# Sémantique opérationnelle

Pierre-Louis Curien (gdt Modélisation et Vérification, LIAFA)

2 avril 2001

## 1 Introduction

Dans ces notes, on expose, dans le cadre de divers langages fonctionnels ou extensions de langages fonctionnels, diverses notions:

- de **convergence** – comment spécifier les règles d'évaluation;
- d'**équivalences observationnelles** de programmes dérivées de cette notion de convergence – on fera apparaître des liens avec la notion de *bisimulation*, et on donnera des illustrations dans un langage fonctionnel qui permet de programmer des calculs de suites infinies;
- de **complète adéquation** – quête de modèles dénotationnels induisant exactement l'équivalence observationnelle; on commentera quelques succès dans ce domaine;
- de **machine abstraite**, spécification plus concrète des règles d'évaluation – on fera apparaître un lien avec le *calcul des séquents*. De même que dans le cochon on mange tout, dans une machine abstraite on type tout.

## 2 $\lambda$-calcul sans types

Référence: Full abstraction in the lazy lambda-calculus, Samson Abramsky and Luke Ong, Information and Computation 105: 159-267 (1993).

Sémantique *à grands pas* ($V$ est une abstraction):

$$\frac{M \Downarrow \lambda x.P \quad P[x \leftarrow N] \Downarrow V}{MN \Downarrow V} \qquad \frac{}{\lambda x.M \Downarrow \lambda x.M}$$

Sémantique *à petits pas* ($P\vec{R}$ est une abréviation de $PR_1 \ldots R_n$):

$$(\lambda x.M)N\vec{N} \rightarrow M[x \leftarrow N]\vec{N}$$

On remarque qu'un terme clos $M$ est en forme normale pour $\to$ ssi $M = \lambda x.N$. On pose $M \downarrow V$ ssi $M \to^\star V$.

**Proposition 2.1**

$$M \Downarrow V \Leftrightarrow M \downarrow V$$

PREUVE. Un terme clos est soit de la forme $\lambda x.M$, auquel cas l'énoncé est trivial, soit de la forme $(\lambda x.M)N\vec{N}$. On prouve $\Rightarrow$ par récurrence sur la taille de la preuve que $(\lambda x.M)N\vec{N} \Downarrow V$. Cette preuve contient strictement une preuve de $M[x \leftarrow N] \Downarrow V'$. On peut donc supposer par récurrence que $M[x \leftarrow N] \to^\star V'$, et donc $(\lambda x.M)N \to^\star V'$. Or si $P \to^\star Q$ alors $P\vec{R} \to^\star Q\vec{R}$. Donc $(\lambda x.M)N\vec{N} \to^\star V'\vec{N}$. On se sert ensuite de la propriété facile uivante:

(†) Si $M\vec{R} \Downarrow$, alors $M \Downarrow V$, et si $M' \Downarrow V$, alors $M'\vec{R} \Downarrow$. Si de plus la preuve de $M' \Downarrow V$ est plus courte que celle de $M \Downarrow V$, alors la preuve de $M'\vec{R} \Downarrow$ est plus courte que celle de $M\vec{R} \Downarrow$.

Donc $V'\vec{N} \Downarrow V$, et l'on peut appliquer la réccurence, soit $V'\vec{N} \to^\star V$, et donc $(\lambda x.M)N\vec{N} \to^\star V$.

Réciproquement, on raisonne par récurrence sur la longueur de la réduction $(\lambda x.M)N\vec{N} \to^\star V$. On peut donc supposer par récurrence que $M[x \leftarrow N]\vec{N} \Downarrow V$, et l'on conclut par la propriété (†). CQFD

On donne trois notions de **préordre observationnel** (Abramsky):

- Le préordre **contextuel** est défini commme suit (syntaxe des contextes: $M ::= [\,] \parallel x \mid MN \parallel \lambda x.M$):

$$M \preceq_{cont} N \quad \text{ssi} \quad \forall C \;\; C[M] \Downarrow \;\Rightarrow\; C[N] \Downarrow$$

- Le préordre **applicatif** est défini comme suit:

$$M \preceq_{app} N \quad \text{ssi} \quad \forall \vec{P} \;\; M\vec{P} \Downarrow \;\Rightarrow\; N\vec{P} \Downarrow$$

