Structures mathématiques du langage

Alain Lecomte

9 avril 2013

1 Calcul symétrique et continuations

1.1 Continuations

La théorie des continuations met au même plan les *fonctions* (ou termes, ou programmes etc.) et les *contextes* dans lesquels elles sont évaluées. La sémantique de Montague nous a déjà préparé à cette notion. Il y a deux manières de voir la génération d'une forme sémantique à partir de **Pierre dort**, la première considère que **Pierre** est représenté par une constante p' de type e (une "valeur") et **dort** par un terme **dort'** de type $e \rightarrow t$. On applique le terme à la constante et on obtient : **dort'**(p'). La seconde considère que **Pierre** sera évalué plus tard dans le contexte fourni par une continuation de phrase qui est [...] **dort**. A première vue, cela revient au même (le contexte s'appliquant au terme **Pierre** pour finalement donner aussi **dort'**(p')), sauf que, dans cette seconde approche, on a considéré **Pierre** non pas comme une *valeur*, mais comme un *programme*, lequel s'applique à une *continuation*. Selon un affreux néologisme, on a "continuisé" l'objet associé à **Pierre**. Ainsi, tout terme, avant de devenir une valeur (la valeur d'un nom ou d'une phrase), devra avoir rencontré une continuation. Que les fonctions (les constantes, les termes, les programmes...) *s'appliquent* à leur contexte rendra plus aisé la gestion de ces derniers. Bien sûr, pour que que **Pierre** s'applique à son contexte **dort'**, en donnant le même résultat que si c'était le contexte qui s'applique à lui, à savoir **dort'**(p'), il faut transformer la constante de type e en une fonction sur les continuations de type $e \rightarrow t$, autrement dit en un objet de type $(e \rightarrow t) \rightarrow t$, ce qu'on a avec $\lambda P.P(p')$.
Cette démarche peut être appliquée à tous les objets, et pas seulement aux noms propres comme cela est le cas dans la grammaire de Montague. On peut ainsi maintenant passer à la continuisation du verbe. **dort** agit alors sur son environnement, cela signifie que c'est un opérateur sur les continuations de type $((e \rightarrow t) \rightarrow t)$, autrement dit qu'il est lui-même de type $(((e \rightarrow t) \rightarrow t) \rightarrow t)$.
De la même manière qu'il est facile de déduire le terme (le programme) associé à **Pierre** au moyen du codage en λ-termes de la preuve du séquent $e \vdash (e \rightarrow t) \rightarrow t$ (montée de type), il sera aisé de déterminer le programme associé à **dort**, de la même façon, et nous obtiendrons : $\lambda \mathcal{U}.\mathcal{U}(\lambda x.dort(x))$, où $\mathcal{U}$ est une variable de type $(e \rightarrow t) \rightarrow t$.
D'une manière générale, soit M et N deux λ-termes, le premier de type $\alpha \rightarrow \beta$ et le second de type α. Ils sont respectivement transformés en leurs versions continuisées M^c et N^c de types respectifs $((\alpha \rightarrow \beta) \rightarrow t) \rightarrow t$ et $(\alpha \rightarrow t) \rightarrow t$. Mais comment, alors, exprimer la combinaison de M^c et de N^c, puisqu'ils ne sont plus désormais tels que l'un ait un type fonctionnel dont l'autre aurait le type de l'argument ? En utilisant la notion de contexte, on voit que ce qui était $(M\ N)$ (M appliqué à N) peut se lire de deux manières, soit comme le fait d'insérer N dans le contexte fourni par $M[\ ]$, puis d'insérer le tout dans un contexte vide, soit comme le fait d'insérer M dans le contexte fourni par $[\ ]N$, puis d'insérer le tout dans un contexte vide. Ces deux manières correspondent en fait à deux preuves possibles du séquent :

$$((\alpha \rightarrow \beta) \rightarrow t) \rightarrow t, (\alpha \rightarrow t) \rightarrow t \vdash (\beta \rightarrow t) \rightarrow t$$

