\circ\,(\rho[f_{\sigma\to\tau} \mapsto \varphi, x_\sigma \mapsto V])(\kappa \mid \lambda))) \\
[\![u \dot{+} v]\!]^\circ\,\rho(\kappa \mid \lambda) & = & [\![u]\!]^\circ\,\rho(m \in \mathbb{Z} \mapsto [\![v]\!]^\circ\,\rho(n \in \mathbb{Z} \mapsto \kappa(m+n) \mid \lambda) \mid \lambda) \\
[\![\dot{-} u]\!]^\circ\,\rho(\kappa \mid \lambda) & = & [\![u]\!]^\circ\,\rho(n \in \mathbb{Z} \mapsto \kappa(-n) \mid \lambda) \\
[\![\texttt{if}\ u = 0\ \texttt{then}\ v\ \texttt{else}\ w]\!]^\circ\,\rho(\kappa \mid \lambda) & = & [\![u]\!]^\circ\,\rho(n \in \mathbb{Z} \mapsto \left\{ \begin{array}{ll} [\![v]\!]^\circ\,\rho\kappa & \text{si } n = 0 \\ [\![w]\!]^\circ\,\rho\kappa & \text{si } n \neq 0 \end{array} \right. \mid \lambda) \\
[\![\texttt{raise}\ E(u)]\!]^\circ\,\rho(\kappa \mid \lambda) & = & [\![u]\!]^\circ\,\rho(V \mapsto \lambda(E)(V) \mid \lambda) \\
[\![\texttt{try}\ u\ \texttt{with}\ E(x_\sigma) \Rightarrow v]\!]^\circ\,\rho(\kappa \mid \lambda) & = & [\![u]\!]^\circ\,\rho(\kappa \mid \lambda[E \mapsto (V \mapsto [\![v]\!]^\circ\,(\rho[x_\sigma \mapsto V])(\kappa \mid \lambda))])
\end{array}
$$

FIGURE 7 – La sémantique de PCF_V en passage à la continuation, avec exceptions

la présence des P-stores S *et* S'*, qui sont des fonctions partielles associant à chaque adresse* $a \in Addr_\tau$ *dans leur domaine une P-valeur de type* τ*. Montrer ensuite la correction de la sémantique dénotationnelle de* $\mathrm{PCF}_V + St$ *par rapport à cette sémantique opérationnelle.*

Exercice 25 *Les sémantiques par continuations permettent de coder, de façon naturelle, une forme d'exceptions dans le style de Caml. On suppose un ensemble* Exn_σ *de* noms d'exception *pour chaque type* σ*. Enrichissons la syntaxe et la sémantique de* PCF_V *comme suit :*

- *Il y a deux nouvelles constructions syntaxiques : si* u*,* v *sont des expressions de type* τ*,* x_σ *est une variable et* $E \in Exn_\sigma$*, alors* $\texttt{try}\ u\ \texttt{with}\ E(x_\sigma) \Rightarrow v$ *est une expression de type* τ*. La notation Caml* $\texttt{try}\ u\ \texttt{with}\ E_1(x_{1\,\sigma_1}) \Rightarrow v_1 \mid E_2(x_{2\,\sigma_2}) \Rightarrow v_2 \mid \cdots \mid E_n(x_{n\,\sigma_n}) \Rightarrow v_n$ *est vue comme une abréviation de* $\texttt{try}(\cdots\texttt{try}(\texttt{try}\ u\ \texttt{with}\ E_1(x_{1\,\sigma_1}) \Rightarrow v_1)\ \texttt{with}\ E_2(x_{2\,\sigma_2}) \Rightarrow v_2)\cdots\texttt{with}\ E_n(x_{n\,\sigma_n}) \Rightarrow v_n$*.*
  De plus, si $E \in Exn_\sigma$ *et* u *est une expression de type* σ*, alors* $\texttt{raise}\ E(u)$ *est une expression de type arbitraire.*
- *La sémantique prend maintenant non plus une continuation* κ *en argument mais un couple* $(\kappa \mid \lambda)$ *formé d'une continuation (« return »,* κ*) et d'une fonction* λ *qui à chaque nom d'exception* $E \in Exn_\sigma$ *associe une continuation* $\lambda(E) \in [\![\sigma]\!]^\perp$ *(quoi faire si l'exception* E *est lancée). Elle est donnée en figure 7. Ici vous devrez adapter la définition de* $[\![\sigma \to \tau]\!]^\circ$ *(ce ne sera plus* $[[\![\tau]\!]^\perp \to [\![\sigma]\!]^\perp]$*) de sorte que la sémantique ait un sens (notamment le cas de l'application* uv*).*

Écrire une sémantique à grands pas de ce nouveau langage $\mathrm{PCF}_V + Exn$*, dans le style de celle de* PCF_V*, mais dont les jugements auront la forme* $E \vdash u \Rightarrow paq$*, où* paq *est un* paquet*, c'est-à-dire soit une P-valeur* p *soit un paquet d'exception* $\texttt{throw}\ E(V)$*, où* $E \in Exn_\sigma$ *pour*

un certain type σ et V est une P-valeur de type σ. (Il est fortement recommander de s'inspirer de la sémantique de C-- avec exceptions.) Montrer ensuite la correction de la sémantique dénotationnelle de PCF$_V$ + Exn par rapport à cette sémantique opérationnelle.

Références

[1] G. Kerneis. Continuation-Passing C : Program Transformations for Compiling Concurrency in an Imperative Language Thèse de doctorat, université Paris Diderot, 2012.

[2] G. D. Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(1) :223–255, 1977.

[3] T. Streicher Domain-Theoretic Foundations of Functional Programming. World Scientific, 2006.