- La **similarité** est définie par le plus grand point fixe suivant:

$$M \preceq_{sim} N \quad \text{ssi} \quad M \Downarrow \lambda x.P \Rightarrow \left\{ \begin{array}{l} N \Downarrow \lambda x.Q \text{ et} \\ \forall R \;\; P[x \leftarrow R] \preceq_{sim} Q[x \leftarrow R] \end{array} \right.$$

**Théorème 2.2** *On a pour tous $M, N$:*

$$M \preceq_{cont} N \quad \Leftrightarrow \quad M \preceq_{app} N \quad \Leftrightarrow \quad M \preceq_{sim} N$$

PREUVE. $(M \preceq_{cont} N \Rightarrow M \preceq_{app} N)$ . Prendre $C = [\,]\vec{P}$.

$(M \preceq_{app} N \Rightarrow M \preceq_{cont} N)$ : Cette implication porte le nom de *Lemme du Contexte*. On utilise $\downarrow$ au lieu de $\Downarrow$ (cf. proposition 2.1). Soit donc $M \preceq_{app} N$ et $C[M] \downarrow V$ On procède par récurrence "lexicographique" sur la longueur de $C[M] \to^\star V$, puis par récurrence sur les cas:

1. $C = \lambda x.C_1$. Ce cas est trivial.

2. $C = [\,]$. Ce cas est également trivial: par définition de $\preceq_{app}$, $M \Downarrow$ implique $N \Downarrow$.

3. $C = (\lambda x.C_1)C_2\vec{C'}$. On pose $C_3 = C_1[x \leftarrow C_2]$. On applique la récurrence avec $C_3\vec{C'}$, d'où $C_3[N]C'[\vec{N}] \downarrow V$, puis $C[N] \downarrow V$ car $((\lambda x.C_1)C_2)[N] \rightarrow C_3[N]$.

4. $C = [\,]C_2\vec{C'}$. On pose $C_1 = MC_2\vec{C'}$. Comme $C_1[M] = C[M]$, la longueur de $C_1[M] \rightarrow^\star V$ est la même que celle de $C[M] \rightarrow^\star V$, et $C_1$ ayant la forme du cas précédent, on peut appliquer la réccurrence à $C_1$. Donc $MC_2[N]C'[\vec{N}] \rightarrow^\star V$. Mais puisque $M \preceq_{app} N$, on déduit $NC_2[N]C'[\vec{N}] \rightarrow^\star V$, soit $C[N] \rightarrow^\star V$.

$(M \preceq_{app} N \Rightarrow M \preceq_{sim} N)$ : Par coinduction, il suffit de prouver que, pour $S_0 = \{(M,N) \mid M \preceq_{app} N\}$ et $F$ défini par

$$F(S) = \{(M,N) \mid M \Downarrow \lambda x.P \Rightarrow N \Downarrow \lambda x.Q \text{ et } (\forall\, R\ (P[x \leftarrow R], Q[x \leftarrow R]) \in S)\}\ ,$$

on a $S_0 \subseteq F(S_0)$. Soit $(M,N) \in S_0$ et supposons que $M \Downarrow \lambda x.P$. Comme $M \preceq_{app} N$ on a (vecteur d'arguments vide) $N \Downarrow \lambda x.Q$. Il faut montrer de plus que $(P[x \leftarrow R], Q[x \leftarrow R]) \in S_0$, i.e. $P[x \leftarrow R] \preceq_{app} Q[x \leftarrow R]$. Soit $\vec{R}$ tel que $P[x \leftarrow R]\vec{R} \Downarrow$. On en déduit par la propriété (†) que $MR\vec{R} \Downarrow$. Donc $NR\vec{R} \Downarrow$ par définition de $\preceq_{app}$, et $Q[x \leftarrow R]\vec{R} \Downarrow$, encore par (†)..