L'une conduit à :

$$(M\ N)^c_v = \lambda u.(M^c\ \lambda m.(N^c\ \lambda x.(u\ (m\ x))))$$

où on évalue l'argument en premier.
L'autre à :

$$(M\ N)^c_n = \lambda u.(N^c\ \lambda x.(M^c\ \lambda m.(u\ (m\ x))))$$

où on évalue en premier la fonction.
Il est facile de vérifier que l'évaluation de la phrase **Pierre dort** conduit au même résultat selon les deux méthodes. En revanche, on vérifiera que celle de **tout le monde voit quelqu'un** ne conduit pas au même résultat, puisque les deux méthodes conduisent à des évaluations des quantificateurs dans des ordres différents.

Une remarque fondamentale en logique et en informatique théorique a consisté à remarquer que nous pourrions substituer $\perp$ au type distingué t (ou bien n'importe quelle constante R interprétée comme le type "réponse" pour un programme. Cela revient à considérer que, fondamentalement, un programme qui calcule un résultat produit d'abord, s'il termine, un résultat qu'on peut interpréter simplement comme le fait qu'il termine, le résultat calculé étant alors vu comme un effet de bord de cette action). De ce fait, les deux traductions précédentes proviendraient de deux preuves d'un séquent pouvant désormais s'écrire comme :

$$((\alpha \to \beta) \to \perp) \to \perp, (\alpha \to \perp) \to \perp \vdash (\beta \to \perp) \to \perp$$

c'est-à-dire :

$$\neg\neg(\alpha \to \beta), \neg\neg\alpha \vdash \neg\neg\beta$$

L'existence d'une telle preuve est triviale en logique classique, où nous disposons de la règle d'élimination de la double négation et où, donc, on a toujours $A \equiv \neg\neg A$, mais dans la logique intuitionniste que nous utilisons (parce qu'elle est constructive), la démarche que nous avons suivie revient à considérer l'évaluation *directe* comme correspondant au séquent de la logique classique $\alpha \to \beta, \alpha \vdash \beta$ et, en passant à la continuisation des termes, à le *remplacer* par le séquent $\neg\neg(\alpha \to \beta), \neg\neg\alpha \vdash \neg\neg\beta$ dont on cherche une preuve en logique intuitionniste. Or cela rencontre des résultats connus en logique depuis longtemps : le fait, entre autres, que l'on puisse encoder la logique classique dans la logique intuitionniste, selon des travaux anciens de Gödel. On peut démontrer en effet que si on prend la traduction suivante :

$$\begin{aligned} p^* &= p \\ (A \to B)^* &= A^* \to \neg\neg B^* \end{aligned}$$

on a :

Théorème 1 *Si π est une preuve de $\Gamma \vdash \alpha$ en logique implicationnelle classique, alors il existe une preuve π^* de $\Gamma^* \vdash \neg\neg\alpha^*$ en logique implicationnelle intuitionniste.*

On notera que, selon cette traduction, si p est un type atomique, $(p \to B)$ se traduit en $p \to \neg\neg B$.
En recherchant une preuve correspondant à $(M\ N)$, nous trouvons le λ-terme qui correspond à notre première traduction : l'argument doit être évalué avant d'être transmis à la fonction, raison pour laquelle on appelle cette méthode : *appel par valeur*.
Il existe aussi une autre traduction de la logique classique en logique intuitionniste, elle consiste à traduire les atomes en eux-mêmes et les implications en la négation de leurs réciproques. Autrement dit :

$$\begin{aligned} p^{**} &= p \\ (A \to B)^{**} &= \neg(B^{**} \to A^{**}) \end{aligned}$$

On a alors le théorème :

Théorème 2 *Si π est une preuve de $\Gamma \vdash \alpha$ en logique implicationnelle classique, alors il existe une preuve π^{**} de $\neg\Gamma^{**} \vdash \neg\alpha^{**}$ en logique implicationnelle intuitionniste.*

