Continuations, logique linéaire, et. . .
Mécanique Quantique

Jerzy Karczmarczuk

Université de Caen.

Résumé : Nous présentons la structure (l'**essence calculatoire**) du modèle fonctionnel des calculs quantiques ; il s'avère que la « logique des ressources » de Girard a son mot à dire dans ce contexte. . .

Concentré de Logique linéaire

Les deux règles (*Contraction*, *Affaiblissement*) disant que la *vérité* peut être exploitée plusieurs fois, et qu'une hypothèse redondante ne gêne personne :

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} \qquad \text{et} \qquad \frac{\Gamma \vdash B}{\Gamma, A \vdash B}$$

sont abolies. Le jugement : $A \to B, A \vdash A \times B$ n'est pas prouvable.

L'implication $A \to B$ est remplacée par la « sucette » $A \multimap B$, « l'implication linéaire » où la premisse est utilisée une seule fois. On dit que LL est une *logique des ressources*.

Il existe une correspondence (l'« isomorphisme de Howard-Curry ») entre les propositions logiques et les programmes fonctionnels. L'implication $A \to B$ est un type fonctionnel du type A vers B.

[[Voici un exemple trivial dans la logique classique, l'équivalence entre $(A \times B) \to C$ et $A \to (B \to C)$.

Ceci se traduit en équivalence entre les fonctions « curryfiées » et non : `f x y = z` et `f x = \y -> z`.

Les variantes linéaires seront utiles plus tard.

Concrètement, nous allons commenter l'équivalence entre :

$A \multimap B$ et $(A \otimes B^{\perp})^{\perp}$,

où $\perp$ et $A^{\perp} = A \multimap \perp$ sont : l'objet « dualisant », et la dualisation ; cette dernière est liée à la constructions des *adjoints*.]]

La $\multimap$ décrit des fonctions *structurellement linéaires*, où l'argument est utilisé une seule fois. On ne peut le copier ni s'en débarasser.

Il existent deux 'conjonctions' $A \otimes B$ (A et B ensemble, produit tensoriel), et $A \,\&\, B$ ('avec': choix entre A et B). La disjonction $A \oplus B$ est : A ou B, mais il existe encore autre disjonction, « parallèle » $A \parr B$.

Le modèle le plus classique (et assez trivial) de cette logique est notre vielle algèbre vectorielle, où la conjonction tensorielle c'est le … produit tensoriel des espaces, et la conjonction $\&$ – leur somme directe. Cependant pour nous ceci n'est pas si trivial que ça, car la structure de la mécanique quantique est en effet basée sur la géométrie linéaire dans l'espace de Hilbert approprié.

Soyons francs : pendant 80 ans on a élaboré la théorie quantique sans jamais avoir besoin de la logique linéaire.

Mais, plus récemment on parle des *calculs quantiques*, on essaye d'établir un cadre pour les modèles calculatoires de la MQ. On note que la **programmation fonctionnelle – variantes du calcul lambda – est vraiment mieux adaptée à cette sorte de problèmes, que la programmation impérative**, car en MQ il n'y a pas d'« effets de bord », tout se passe par une transformation linéaire de l'état, jusqu'aux mesures (finalisations), où la théorie s'arrête.

Un processus quantique – action des opérateurs sur les vecteurs d'états – est comme l'application d'une fonction, dont le but est de passer le résultat à une autre fonction, sa *continuation linéaire*, etc.

Décrivons donc un modèle informatique des entités quantiques, qui réalise de manière minimaliste ce que les physiciens font sur papier. Le modèle a été réalisé en Haskell.

Plusieurs modèles de « mécanique quantique dans l'ordinateur » sont élaborés actuellement [Selinger, VanTonder, Sabry, etc.]. Il y a des langages impératifs, avec des « registres quantiques », on essaye de simuler le non-déterminisme de la MQ par un calcul lambda non-déterministe (où on retourne une liste de résultats possibles), etc. Les états sont des structures de données, opérateurs sont des matrices concrètes.

Ces modèles, conçus par des informaticiens qui sont loin des souffrances épistémologiques quantiques, sont – à notre avis – largement **trop forts**, contiennent souvent des entités artificielles, sacrifient la compréhension en faveur de facilité du calcul. Les « vrais » états quantiques sont abstraits, et, par exemple, sont *indépendants de la base choisie*. Nous avons donc essayé d'être beaucoup plus modeste, et de découvrir au lieu d'inventer. . .