$(M \preceq_{sim} N \Rightarrow M \preceq_{app} N)$ Supposons que $M \preceq_{sim} N$ et $M\vec{R} \Downarrow$, avec $\vec{R}$ de longueur $n$. On raisonne par récurrence sur la longueur de $\vec{R}$. Plus précisément, on pose $M \preceq_{app\,n} N$ si pour tout vecteur $\vec{P}$ de longueur $n$ on a $M\vec{P} \Downarrow \Rightarrow N\vec{P} \Downarrow$. Si $n = 0$, par définition de $\preceq_{sim}$, on a $N \Downarrow$. Si $\vec{R} = R\vec{R'}$, on a a fortiori $M \Downarrow \lambda x.P$. Donc $N \Downarrow \lambda x.Q$ et $P[x \leftarrow R] \preceq_{sim} Q[x \leftarrow R]$, par définition de $\preceq_{sim}$. Par réccurence, on a $P[x \leftarrow R] \preceq_{app\,n-1} Q[x \leftarrow R]$. Or on a $P[x \leftarrow R]\vec{R} \Downarrow$, par la propriété (†). Par définition de $\preceq_{app\,n-1}$, on a alors $Q[x \leftarrow R]\vec{R} \Downarrow$, dont on déduit que $NR\vec{R} \Downarrow$. CQFD

**Remarque 2.3** *La similarité est beaucoup plus qu'une reformulation du préordre applicatif, car elle ouvre au* raisonnement co-inductif*:*

*Si $F$ est monotone, alors $F$ a un plus grand point fixe, noté $\nu X.F(X)$, et*

$$\textit{(co-induction)} \quad S \subseteq F(S) \ \Rightarrow\ S \subseteq \nu X.F(X)$$

# 3 PCF

Référence (ainsi que pour les sections 5 et 6): Domains and Lambda-calculi, Roberto Amadio and Pierre-Louis Curien, Cambridge University Press (1998).

On passe à un cadre typé, On suppose qu'il y a des types de base, et des types de fonctions:

$$A ::= \mathit{bool} \parallel \mathit{int} \parallel A \to A$$

On ajoute au langage les constantes correspondantes de type *bool* et *int*, et d'autres: successeur, prédécesseur, test à zéro, conditionnelle, définition récursive de fonction.

On désigne un type de base par le nom générique $\kappa$. On désigne les constantes de type $\kappa$ par le nom générique $v$.

La convergence (à petits pas) est maintenant définie seulement sur les termes (clos) *de type de base*. On a maintenant qu'un terme clos $M$ de type de base est en forme normale pour $\to$ ssi $M = v$. On pose $M \Downarrow v$ ssi $M \to^\star v$.

On redéfinit $\preceq_{cont}$, $\preceq_{app}$ et $\preceq_{sim}$:

- Préordre contextuel:

$$M \preceq_{cont} N \quad \text{ssi} \quad \forall C \text{ de base} \quad C[M] \Downarrow \Rightarrow C[N] \Downarrow$$

- Préordre applicatif:

$$M \preceq_{app} N \quad \text{ssi} \quad (\forall \vec{P} \text{ tel que } M\vec{P} : \kappa \; et N\vec{P} : \kappa \; (M\vec{P} \Downarrow \; \Rightarrow \; N\vec{P} \Downarrow))$$

- Similarité:

$$M \preceq_{sim} N \quad \text{ssi} \quad \begin{cases} \forall R \;\; MR \preceq_{sim} NR & (M, N : A \to B) \\ M \Downarrow v \Rightarrow N \Downarrow v & (M, N : \kappa) \end{cases}$$

**Théorème 3.1** *On a pour tous $M, N$:*

$$M \preceq_{cont} N \quad \Leftrightarrow \quad M \preceq_{app} N \quad \Leftrightarrow \quad M \preceq_{sim} N$$

Preuve. Pas essentiellement différente de celle du théorème 2.2. La preuve de $\preceq_{app} = \preceq_{sim}$ est même plus simple. CQFD

**Remarque 3.2** *On a gardé les mêmes noms pour simplifier, mais ce ne sont plus les mêmes notions d'équivalence. Ainsi, si $\Omega$ est un programme divergent, on a:*

- *$\lambda x.\Omega x \not\preceq_{sim} \Omega$ avec la définition d'Abramsky (car $\lambda x.\Omega x \Downarrow$ alors que $\Omega$ diverge);*
- *$\lambda x.\Omega x \preceq_{sim} \Omega$ avec la définition ci-dessus. Ces deux termes, une fois appliqué à $R$, ne peuvent être distingués.*