Considérons par exemple, là encore, le séquent $\alpha, \alpha \to \beta \vdash \beta$, il vient par cette traduction :

$$1.\neg\alpha, \neg(\alpha \to \beta)^{**} \vdash \neg\beta$$
$$2.\neg\alpha, \neg\neg(\beta^{**} \to \alpha^{**}) \vdash \neg\beta$$
$$3.\neg\alpha, \beta \to \alpha \vdash \neg\beta$$

qui n'est rien d'autre que le *modus tollens* !
Nous reviendrons plus loin sur cette traduction.

Désormais, nous pouvons distinguer :

- des *valeurs* : les objets sans négation, elles seront dans la suite notées V_α, où α est leur type,
- des *continuations* : les objets de type $\neg V$, où V est une valeur, elles seront notées dans la suite K_α (pour la continuation d'un objet de type α, autrement dit une continuation K_α est un objet de type $\neg\alpha$),
- des termes de type élevé, que nous avons appelés *programmes* (non évalués) : les objets de type $\neg\neg V$, qui seront notés dans la suite C_α (pour un programme de type α, il s'agit donc d'un objet de type $\neg\neg\alpha$).

On aura noté que l'on obtient une valeur en appliquant un programme à une continuation. Noter aussi qu'un terme fonctionnel, de type $\alpha \to \beta$, peut être vu de deux manières selon les traductions ci-dessus :

1. dans la traduction dite *appel par valeur*, comme une application d'une valeur de type α vers un programme de type $\neg\neg\beta$, donc une application $V_\alpha \to C_\beta$,
2. dans la deuxième, comme une application d'une continuation de type $\neg\beta$ vers une continuation de type $\neg\alpha$, autrement dit, par contraposition, comme une application d'un programme de type $\neg\neg\alpha$ vers un programme de type $\neg\neg\beta$, c'est-à-dire une application $C_\alpha \to C_\beta$. Puisque la valeur de l'antécédent n'est pas évaluée, on appellera cette méthode *appel par nom*.

Il devient maintenant possible d'appliquer ces notions au cas du calcul symétrique de Lambek-Grishin (selon une démarche que l'on doit à M. Moortgat et R. Bernardi).

1.2 Calcul symétrique

Le calcul symétrique de Lambek-Grishin est un exemple de calcul "classique". Les séquents y sont en effet symétriques. On cherche à obtenir l'équivalent de l'isomorphisme de Curry-Howard pour ce calcul. Or, on sait que la logique classique n'est pas constructive au sens où l'est la logique intuitionniste. Comment faire ? En réalité, de nombreuses études ont montré que l'on pouvait rendre constructive la logique classique à condition que l'on puisse exercer un contrôle sur les réductions. Nous allons ici considérer seulement une version non orientée (et non sensible aux ressources) de ce calcul (autrement dit pas de $/$ ni de $\backslash$ mais seulement une flèche $\to$, pas de $\diagdown\!\!\diagdown$ ni de $\diagup\!\!\diagup$, mais seulement une soustraction $-$), de plus nous nous contenterons de la version purement implicative (pas de $\otimes$, ni de $\oplus$). Les règles logiques que nous considérons sont donc les suivantes :

$$\frac{\Gamma, A \vdash B, \Delta}{\Gamma \vdash A \to B, \Delta}\,[\to D] \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \to B \vdash \Delta}\,[\to G]$$

$$\frac{\Gamma, A \vdash B, \Delta}{\Gamma, A - B \vdash \Delta}\,[-G] \qquad \frac{\Gamma \vdash A, \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma \vdash A - B, \Delta}\,[-D]$$

La règle de coupure sera, comme d'habitude :

$$\frac{\Gamma \vdash A, \Delta \qquad \Gamma', A \vdash \Delta'}{\Gamma, \Gamma' \vdash \Delta, \Delta'}$$