Introduction à la quantisation

Récapitulons l'approche « informatique » à la définition des systèmes physiques, classiques et quantiques. L'état classique d'un système est l'ensemble de ses observables : position, vitesse, angles. Pour une particule, c'est (x_c,p_c), etc. Si **on sait** que le système est quantique, on ajoute d'autres descriptions, tel l'énergie (niveau) d'excitation. Les paramètres *complémentaires*, comme la position et l'impulsion doivent être *alternatifs*. Quelques exemples :

```
data Mpoint =Xc Real | Pc Real   Xc n'est qu'une balise
data Rotator=Ang Real Real | Jm Int Int
data Oscil  =X Real | P Real | N Int
data Qubit  =B0 | B1     --  seul essentiel pour les "infos"
```

Un état quantique est un **vecteur indexé par les états classiques**. On le note : $|N\rangle$, $|B0\rangle$ (**plutôt :** $|0\rangle$, $|1\rangle$), $|\alpha,\beta\rangle$, etc.

On peut construire des superpositions linéaires des vecteurs : $|\psi\rangle = u|0\rangle + u|1\rangle$. Le produit scalaire $\langle\alpha|\beta\rangle$ des vecteurs $|\alpha\rangle$ et $|\beta\rangle$ est l'amplitude de probabilité de trouver la configuration α dans l'état préparé à partir de β. La probabilité $p(\alpha,\beta) = |\langle\alpha|\beta\rangle|^2 = \langle\alpha|\beta\rangle \cdot \langle\beta|\alpha\rangle$.

Mesurer un état est lui appliquer un *projecteur*. (En fait on ***mesure le projecteur dans cet état***). Par exemple, en appliquant l'opérateur – produit dyadique $|\alpha\rangle\langle\alpha|$ à un $|\psi\rangle$, on obtient forcément $|\alpha\rangle$, et sa normalisation $\langle\alpha|\psi\rangle$ est l'amplitude de probabilité de mesurage réussit.

Toute *observable* (spin, énergie, etc.,) est un opérateur hérmitien qui peut être representé comme une somme de projecteurs sur ses vecteurs propres, pondérés par les valeurs propres : $\hat{O} = \sum_k \lambda_k |\alpha_k\rangle\langle\alpha_k|$. Les λ_k sont les seuls valeurs qui peuvent se manifester quand on effectue la mesure.

J’ai proposé une représentation *universelle* des états quantiques comme objets fonctionnels sur les configurations classiques, sachant qu’un espace fonctionnel est un espace linéaire naturel. Si on définit une fonction *quelconque* respectant quelques règles de symétrie et positivité, par exemple un « *bracket* – supersélecteur classique »

```
axis al be = if al==be then 1 else 0 -- Kronecker
```

(d’autres possibilités sont aussi importantes), alors la fonction d’un argument `axis al = \be->axis al be` (où `\` dénote le λ de Church) est un vecteur naturel :

```
axis x + axis y = \z->axis x z + axis y z
c*axis x = \z->c*(axis x z)
```

Ainsi nous pouvons modéliser l'espace de Hilbert dans l'ordinateur, **`axis alpha`** impléménte $\langle\alpha|$. Il nous faut encore la base duale, pour pouvoir définir les produits scalaires (la métrique et les probabilités).

On cherche des moyens calculatoires permettant de **simuler les systèmes quantiques**, utiles pour la compréhension des nouveaux algorithmes, et aussi pour faciliter les calculs en physique, jeter un pont entre le raisonnement formel *abstrait* et les calculs numériques.

Un « ket », **vecteur d'état** primitif $|\alpha\rangle$, dual à $\langle\alpha|$ est donné comme une fonction *sur les axes*.

```
ket alpha = \ax->ax alpha
```

La théorie mathématique nous donne un cadeau gratuit : les fonctions – *kets* ne sont pas seulement des vecteurs (étant des fonctions), mais des ***fonctionnelles linéaires***. La preuve est triviale.

```
ket a (ax1 + ax2) = (ax1 + ax2) a = ax1 a + ax2 a
                  = ket a ax1 + ket a ax2
```

Malgré cette trivialité, le résultat est conceptuellement important. Personne ne sait pourquoi la M. Q. est une théorie linéaire. Le passage aux espaces fonctionnels *ne répond pas* à cette question, mais offre une pseudo-réponse : ***c'est naturel...*** L'affinité entre la linéarité algébrique et logique (argument d'une fonction/preuve n'est utilisé qu'une seule fois) est connue.