# 4 Listes infinies paresseuses

Référence: Bisimilarity as a Theory of Functional Programming. Mini-course, Andrew D. Gordon.
Electronically: http://www.research.microsoft.com/ adg/Publications/details.html

Pour avoir de vrais exemples concrets d'équivalence de programmes, on ajoute les listes (potentiellement infinies), ou suites. Le système de types est le suivant:

$$A ::= \mathit{bool} \parallel \mathit{int} \parallel \mathit{list}(A) \mid A \to A$$

La convergence est définie sur les termes (clos) de type de base *et de type liste*:

- Un terme clos $M$ de type de base est en forme normale pour $\to$ ssi $M = v$. On pose $M \Downarrow v$ ssi $M \to^\star v$ (comme plus haut).
- Un terme clos $M$ de type $\mathit{list}(A)$ est en forme normale, ou plutôt *canonique*, pour $\to$ ssi $M = (M_1, M_2)$. On pose

  $$M \Downarrow (M_1, M_2) \quad \text{ssi} \quad M \to^\star (M_1, M_2)$$

  Noter que $M_1$, et surtout $M_2$ sont non évalués – listes paresseuses, e.g.

  $$\mathit{pair}\ n = (2n) :: (\mathit{pair}\ (n+1))$$

On a encore l'équivalence entre trois relations $\preceq_{\mathit{cont}}, \preceq_{\mathit{app}}$ et $\preceq_{\mathit{sim}}$:

- Préordre contextuel:

  $$M \preceq_{\mathit{cont}} N \quad \text{ssi} \quad \forall C\ \mathit{debase}\ \ C[M] \Downarrow \Rightarrow C[N] \Downarrow$$

- Préordre applicatif: comme plus haut, on peut se contenter de contextes d'une forme particulière – moralement, ce sont:$[\,]R$, $\mathit{hd}[\,]$ et $\mathit{tl}[\,]$; mais le langage de Gordon n'a pas $\mathit{hd}$ et $\mathit{tl}$, mais une opération $\mathit{scase}$ telle que $\mathit{scase}(a :: l, b, c) \to \mathit{cab}$ qui permet de récupérer des variantes *totales* de $\mathit{hd}$ et $\mathit{tl}$ en prenant respectivement $c = \lambda xy.x$ et $c = \lambda xy.y$ – détails omis;

- Similarité:

$$M \preceq_{\mathit{sim}} N \quad \text{ssi} \quad \begin{cases} \forall R\ \ MR \preceq_{\mathit{sim}} NR & (M, N : A \to B) \\ M \Downarrow (M_1, M_2) \Rightarrow \begin{cases} N \Downarrow (N_1, N_2), \\ M_1 \preceq_{\mathit{sim}} N_1 \text{ et} \\ M_2 \preceq_{\mathit{sim}} N_2 \end{cases} & (M : N : \mathit{list}(A)) \\ M \Downarrow v \Rightarrow N \Downarrow v & (M, N : \kappa) \end{cases}$$

Il est facile de voir que la similarité peut être ici formulée en termes de système de transition étiqueté (automate). On pose:

$$\begin{array}{ll} M \stackrel{[]R}{\longmapsto} MR & (M : A \to B) \\ M \stackrel{hd}{\longmapsto} M_1 & (M : list(A), M \Downarrow (M_1, M_2) \\ M \stackrel{tl}{\longmapsto} M_2 & (M : list(A), M \Downarrow (M_1, M_2) \\ M \stackrel{v}{\longmapsto} \mathbf{0} & (M : \kappa, M \Downarrow v) \end{array}$$

**Proposition 4.1** *La relation $\preceq_{sim}$ est le plus grand point fixe de l'équation suivante:*

$$M \preceq_{sim} N \Leftrightarrow \forall a \ (M \stackrel{a}{\longmapsto} M' \ \Rightarrow \ N \stackrel{a}{\longmapsto} N' \ et \ M' \preceq_{sim} N')$$

PREUVE. C'est une simple reformulation de la définition de $\preceq_{sim}$. CQFD

Voici des exemples d'équivalence observationnelle qui sont prouvables par coinduction.