Etiqueter ces séquents avec des termes de preuve, comme on le fait en logique intuitionniste, pose problème. En logique intuitionniste, il y a un seul type résultant, à droite du $\vdash$, et c'est là qu'on met en principe l'encodage de la preuve, mais ici, il y a un nombre arbitraire de types sur la droite ! Et d'ailleurs comment interpréter ce fait ? Nous savons que les formules en partie droite doivent s'interpréter comme liées par $\vee$. On peut donc dire qu'un séquent $A_1, ..., A_n \vdash B_1, ..., B_p$, sur le plan calculatoire, s'interprète comme un programme qui calcule l'une des valeurs B_i à partir de tous les inputs A_j. On va, à partir de maintenant, sélectionner le B_i qu'on veut calculer : on dira qu'on met le *focus* sur lui, ou qu'on le rend *actif*.
On notera alors que, puisque nous sommes dans un calcul classique, où les formules peuvent être déplacées librement d'un côté à l'autre du séquent, à condition d'être niées, le séquent $A_1, ..., A_n \vdash B_1, ..., B_i, ..., B_p$ où on a mis le focus sur B_i revient à :

$$A_1, ..., A_n, \neg B_1, ..., \neg B_{i-1}, \neg B_{i+1}, ..., \neg B_p \vdash B_i$$

Les formules A_i donnent alors les types de valeurs $x_1, ..., x_n$ et les $\neg B_j$ sont des continuations, c'est-à-dire les types de continuations $\alpha_1, ..., \alpha_{i-1}, \alpha_{i+1}, ..., \alpha_p$. Dans ce cas, le séquent exprime en tant que but la construction d'un terme M de type B_i.
Mais comme le calcul est symétrique, nous ferons aussi la même chose du côté gauche. Si une formule active sur la droite correspond à un terme à construire, une formule active sur la gauche correspond alors à un contexte à construire. Cela sous-entend donc l'existence de règles pour *focaliser*. Ce sont les règles suivantes :

$$\frac{c : (\Gamma, x : A \vdash \Delta)}{\tilde{\mu}x.c : (\Gamma; A \vdash \Delta)}\,[\tilde{\mu}] \qquad \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\mu\alpha.c : (\Gamma \vdash A; \Delta)}\,[\mu]$$

La convention est la suivante : les termes de preuve étiquettent non plus une formule distinguée, mais un séquent dans son ensemble, la formule focalisée est celle qui n'a pas d'étiquette, en même temps, elle est séparée du reste des autres formules par un " ;"[1]. $[\tilde{\mu}]$ permet de stocker l'étiquette (x) de la formule focalisée lors du traitement de c, tandis que $[\mu]$ stocke un environnement. $[\tilde{\mu}]$ crée un nouveau contexte, et $[\mu]$ un nouveau terme. L'axiome d'identité possède alors deux variantes :

$$\alpha : (\Gamma; A \vdash \alpha : A, \Delta) \qquad\qquad x : (\Gamma, x : A \vdash A; \Delta)$$

Le premier axiome construit un contexte au moyen d'une variable de contexte α alors que le second construit un terme au moyen d'une variable x.
La règle de coupure devient :

$$\frac{v : (\Gamma \vdash A; \Delta) \qquad e : (\Gamma'; A \vdash \Delta')}{(v \bullet e) : (\Gamma, \Gamma' \vdash \Delta, \Delta')}$$

où $v \bullet e$ peut s'interpréter comme l'évaluation de v dans le contexte e, autrement dit : $e[v]$. Les séquents ainsi obtenus par coupure sont non-focalisés, on les dit centrés. Ils sont interprétés comme des *commandes*.
L'interprétation calculatoire peut alors être la suivante :

[1] Cette partie, détachée du reste, est dénommée "le bénitier" par Girard !