Dualité. Notons $|\alpha\rangle$ un vecteur élémentaire, et $|\psi\rangle$ – quelconque. Le formalisme est suffisamment complet, pour permettre la construction de l'axe $\langle\psi|$ dual à ce dernier. On sait que $\langle\psi|\alpha\rangle = \langle\alpha|\psi\rangle^*$. Ceci est implémenté comme

```
(dual psi) alpha = conjugate (psi (axis alpha))
```

Les scalaires constituent les « objets dualisants » de Girard, et pour toute entité A, l'objet dual $A^{\perp} = A \multimap \perp$ est la fonction qui « finalise » A. Le rapport entre la dualité algébrique (conjugaison de Hermite, les adjoints) et la dualité logique n'est pas trivial. Les kets sont duals aux axes (ils finalisent les axes), mais pas *vice-versa*, il faut construire des « bras » **isomorphes** aux axes, qui finalisent les kets.

```
(coax psi) chi = psi (dual chi)
```

Les états des systèmes composites sont des *produits tensoriels* $|\phi\rangle \otimes |\psi\rangle$ des états composants. On construit des fonctions bi- (et n-) linéaires :

```
psip = phi <*> psi = \ax ay ->phi ax * psi ay
```

Les états évoluent (sont modifiés) par des actions des *opérateurs*, qui peuvent aussi être multi-linéaires.

Ils s'enchaînent, $|\psi'\rangle = \hat{O}_1\hat{O}_2|\psi\rangle$, etc. Nous pouvons « quantiser » les « opérateurs classiques » – fonctions agissant sur le substrat, par exemple **`const B0`** (où **`const x y = x`**), or **`not x = if x==B0 then B1 else B0`** par la construction **`opqu = lift opcl`**:

```
(lift opcl) psi = \ax->psi (\alpha->ax(opcl alpha))
```

Les opérateurs sont évidemment des bestioles du genre $A \multimap B$ (avec B équivalent à A dans des cas typiques).

On peut écrire la définition d'un opérateur concret sous forme **`opqu psi ax = z...`**. Mais cela, selon l'isomorphisme HC représente la formule $(A \otimes B^{\perp}) \multimap \perp$, en accord avec LL.

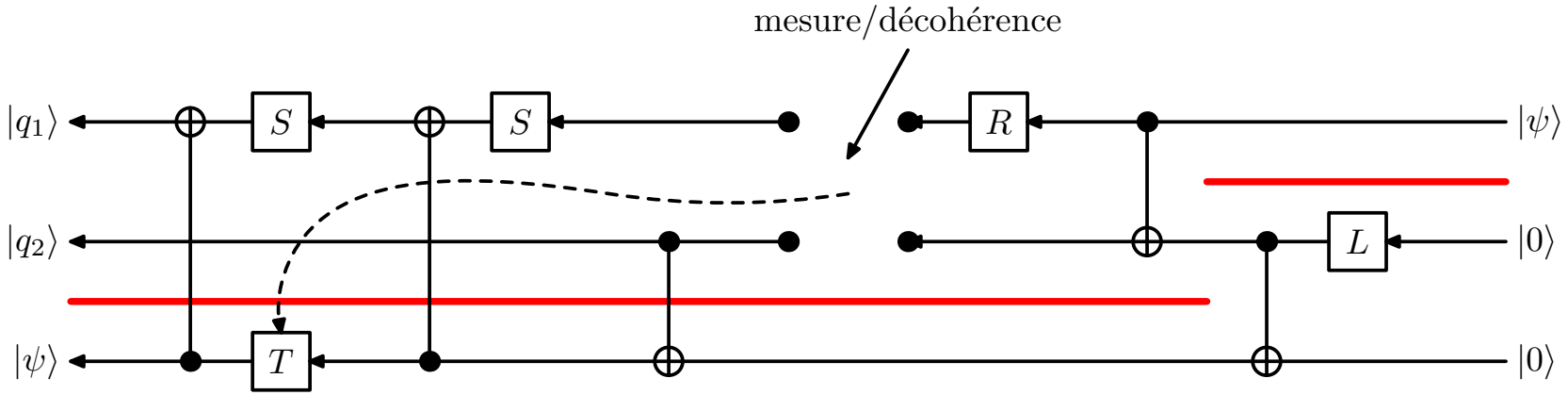
On peut, bien sûr, construire des opérateurs duals, c'est un fragment de la construction ci-dessus. Mais il ne faut oublier que passage d'un espace $\mathbb{T}$ vers l'espace des fonctions sur $\mathbb{T}$ est un **foncteur contravariant**, ce qui signifie que l'on construit l'opérateur adjoint. La construction est simple, comme presque toutes ici ; l'universalité catégorique a l'apparence superficielle. . .