Soit

$$\begin{array}{l} map\ f\ (a_1 :: a_2 :: \ldots :: a_n :: \ldots) = (fa_1) :: (fa_2) :: \ldots :: (fa_n) :: \ldots \\ iterate\ f\ (a_1 :: a_2 :: \ldots :: a_n :: \ldots) = a_1 :: (fa_2) :: \ldots :: (f^{n-1}a_n) :: \ldots \\ filter\ f\ (a_1 :: \ldots :: a_n :: \ldots) = a_{i_1} :: \ldots :: a_{i_n} :: \ldots \quad (\{i \mid f(i) = true\} = \{i_1, \ldots, i_n, \ldots\}) \end{array}$$

On a:

$$\begin{array}{lll} (iterate\ f\ (fx)) & =_{sim} & (map\ f\ (iterate\ f\ x)) \\ (filter\ f) \circ (map\ g) & =_{sim} & (map\ g) \circ (filter\ (f \circ g)) \end{array}$$

# 5 Adéquation et adéquation complète

Référence: Domains and Lambda-calculi, Roberto Amadio and Pierre-Louis Curien, Cambridge University Press

On a vu plusieurs langages, et pour chacun une notion de convergence $\Downarrow$. On souhaite qualifier un modèle, une *sémantique dénotationnelle*, par rapport à cette notion de convergence.

Complétude par rapport à la convergence (Computational Adequacy):

$$(CA) \quad M \Downarrow \quad \Leftrightarrow \quad [\![M]\!] \neq \bot$$

Cette propriété est vérifiée pour la plupart des notions de convergence et de modèle. L'implication $\Rightarrow$ est la conséquence de la validité des règles de la sémantique opérationnelle dans le modèle, et de ce que si $M \Downarrow$, alors $[\![M]\!] \neq \bot$. L'implication contraire est difficile, et se montre par une technique très répandue pour le $\lambda$-calcul, la *réalisabilité*.

Mais ceci n'est qu'une étape. Pour un modèle satisfaisant $(CA)$, on souhaite comparer l'(in)égalité induite par la sémantique opérationnelle et celle induite par la sémantique dénotationnelle:

$$M \preceq_{cont} N \quad versus \quad [\![M]\!] \leq [\![N]\!]$$

L'implication

$$M \preceq_{cont} N \quad \Leftarrow \quad [\![M]\!] \leq [\![N]\!]$$

est appelée **adéquation** et est conséquence de $(CA)$. Car $(CA)$ permet de reformuler $\preceq_{cont}$ en termes de sémantique dénotationnelle seulement:

$$M \preceq_{cont} N \quad \Leftrightarrow \quad \forall C \ ([\![C[M]]\!] \neq \bot \Rightarrow [\![C[N]]\!] \neq \bot)$$

La compositionalité de la sémantique dénotationnelle garantit que $[\![M]\!] \leq [\![N]\!] \Rightarrow [\![C[M]]\!] \leq [\![C[N]]\!]$. Donc a fortiori, si $[\![C[M]]\!] \neq \bot$, alors $[\![C[N]]\!] \neq \bot$.

L'autre implication est difficile. Lorsqu'elle est vérifée, on dit que le modèle est complètement adéquat.

**Remarque 5.1** *En logique, la complétude part du modèle et va vers la syntaxe: si A est valide, alors A est prouvable. Si la terminologie a un sens, on est donc conduit à penser à la sémantique opérationnelle comme au modèle et à la sémantique dénotationelle comme à la syntaxe. Et en effet, les modèles peuvent être présentés syntaxiquement:*

$[\![M]\!]$ = *l'ensemble des types que M peut recevoir dans un système de types*

*C'est la théorie des* types avec intersections. *La "vraie" sémantique est donc celle induite par l'équivalence contextuelle.*

Ce qui est vraiment difficile/intéressant, ce n'est pas simplement de construire un modèle complètement adéquat. On peut le faire par force brute: prendre un modèle de termes, le quotienter par l'équivalence observationnelle. C'est d'ailleurs souvent par ce genre de force brute et souvent triviale que l'on prouve des théorèmes de complétude en logique.