1. pour un séquent droite (où la formule focalisée est à droite), au moyen de la conjonction des inputs x_i (valeurs de types respectifs A_i), on construit un terme M de type B *ou* on passe une valeur à un contexte de type B_j pour un certain j,
2. pour un (co-)séquent ou séquent gauche, avec les valeurs données en inputs et en consommant le contexte construit L de type A, on passe une valeur à un des contextes B_j,
3. pour un séquent centré ou commande, on éxécute un calcul qui consiste à passer une valeur à l'un des contextes B_j en utilisant toutes les valeurs x_i données en input.

Le théorème d'élimination des coupures s'applique et il produit les règles de réduction suivantes :

$$(\mu) \quad (\mu\alpha.c \bullet e) \to c[\alpha ::= e]$$

$$(\tilde{\mu}) \quad (v \bullet \tilde{\mu}x.c) \to c[x ::= v]$$

La dernière expression correspond à l'instruction let en informatique. $c[x ::= v] \equiv \mathsf{let}\ x = v\ \mathsf{in}\ c$. Par analogie, on définit aussi : $c[\alpha ::= e] \equiv \mathsf{let}\ \alpha = e\ \mathsf{in}\ c$.
Quand on traduira ce calcul en une version intuitionniste selon la méthode *appel par valeur*, la coupure se traduira évidemment par une substitution, celle de e au contexte attendu par v. Cela revient à appliquer le λ-terme traduisant v : $\lambda k.k(v)$ au contexte e. Dans cette traduction, les variables ordinaires x se traduisent donc en $\lambda k.k(x)$ où k est une variable de contexte, et ces dernières se traduisent en simplement elles-mêmes (alors qu'en *appel par nom*, ce sera l'inverse).
Comme $\mu\alpha.c \bullet e$ se réduit à e substitué à α dans c, il ne sera pas étonnant que $\mu\alpha.c$ se traduise en $\lambda\alpha.c^t$, où ^{t} dénote la traduction souhaitée. De la même manière, $\tilde{\mu}x.c$ se traduira en $\lambda x.c^t$.
L'existence d'une paire critique :

$$(\mu\alpha.(y_1 \bullet \beta_1) \bullet \tilde{\mu}x.(y_2 \bullet \beta_2))$$

qui se réduit ou bien en $(y_1 \bullet \beta_1)$ ou bien en $(y_2 \bullet \beta_2)$ conduit justement à distinguer deux stratégies de réduction, l'une où $(\tilde{\mu})$ est systématiquement préférée, l'autre où c'est (μ). La première est notre *appel par nom*, la seconde à notre *appel par valeur*. On vérifie bien que dans l'*appel par nom*, on transmet un terme à un contexte sans qu'il soit nécessairement évalué, alors que dans l'*appel par valeur*, seule une valeur peut être transmise à un contexte.
Si nous ajoutons les règles logiques vues plus haut, nous obtenons :

$$\frac{v : (\Gamma, x : A \vdash B; \Delta)}{\lambda x.v : (\Gamma \vdash A \to B; \Delta)}\,[\to D] \qquad \frac{v : (\Gamma \vdash A; \Delta) \qquad e : (\Gamma; B \vdash \Delta)}{v@e : (\Gamma; A \to B \vdash \Delta)}\,[\to G]$$

$$\frac{e : (\Gamma; A \vdash \alpha : B, \Delta)}{\tilde{\lambda}\alpha.e : (\Gamma; A - B \vdash \Delta)}\,[-G] \qquad \frac{v : (\Gamma \vdash A; \Delta) \qquad e : (\Gamma; B \vdash \Delta)}{v - e : (\Gamma \vdash A - B; \Delta)}\,[-D]$$

Par exemple, $v@e$ est un contexte obtenu à partir d'un terme v et d'un contexte e. Il consiste à évaluer v et à mettre le résultat dans e. Le contexte obtenu attend alors un terme de type $A \to B$. On est passé d'un contexte qui attendait un terme de type B à un contexte qui attend un terme de type $A \to B$.
On doit noter également les règles de changement de *focus* ou *switch* :

$$\frac{M : (\Gamma; A \vdash \alpha : B, \Delta)}{\mu\alpha.(x \bullet M) : (\Gamma, x : A \vdash B; \Delta)}\,gd \qquad \frac{N : (\Gamma, x : A \vdash B; \Delta)}{\tilde{\mu}x.(N \bullet \alpha) : (\Gamma; A \vdash \alpha : B, \Delta)}\,dg$$

Ce sont bien sûr des règles dérivées (**exercice** : les démontrer !).

