Peut-on simuler un système quantique?

Jerzy Karczmarczuk
Dept. d'Informatique, Université de Caen, France
(mailto:karczma@info.unicaen.fr)

Résumé

Nous présentons une méthodologie de construction informatique assez générale des entités qui correspondent aux états et aux opérateurs dans la mécanique quantique. La construction est « abstraite » au sens discuté ultérieurement, mais réalisable dans un langage populaire de programmation fonctionnelle, Haskell. Nous suggérons que la méthodologie adoptée correspond mieux que plusieurs autres à la notion de *simulation*, où les entités réalisées dans l'ordinateur réflètent notre état de connaissances des systèmes simulés, en gardant une certaine indépendance par rapport au contexte d'observation. L'article peut être lu par les personnes possédant un *minimum* de connaissances sur la mécanique quantique

1 Introduction : est-ce une bonne question?

En effet, vu le nombre de papiers scientifiques et de réalisations logicielles consacrées à la simulation des « circuits quantiques » – voir p. ex. le site de Julia Wallace [1], qui répertorie tous les travaux significatifs dans ce domaine, ou même des langages de programmation conçus spécialement pour modéliser des objets quantiques (registres de qubits, etc.) [2] – on peut penser que la réponse à la question du titre soit trivialement positive. Les premiers (« vrais », classiques…) ordinateurs des années '40, avaient pour but la solution des problèmes par excellence quantiques, comme la fission nucléaire. Un des objectifs primaires de l'élaboration des architectures parallèles était leur application dans la physique moderne. Le statut épistémologique de la théorie quantique dans sa couche « standard », acceptée par le milieu scientifique, s'est stabilisé dans les années '40 – '50. Les techniques calculatoires, numériques, évoluent toujours, mais depuis 20 ans aucune révolution, ni technique ni conceptuelle n'a vu le jour. L'expérience acquise est donc bonne, et relativement stable.

On sait que la complexité algorithmique des calculs quantiques n'est pas comparable avec la théorie classique, et qu'un ordinateur quantique peut résoudre en temps polynomial des problèmes NP-durs, voir p. ex. [3, 4]. Ce fait a provoqué une véritable explosion dans le milieu informatique, et actuellement le nombre de scientifiques intéressés par le « mythe » de l'ordinateur quantique subit une très forte croissance. Certes, il y a des dérives formalistes, certains informaticiens élaborent des modèles du calcul quantique sans tenir compte des intrications liées aux problèmes de la *mesure d'information effective* dans un système non-classique, mais le progrès est indiscutable.

Les physiciens eux aussi continuent à s'intéresser aux calculs quantiques. Il est d'ailleurs fort probable que les nouveaux algorithmes dans le domaine de la cryptographie quantique, solution des problèmes NP-durs, sécurisation du transfert d'information, etc., tout ce qui est actuellement le « *spiritus movens* » du domaine parmi les informaticiens « purs », constitue une branche secondaire du développement ; ce qui est ***vraiment*** l'essentiel dans le domaine d'application de ces nouvelles architectures dont tout le monde rêve, est la *possibilité de simuler des systèmes quantiques quelconques*, les structures de la matière réelle : noyaux atomiques, semi-conducteurs, entités biochimiques, etc., par un ordinateur, c'est-à-dire par un système quantique discret, contrôlable et programmable. Ceci était l'opinion de Richard Feynman [5], et ainsi s'exprime actuellement John Preskill [6], et autres, p. ex. [7]. Une véritable armée, engagée et multi-compétente, travaille sans arrêt depuis des années sur la modélisation des problèmes calculatoires quantiques sur l'ordinateur…

Alors quel est le problème, et est-ce qu'il y a un problème?