C'est de construire ce modèle *par d'autres moyens.* C'est un peu vague. Pour préciser, on peut vouloir que cette construction offre des moyens nouveaux de prouver que deux termes sont équivalents ou non-équivalents. Il faut donc que l'on puisse faire des preuves dans le modèle dénotationnel. Ce n'est clairement pas le cas si le modèle est obtenu par "collapse extensionnel", qui est la version mathématique de $\preceq_{app}$.

# 6 Complète adéquation pour les langages PCF, SPCF, IA

Référence pour Idealized Algol (IA): Linearity, Sharing and State: A Fully Abstract Game Semantics for Idealized ALGOL with Active Expressions, Samson Abramsky and Guy McCusker, in O'Hearn and Tennent (eds.), Algol-like languages, Birkhauser 1997.
Electronically: http://www.cogs.susx.ac.uk/users/guym/papers.html.

## 6.1 PCF

Dans le cas de PCF, un résultat négatif de Loader indique que *l'équivalence observationnelle n'est pas décidable sur les types finis* (construits à partir de *bool* et $\rightarrow$). Un quotient "aveugle" est donc inévitable pour arriver à la complète adéquation. Néanmoins, en 1994, les sémantiques de jeux (Abramsky-Jagadeesan-Malacaria, Hyland-Ong) ont permis de présenter le modèle de termes (avant quotient observationnel) d'une manière indépendante de la syntaxe. Le résultat principal de Hyland-Ong est le suivant:

bijection entre **arbres de Böhm** pour PCF et stratégies $\left\{ \begin{array}{l} \textbf{innocentes} \text{ et} \\ \textbf{bien parenthésées} \end{array} \right.$

(cf. exposé Danos). Si on y tient absolument, on peut dire que *le* modèle complètement adéquat (Milner a montré qu'il était unique) est le quotient observationnel du modèle de jeux, mais ce qui est plus important, c'est cette caractérisation des termes (arbres infinis, plus exactement) en forme normale, qui est d'autant plus intéressante qu'elle a permis de lancer un programme de classification. L'output le plus intéressant a été "la perte de l'innocence".

## 6.2 IA

La relaxation de l'innocence (en gardant le bon parenthésage) conduit à un modèle complètement adéquat de IA, *sans quotient*. On ajoute comme types de base le type des commandes et celui des variables. Le langage comprend le séquencement de commandes, l'affectation $x ::= e$, la lecture *deref* $x$, la déclaration de variables (impératives locales). L'idée est de modéliser une variable locale par des coups des joueurs (lecture, écriture); la valeur lue dépend crucialement de l'histoire.

**Théorème 6.1 (Abramsky-McCusker (1996))** *Le modèle des stratégies bien parenthésées est complètement adéquat pour IA.*

PREUVE. C'est le même schéma de preuve que pour le théorème 6.2. Notons que la stratégie $a$ dans cette preuve n'est pas en général innocente. Cela tient à ce que même si $c_1\ v_1\ \ldots\ c_n\ v_n$ est une vue pour Joueur, ce n'en est pas une pour Opposant, qui est Joueur dans $A \to \kappa$. CQFD

## 6.3 SPCF

La relaxation de l'autre condition (bon parenthésage) (en gardant l'innocence) permet de modéliser des opérateurs de contrôle (tels que *catch* et *throw*). Mais cela ne conduit pas directement à un modèle complètement adéquat: l'épouvantail du quotient observationnel est toujours là. Mais il se trouve qu'une autre variante de jeux, bien antérieure (et non formulée en termes de jeux, mais déjà clairement en termes d'interaction), permet d'avoir l'adéquation complète *sans quotient*: c'est le modèle des **algorithmes séquentiels** de Berry-Curien (1978). La différence essentielle avec les sémantiques de jeux "anglaise" de 1994, c'est le traitement de l'exponentielle $!A$: dans les algorithmes séquentiels, on ne répète pas les questions, et $[\![bool \to bool]\!]$ est de cardinal fini. (Notons que la répétition des questions est utilisée de façon essentielle pour modéliser le comportement d'une variable impérative.)