Si **`aop`** est un opérateur agissant sur les axes, alors on construit l'opérateur adjoint agissant sur les kets : **`kop = boost aop`**, où

```
(boost aop) psi = \ax -> psi (aop ax)
```

(Un exemple d'adjonction algébrique. Si $\langle n|$ décrit le niveau d'excitation d'un oscillateur, et $\langle n|\hat{O}_a$ (agissant à gauche) donne $\langle n-1|$, alors l'opérateur « boosté » fait $\hat{O}_k|n\rangle = |n+1\rangle$. Mais ce n'est pas la même chose que l'inverse ; plutôt transposé.)

On construit des états composites $|\alpha\rangle|\psi\rangle$ comme des produits tensoriels de composantes individuelles, et on y applique des opérateurs composites, en construisant des« boîtiers » ou « circuits » à partir des opérations primitives comme le *controlled NOT* et autres. Voici le circuit de téléportation d'un qubit. Les détails ne sont pas importants, on voit la linéarité (visuelle) des transitions.



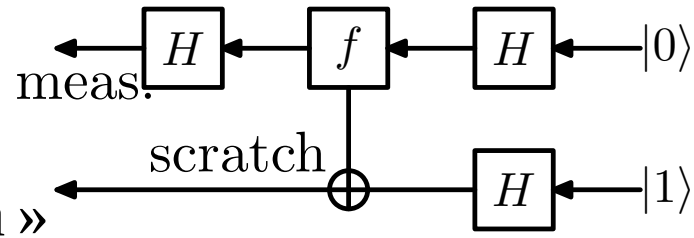
La ligne rouge sépare les sites des acteurs A et B. Le combinateur de deux lignes est le « *controlled not* » quantique. L'opérateur $S = i|0\rangle\langle 0| - |1\rangle\langle 1|$, $L = R^{+} = (|1\rangle\langle 0| - |0\rangle\langle 1| + \mathbf{1})/\sqrt{2}$, T est la rotation du spin effectuée par B selon le résultat de la mesure, transmis classiquement par A, etc. Tout cela correspond à quelques fonctions Haskell pas trop compliquées. Le *controlled not* est :

```
cnot kt ax ay = p B0 + p B1 where
  p b = kt (qproj ax b) (xor (axis b) ay)
```
où

```
qproj ax b = ax b *> axis b
warp a b = \ax -> ax a*>axis b
xor r = r B0 *> id +
        r B1 *> (warp B0 B1 + warp B1 B0)
```
*mult.: scalaire*vecteur*

$|a\rangle\langle b|$

Nous avons fait quelques autres calculs concrets dans le cadre de notre formalisme, par exemple nous avons implémenté l'« oracle de Deutsch » (simulé, bien sûr…).



(C'est une fonction capable de répondre en **un pas** si une fonction sur les bits est constante ($f(0) = f(1)$) ou pas. L'opérateur $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ est la matrice de Hadamard, la boîte f c'est le lifting de la fonction cherchée aux qubits. Les détails n'ont pas d'importance.) Ce qui nous intéresse ici c'est la structure du modèle. Son avantage par rapport à l'usage des listes, tableaux, etc. pour représenter les vecteurs d'état et les opérateurs, est sa modéstie, naturalité et l'impossibilité de « tricher »: clôner les états inconnus, ou observer illicitement la structure d'un état.