1.3 Exemple

Soit à analyser **Pierre dort**, avec **Pierre** : sn et **dort** : $sn \to s$. La dérivation est :

$$\dfrac{\dfrac{x : (x : sn \to sn) \qquad \alpha : (s \vdash \alpha : s)}{x@\alpha : (x : sn; sn \to s \vdash \alpha : s)}\,[\to G]}{\mu\alpha.(y \bullet (x@\alpha)) : (x : sn, y : sn \to s \vdash s)}\,switch$$

où la règle $switch$ doit être utilisée pour déplacer le focus du verbe vers le type résultant, de manière à pouvoir instancier y.
Par instanciation des variables x et y à, respectivement, p' et **dort'**, on obtient, pour la phrase la représentation :

$$\mu\alpha.(\mathbf{dort}' \bullet (p'@\alpha))$$

qu'il reste bien sûr à évaluer pour obtenir, par exemple **dort'**(p'). On fait alors appel à la traduction des termes en *call-by-value* (ou en *call-by-name*). Cela revient (pour *cbv*) à adopter un des plongements possibles de la logique classique dans la logique intuitionniste[2]. On a, par exemple :

- $\Gamma \vdash V : A; \Delta$ donne : $\Gamma^v, \neg\Delta^v \vdash V^{vl} : A^v$
- $\Gamma \vdash v : A; \Delta$ donne : $\Gamma^v, \neg\Delta^v \vdash v^v : \neg\neg A^v$
- $\Gamma; e : A \vdash \Delta$ donne : $\Gamma^v, \neg\Delta^v \vdash e^v : \neg A^v$
- $c : (\Gamma \vdash \Delta)$ donne : $\Gamma^v, \neg\Delta^v \vdash c^v : \bot$

d'où l'on déduit qu'un terme de type $A \to B$ s'encode comme un terme de type $A \to \neg\neg B$, donc un terme de type $sn \to s$ comme **dort** se convertit en un terme de type $sn \to ((s \to \bot) \to \bot)$ ou, par *décurryfication*, de type $(sn \times (s \to \bot)) \to \bot$, donc un terme qui, à partir d'une valeur x et d'une continuation k, construit une valeur de vérité (ou "une sortie de programme"). Il vient donc la traduction suivante pour **dort** : $\lambda(x, k).k(dort(x))$.
Noter ensuite qu'on a les traductions suivantes :

$$(p'@\alpha)^v = \lambda x.((x\ p'^v)\ \alpha^v)$$

d'où on obtient par remplacement et β-réduction :

$$\mu\alpha.(\mathbf{dort}' \bullet (p'@\alpha)) \quad = \quad \mu\alpha.((dort\ p'^v)\ \alpha^v)$$

Il se trouve qu'en *cbv*, $\mu\alpha.c$ se convertit simplement en $\lambda\alpha.c^v$. Il suffit alors de remplacer **dort'** par le λ-terme ci-dessus (et p'^v par p' en tant que valeur directement donnée) pour obtenir :

$$\lambda\alpha.((\lambda(x, k).k(dort(x))\ p')\ \alpha) \to \lambda\alpha.\alpha(dort(p'))$$

et de considérer que la continuation à laquelle ce terme s'applique est vide ([ ] traduit par l'identité Id) (il ne reste plus rien de la phrase après l'application du verbe) pour obtenir :

$$(\lambda\alpha.\alpha(dort(p'))\ \ Id) \to Id(dort(p')) \to dort(p')$$

Noter qu'une expression ne peut être évaluée que lorsque son environnement (une continuation) lui a été fournie. Dire qu'on évalue en *cbv* revient donc à dire qu'une expression ne peut être passée comme valeur à un programme (un terme) que lorsque les continuations nécessaires ont été fournies. En *cbn*, au contraire, on passe l'expression à un programme avant que les continuations n'aient été fournies,