Le langage SPCF est PCF auquel on ajoute *catch* avec les règles (simplifiées ici):

$$catch(\lambda x_1 \ldots x_n.x_i M_1 \ldots M_m) = i - 1$$
$$catch(\lambda x_1 \ldots x_n.m) = n + m$$

**Théorème 6.2 (Cartwright-Curien-Felleisen (1994))** *Le modèle des algorithmes séquentiels est complètement adéquat pour SPCF.*

PREUVE. On montre:

(1) que tout algorithme séquentiel compact (c'est-à-dire fini) $a$ est définissable, i.e. qu'il existe $M$ tel que $[\![M]\!] = a$;

(2) que si $b$ et $c$ sont deux algorithmes séquentiels de type $A$ tels que $b \not\leq c$, alors on peut trouver un algorithme séquentiel de type $A \to \kappa$ tel que

$$a \cdot b = v \quad \text{et} \quad a \cdot c \neq v$$

Ici, l'opération $a \cdot b$ est l'application, définie par interaction entre $a$ (contre-stratégie) et $b$ (stratégie). Si $c_1\ v_1\ \ldots\ c_n\ v_n$ est une position qui est dans $b$ et non dans $c$, on peut définir $a$ comme suit:

$$
\begin{array}{lcl}
A & \to & \kappa \\
 & & \\
 & & ? \\
c_1 & & \\
v_1 & & \\
\vdots & & \\
c_n & & \\
v_n & & \\
 & & v
\end{array}
$$

On termine comme suit: Si $[\![M]\!] \not\leq [\![N]\!]$, alors par (2) il existe $a$ tel que $a.[\![M]\!] = v$ et $a \cdot [\![N]\!] \neq v$. Par (1), il existe $P$ tel que $a = [\![P]\!]$. On a donc $[\![PM]\!] = v$ et $[\![PN]\!] \neq v$. On conclut que $M \not\leq_{cont} N$, en prenant $C = P[]$. CQFD

# 7 Machines abstraites et calcul des séquents

Référence: The duality of computation, Pierre-Louis Curien and Hugo Herbelin, ICFP 2000.
Electronically: http://www.pps.jussieu.fr/ curien.

On revient au $\lambda$-calcul (typé). On veut spécifier une sémantique opérationnelle qui gère explicitement les appels récursifs implicites dans la spécification à grands pas donnée au début de ce texte. La solution est connue, il faut une pile:

$$
\begin{array}{lllll}
(NL) & \langle MN|S\rangle & \to & \langle M|N :: S\rangle \\
(\to) & \langle \lambda x.M|N :: S\rangle & \to & \langle M[x \leftarrow N]|S\rangle
\end{array}
$$

Les termes ont un type. On va aussi donner un type aux piles, et aux états $\langle M|S\rangle$. Dans une tel état, la pile représente un contexte (ou une continuation) qui attend le résultat de l'evaluation de $M$. D'où l'idée de jugements de la forme:

$$\Gamma \mid S : A \vdash \Delta$$

Ce qui importe, c'est le type du résultat que $S$ attend *en entrée* pour continuer le calcul. Pour plus de symétrie, au lieu des jugements de typage habituels $\Gamma \vdash M : A$ pour les termes, on adopte $\Gamma \vdash M : A \mid \Delta$, et en effet, c'est ce typage qui convient pour des extensions du $\lambda$-calcul avec des opérateurs de contrôle (cf. plus haut). Logiquement, le passage d'une formule à droite de $\vdash$ à plusieurs formules à droite de $\vdash$ correspond à la levée de la restriction au raisonnement intuitionniste. Mais nous laissons cela ici, cela nous entrainerait trop loin. Il reste à typer un état: c'est la *règle de coupure*:

$$\frac{\Gamma \vdash M : A \mid \Delta \qquad \Gamma \mid S : A \vdash \Delta}{\langle M|S\rangle : (\Gamma \vdash \Delta)}$$

La coupure, c'est la notion de lemme. Le calcul des séquents a été conçu autour de cette notion. Une preuve sans coupure est un formidable outil, qui sert notamment à prouver la cohérence logique d'un système formel. On montre d'abord que toute formule prouvable l'est par une preuve sans coupure. Les preuves sans coupure ont de remarquables propriétés, en particulier celle de la *sous-formule*: toute formule apparaissant dans la preuve doit être dans la conclusion. Aucun moyen alors de prouver un atome tel que $0 = 1$. D'où l'importance en logique de l'élimination des coupures.