Continuation Passing Style

Un rappel sommaire du CPS, **le** paradigme fondamental de sérialisation et de dé-hiérarchisation des applications dans la programmation fonctionnelle. Au lieu de prendre une donnée `x` et lui appliquer une fonction, `(f x)`, et ainsi de suite : `g (f x))`..., on demande à ce que **toute** fonction f prenne un paramètre supplémentaire - le « futur », une fonction appelée *continuation*, qui fait quelque chose avec le résultat de f.

Ainsi, au lieu de l'addition primitive `(+)` on utilise l'« addition continuée » : `addc x y cnt = cnt (x+y)`. Une simple donnée `x` est « liftée » vers un objet fonctionnel dérivé de `x` :

```
xc = \cnt->cnt x
```

Un calcul composite : $\sqrt{a^2 + b^2}$ sera transformé en :

```
ac              \a->
bc              \b->
mulc a a        \a2->
mulc b b        \b2->
addc a2 b2 \z->
sqrtc z          ... le futur de la valeur finale
```

ce qui suggère un « code assembleur » pour une machine à registres. C'est une des raisons pourquoi CPS est important dans le domaine de compilation. Mais, nous pouvons constater qu'un protocole similaire s'est manifesté dans notre modèle des calculs quantiques.

Notre vecteur d'état dérivé d'une configuration classique α, le $|\alpha\rangle$ défini comme **`\ax->ax alpha`** possède justement cette forme, la fonctionnelle de Fischer-Plotkin, qui « lifte » les valeurs élémentaires sur le domaine CPS, typiquement représentée par la translation $x \Rightarrow \overline{x} = \lambda k \to k\, x$.

Les translations CPS « classiques »: $M \Rightarrow \overline{x}_M$ (Fischer-Plotkin et successeurs) sont présentés d'habitude sous forme

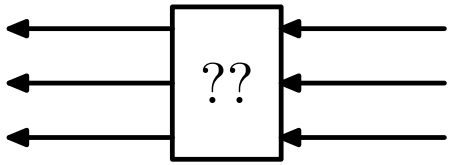
$$\begin{aligned} (\lambda x \to M) &\Rightarrow \lambda k \to k(\lambda x h \to \overline{x}_M\ h) \\ M\ N &\Rightarrow \lambda k \to \overline{x}_M(\lambda c \to \overline{x}_N(\lambda x \to c\ x\ k)) \end{aligned} \qquad (1)$$

Le lifting des opérateurs agissant sur les états produit un type similaire (non pas identique, car les prémisses et les détails du lifting sont différents). Rappelons-nous :

```
opquant psi = \ax->psi (\alpha->ax(opclas alpha))
```

Comme auparavant, les axes jouent le rôle de continuations pour les kets (et vice-versa, mais ceci est tordu. . .). On obtient un enchaînement linéaire, qui peut être codé sous forme des appels récursifs terminaux. Les continuations pertinentes sont *linéaires* au sens : utilisées exactement une fois. Ceci se généralise aux produits tensoriels, « états multi-particules ».

On peut se poser la question : comment « produire deux lignes » pour résultat? CPS en donne la réponse :



C'est équivalent à la *consommation du consommateur* de ces deux lignes, et c'est justement ce que nous avons. Un état quantique n'est pas un objet/donnée. C'est un *calcul* (*computation*) qui demande sa finalisation ultérieure, dont le sens est de calculer, les amplitudes de probabilité – et de les confronter avec l'expérience, ce qui est sort de notre cadre théorique.

Observations finales

Les physiciens théoriciens gaspillent de plus en plus le papier pour manipuler les formules et constructions mathématiques malgré une bonne maîtrise de paquetages de calcul formel, car ceux-ci gèrent des symboles et des structures syntaxiques, plutôt que des *entités abstraites*, définies mathématiquement. Ils veulent des *vraies* formes différentielles, indépendentes du système des coordonnées, non pas des tableaux ou records. Ils veulent que l'ordinateur comprenne l'associativité des opérateurs et en fait l'usage. Etc.

Avec toutes les méta-hiérarchies, subsomptions, etc. en mathématiques, les systèmes orientés-objet sont actuellement un peu trop rigides. La programmation fonctionnelle qui permet de « réifier » les morphismes, foncteurs, relations, etc. de manière paramétrique, semble donner plus de puissance formelle au programmeur. C'est ma vision des choses…