[2]Le fait qu'il existe deux tels plongements entraîne le fait qu'il y ait deux stratégies d'évaluation.

autrement dit, on perpétue l'attente. On passe non de valeur en terme de type élevé, mais de terme en type élevé en terme de type élevé.
Par ailleurs, il est assez clair que notre présentation repose sur une parfaite symétrie entre termes et contextes (entre valeurs et continuations). Il est donc légitime de faire l'hypothèse que l'on passe de *cbv* à *cbn* par pure symétrie. Si on note ^{o} cette opération de symétrie, on aura :

$$\begin{array}{rcl} X^o & = & X \\ (A \to B)^o & = & B^o - A^o \\ (B - A)^o & = & A^o \to B^o \end{array}$$

avec les échanges suivants

$$\begin{array}{rcl} x & \leftrightarrow & \alpha \\ \mu\alpha & \leftrightarrow & \tilde{\mu}x \\ \lambda x & \leftrightarrow & \tilde{\lambda}\alpha \end{array}$$

Il existe donc une deuxième évaluation possible pour l'expression **Pierre dort**.
$\mu\alpha.(\mathbf{dort}' \bullet (p'@\alpha))$ devient par symétrie : $\tilde{\mu}x.((x - p') \bullet \mathbf{dort}')$.
mais en *cbn*, les entrées lexicales n'ont pas la même traduction, ainsi p' est remplacé par $\lambda k.k(p')$ et le verbe devient une fonction qui, à un couple formé d'une continuation de type sn et d'un programme de type s, associe une sortie $\perp$. Autrement dit, **dort'** se traduit par :

$$\lambda(k,q).q(\lambda x.k(\mathbf{dort}'(x)))$$

Noter que, par ces symétries, on obtient :

$$\phi^n = (\phi^o)^v$$

d'où :

$$\begin{array}{rcl} (A \to B)^n & = & (B^o - A^o)^v \\ & = & (B^o)^v \wedge \neg(A^o)^v \\ & = & B^n \wedge \neg A^n \end{array}$$

Autrement dit, en *cbn*, la traduction d'une implication est la négation de sa réciproque, sauf en ce qui concerne une formule atomique, $X^n = X$, ce qui est exactement la deuxième des traductions que nous avons vues à la section 1.
On obtient aussi que la traduction de $A \to B$ est la conjonction de la traduction de B et de la traduction de $\neg A$, soit :

$$(A \to B) \to \perp = (B \to \perp) \wedge ((A \to \perp) \to \perp)$$

ce qui établit une égalité entre continuations de type $A \to B$ et couples formés d'une continuation de type B et d'un programme de type A. Soit :

$$K_{A \to B} = C_A \times K_B$$

d'où

$$C_{A \to B} = C_A \to C_B$$

ainsi que :

$$K_{A \to B} = K_B \to K_A$$

Autrement dit nous retrouvons le fait qu'une continuation de type $A \to B$ associe une continuation de type A à une continuation de type B.

1.4 Continuations vues comme monades

Pour tout type fixé ω (par exemple t ou bien $\perp$), on peut considérer une *monade continuation* avec le type de réponse ω. Le foncteur T est défini par :

$$T(\alpha) = (\alpha \to \omega) \to \omega$$

L'injection η est définie par :

$$\forall a : \alpha, \eta(a) = \lambda c.c(a) : T(\alpha)$$

et la composition est :

$$\forall m : T(\alpha), k : \alpha \to T(\beta), m \star k = \lambda c.m(\lambda a.k(a)(c)) : T(\beta)$$