Typons maintenant l'opération $N :: S$ d'empilement. Puisque $\langle MN|S\rangle$ est un état, on doit avoir:

$$\Gamma \vdash M : A \to B \,|\, \Delta \qquad \Gamma \vdash N : A \,|\, \Delta \qquad \Gamma \,|\, S : B \vdash \Delta$$

De même, puisque $\langle M|N :: S\rangle$ est un état, on doit avoir:

$$\Gamma \,|\, (N :: S) : A \to B \vdash \Delta$$

On vient de synthétiser la règle suivante:

$$\frac{\Gamma \vdash N : A \,|\, \Delta \qquad \Gamma \,|\, S : B \vdash \Delta}{\Gamma \,|\, (N :: S) : A \to B \vdash \Delta}$$

C'est la règle d'introduction à gauche de l'implication en calcul des séquents:

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}$$

Le calul des séquents n'a que des règles d'introduction, à droite et à gauche, pour les connecteurs logiques. La déduction naturelle, qui correspond exactement au $\lambda$-calcul typé, a des règles d'introduction (à droite) et d'élimination (à droite). On peut donc voir la règle $(NL)$ comme une règle de transfert de la déduction naturelle vers le calcul des séquents (les lettres $N$ et $L$ sont associées traditionellement et respectivemnt à ces systèmes).

Considérons maintenant la règle $(\to)$. Logiquement, en oubliant les termes, elle correspond à transformer une preuve se terminant ainsi (où l'on retrouve en haut, de gauche à droite, $M, N$ et $S$, respectivement):

$$\frac{\dfrac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta} \qquad \dfrac{\Gamma \vdash A, \Delta \quad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}}{\Gamma \vdash \Delta}$$

en une preuve se terminant ainsi (les feuilles correspondant toujours à $M, N$ et $S$, de gauche à droite):

$$\dfrac{\dfrac{\Gamma, A \vdash B, \Delta \quad \Gamma \vdash A, \Delta}{\Gamma \vdash B, \Delta} \qquad \Gamma, B \vdash \Delta}{\Gamma \vdash \Delta}$$

On a transformé une coupure sur une formule complexe $(A \to B)$ en deux coupures sur ses sous-formules immédiates. Cela remonte au grand logicien Gentzen. On fait deux autres remarques:

- Si je fais un zoom sur

$$\dfrac{\Gamma, A \vdash B, \Delta \quad \Gamma \vdash A, \Delta}{\Gamma \vdash B, \Delta}$$

  et si je relis cette inférence en termes de syntaxe, ce n'est pas la substitution effective de $N$ dans $M$ à chaque occurrence de $x$ qui est décrite ici, mais une opération nouvelle, dite de substitution explicite. Et en effet, dans une implantation, on ne va pas effectuer ces substitutions partout, mais plutôt accéder à des tables qui donnent les valeurs des variables libres.

- L'ordre entre ces deux coupures est un détail sans importance, on peut aussi bien faire une "multi-coupure":

$$\dfrac{\overbrace{\Gamma, A \vdash B, \Delta}^{M} \quad \overbrace{\Gamma \vdash A, \Delta}^{[x \leftarrow N]} \quad \overbrace{\Gamma, B \vdash \Delta}^{S}}{\Gamma \vdash \Delta}$$

Ces deux remarques suggèrent le passage à une machine abstraite à trois composantes, avec des états de la forme

$\langle M[\rho] | S \rangle$

$M$ est le code,
$\rho$ l'environnement,
$S$ la pile.

Un couple $M[\rho]$ est appelé une *fermeture] dans la communauté des langages fonctionnels.*
*Et on arrive à la machine de Krivine:*

$$\begin{array}{lcl}
\langle\, (MN)[\rho] \mid S \,\rangle & \to & \langle\, M[\rho] \mid N[\rho] :: S \,\rangle \\
\langle\, (\lambda x.M)[\rho] \mid N[\rho'] :: S \,\rangle & \to & \langle\, M[\rho[x \leftarrow N[\rho']]] \mid S \,\rangle \\
\langle\, x[\rho] \mid S \,\rangle & \to & \langle\, \rho(x) \mid S \,\rangle
\end{array}$$