2 Simulation est un mot très fort !

2.1 Ontologie quantique reste un mystère…

…, alors l'épistémologie se trouve dans un état douteux également. Notre vision du monde est évidemment classique[1], et désormais le mot « simulation » signifie la simulation sur une architecture classique. Nous voulons dès le début distinguer entre la solution d'un problème quantique (solution du problème de Sturm-Liouville de l'équation de Schrödinger, diagonalisation d'une matrice, solution numérique d'un problème perturbationnel, etc.), et la simulation proprement dite, ce qui signifie l'implantation dans l'ordinateur de quelque chose qui « correspond à un objet réel ». C'est la signification donnée au mot « simulation » par un physicien typique. Alors, dans le domaine classique, la solution des équations de Newton ne constitue pas *per se* une simulation, mais la création d'un objet, ou « chose » dont les attributs sont la position et la vitesse résultantes desdites équations – si.

Mais nous ne savons vraiment pas ce qui est une « chose » dans le monde quantique. Comment dans ces conditions « mettre un électron » dans l'ordinateur? Un tableau qui représente sa fonction d'onde, l'état quantique $|\Psi\rangle$? Dans quelle représentation : « $\vec{x}$ » ou « $\vec{p}$ », sachant que les deux sont incompatibles?

Comme il a été dit, classiquement on simule une particule par une structure de données qui contient $(\vec{x},\vec{p})$, et on construit les équations du mouvement, le changement de ces variables dans le temps. La construction des programmes qui gèrent ces équations est un jeu d'enfant, et actuellement *peut être réalisée par les enfants*, voir [8]. Dans la mécanique quantique ces variables sont des ***opérateurs***, qui peuvent être représentés par des matrices (de taille finie ou infinie), ou les opérations de dérivation, etc., et qui *ne contiennent aucune dépendance temporelle*. Seulement l'état, disons, la fonction $\Psi(\vec{x},t)$ de Schrödinger dépend de temps.

En fait, ceci n'est pas vrai… Comme tout physicien le sait, cette dépendance temporelle caractérise seulement le *cadre*[2] *de Schrödinger*. Selon la représentation de Heisenberg, les opérateurs $(\vec{x},\vec{p})$ (les « observables » cinématiques) dépendent de temps, mais l'état Ψ reste immuable, défini une fois pour toutes, au moment de la préparation du système. En tout cas il n'est jamais directement observable…

Sur le plan théorique ou calculatoire ceci n'est pas grave, on sait comment passer d'une représentation à l'autre, et on sait également que les quantités mesurables ne dépendent pas de notre choix du « cadre ». Cependant, si nous accordons une signification pragmatique *réaliste* au mot **simulation**, si une structure implémentable dans un langage de programmation devrait correspondre à une entité dans le monde réel, la situation devient très ambiguë, personne ne connaît cette entité.

La solution de l'équation de Schrödinger n'est pas un objet, même si représente – indirectement – une quantité mesurable, la distribution de probabilité de trouver la particule dans un endroit concret.

On sait qu'une mesure, p. ex. de la position de la particule détruit son état quantique précédent, et crée un nouvel état, filtré par l'observateur (équipement, pas obligatoirement humain). Ceci rend très difficile la séparation entre le système simulé, et l'observateur. Le monde devient ***inséparable***, ce qui peut nous déranger philosophiquement ou non, mais remet en question la possibilité de construire un simulateur méthodologiquement propre. Si on ajoute ce fait au contexte déjà catastrophique – le non-déterminisme *absolu* des systèmes quantiques, le fait que la théorie – apparemment fondamentale – *refuse* de prédire le résultat d'une expérience sur un individu, et demande que l'on se place dans le cadre d'observations statistiques – les chances de « mettre une particule quantique » dans l'ordinateur sont petites.

2.2 Modestie philosophique est de rigueur

Il existe, depuis les années '60 *un* modèle ontologique de la physique quantique, celui de Everett, ou le « modèle des Univers multiples » dont l'ambition est d'« *expliquer* » le non-déterminisme quantique, en relativisant la réalité classique. Selon Everett (ou plutôt, selon une des réinterprétations de son hypothèse, plutôt obscure) le *vrai univers* est une variété de « mondes parallèles », qui réalisent **toutes** les possibilités d'évolution compatibles avec le formalisme quantique. Si l'électron peut – selon l'orthodoxie – passer par une de deux fentes, selon la hypothèse des multi-univers, dans la moitié de ces mondes il passe par l'une, dans la seconde moitié – par l'autre. La question « pourquoi il est allé à gauche » n'a pas de sens, car dans d'autres univers l'observateur

1. Nous refusons catégoriquement toutes les spéculations gratuites sur ce sujet, même si nous adorons la Science-Fiction…
2. ou « représentation », mais ce terme officiel français est ambigu. En Anglais on dit "picture".

pose la question inverse, et la question se réduit à une autre, explicitement ridicule : « pourquoi moi, je suis moi et non pas quelqu'un d'autre? ».

Ces mondes intéragissent par le phénomène d'interférence quantique. La « vraie » particule est un corpuscule classique, mais elle est représentée par une onde qui est présente, et traverse l'univers multiple, engendrant ainsi le dualisme corpusculaire-ondulatoire, ses manifestations expérimentales, et son statut formel.

Le modèle peut être formulé de manière cohérente, mais les physiciens pragmatiques, après une courte fascination déclenchée par les tentatives de quantiser la cosmologie, l'ont rejetés en bloc, vu son aspect *purement* spéculatif, sans aucun pouvoir de prédiction et sans l'attribut de réfutabilité Popperienne.

Paradoxalement, le modèle a été exhumé dans le contexte informatique, et D. Deutsch a essayé – avec un certain succès – populariser cette vision du monde en suggérant que l'immense puissance calculatoire des ordinateurs quantiques est le résultat du fait qu'ils « calculent simultanément dans plusieurs mondes », et l'interférence, plus les mesures quantiques finissent l'assemblage du résultat final.

Chez les spécialistes en physique quantique ceci a provoqué un certain dégoût, car il est très difficile de vivre à cheval entre la Science et la Science-Fiction de qualité moyenne (sans trop de valeurs littéraires...).

Est-il – néanmoins – possible d'en tirer quelques conclusions positives? Répétons que le sujet de ce travail est l'analyse de la possibilité de *simuler* les objets quantiques. Dans l'interprétation de Copenhague la seule entité à notre disposition est la fonction d'onde, qui représente l'amplitude des probabilités, de potentialités du système. Même si on ne sait pas ce qui veut dire « potentialité » (un mot décidément scholastique...), on voit que l'objet quantique est un objet **abstrait**.

Un objet dans « notre » monde selon Everett possède également des « potentialités », son prolongement dans d'autres instances de l'univers qui ne sont pas visibles. Elles contribuent à notre entité de manière abstraite.

Un état quantique $|\Psi\rangle$ peut être instancié comme une fonction d'onde de Schrödinger $\Psi(x) = \langle x|\Psi\rangle$ dans l'espace, ou également donnée par la transformée de Fourier de cette fonction, la représentation dans l'espace des impulsions $\widehat{\Psi(p)} = \langle p|\Psi\rangle$. Ce sont des « concrétisations potentielles » qui ne caractérisent vraiment pas le système, mais son observation seulement. L'état lui même en est indépendant. Il est donc abstrait, encore une fois.

Il ne faut pas sur-interpréter le mot « abstraction » vu de la perspective de ce travail. Un type de données abstrait dénote des objets dont la structure interne n'est pas observable, dont toute communication passe par les « propriétés contractuelles » (« méthodes », si on veut utiliser le langage de la programmation par objets) reconnues par le compilateur, et dûment codées en accord avec les spécifications du noyau exécuteur, *hardware* ou machine virtuelle.

Un objet, p. ex. un vecteur abstrait, est une entité géométrique **réelle**, qui existe indépendamment de l'observateur. Cependant, si nous voulons transmettre nos connaissances de cet objet, il faut le « concrétiser » au sens : présenter ses composantes, trouver son instance dans un système de coordonnées. Ceci est vrai également dans la physique classique, seulement le passage classique entre les représentations est neutre, tandis que dans la physique quantique il est actif, modifie les résultats des expériences ultérieures. De plus, selon les règles de la théorie quantique *un état arbitraire ne peut être copié, « clôné »*[9], ce qui constitue une contrainte très non-naturelle dans le domaine de programmation sur des données classiques.

Notre conclusion est donc un peu paradoxale, mais la seule qui nous paraît sérieuse. Si nous voulons réellement simuler un système quantique, le plus abstrait est le modèle, le mieux, car ainsi il devient le moins dépendant de l'observateur. Oui, la simulation des systèmes quantiques passe par l'abstraction, même si dans cette formulation l'arrière-goût de l'idéalisme Platonicien peut devenir insupportable pour quelques uns... Nous voulons donc souligner encore une fois que notre philosophie est à 100% réaliste, nous cherchons seulement des moyens d'implémentation adaptés aux exigences quantiques. Il va de soi que le role joué par le modèle de Everett dans notre travail est purement anecdotique. Les objets fonctionnels sont abstraits comme « types de données abstraites », (ADT), et se « concrétisent » par leur application aux données spécifiant les relations entre le système et son observateur.

3 Abstractions et programmation fonctionnelle

Le reste de notre travail tourne autour la thèse suivante : actuellement le meilleur modèle calculatoire, implémentable sur l'ordinateur et utilisable par tous, et capable de décrire des abstractions du genre trouvé dans l'analyse des systèmes quantiques, est la **programmation fonctionnelle**. Ces paradigmes semblent être relativement bien appropriés à la création des entités abstraites (réalisées comme des objets fonctionnels), et permettent de leur imposer plusieurs caractéristiques mathématiques de manière assez naturelle et générique. Un objet fonctionnel est opaque, sa seule « concrétisation » est son application à un ensemble de données. Ceci constitue la réalisation de la « potentialité » qui a été mentionnée dans la section (2.2).

Nous avons réalisé un petit paquetage en Haskell (voir le site [10]) – un langage fonctionnel pur, polymorphe, typé statiquement avec l'inférence automatique de types (extension du système de typage de Hindley-Milner), et paresseux. Le paquetage construit de manière très générique les vecteurs d'état, les observables etc., et au maximum exploite la généricité de l'approche, la possibilité d'utiliser les mêmes mécanismes combinatoires (bases duales, produits tensoriels, etc.) indépendamment du système quantisé. Les mathématiques classiques (l'analyse) nous enseignent que ***les fonctions constituent un espace vectoriel***. Nous allons construire concrètement ces fonctions, et nous allons équiper l'espace de ces fonctions avec un produit scalaire « naturel », ce qui génère un espace métrique avec des bonnes propriétés, donc l'espace de Hilbert.

Le cadre construit est suffisant pour résoudre des exercices typiques en mécanique quantique, et aussi permet la construction des « circuits quantiques », pour manipuler les qubits, et réaliser quelques algorithmes quantiques récents, qui ont fait une belle carrière.

3.1 Comment quantiser un système dynamique?

Prenons un système physique classique : un rotateur décrit par son axe de révolution et l'angle azimutal, ainsi que par sa vitesse de rotation, ou un oscillateur harmonique, décrit par la position et par l'impulsion de la particule qui oscille. Ou, un système à deux états, p. ex. un atome dans son état de base, ou excité.

La première leçon à apprendre est la suivante : ***une configuration classique est une « étiquette », une balise*** qui identifie un vecteur abstrait, et constitue la spécification d'une *base naturelle* dans un espace vectoriel.

La seconde leçon est que parfois les informations classiques répertoriées ci-dessus sont redondantes de manière inacceptable. Sans même engager la théorie des observations quantiques et le principe d'incertitude de Heisenberg, nous pouvons dire que les attributs considérés complémentaires, comme la position et l'impulsion, ou : l'angle et le moment cinétique associé, constituent des ***bases alternatives***.

Voici donc quelques spécifications de configuration de systèmes dynamiques en Haskell. Les barres verticales séparent les alternatives. Donc, un système à deux niveaux qui peut se trouver dans l'état **`Up`** *ou* **`Down`** est représenté par un type de données qui peut être nommé **`Qubit`**, et qui possède la représentation symbolique alternative **`Up | Down`**. Nous aurons :

```
data Qubit   = Up | Down

data Rotator = Ang Double Double | Jm Integer Integer
data Oscil   = X Double | P Double | N Integer
```

Pour une particule *toute position*, $x = 1.65; x = -12.985$ etc., constitue une configuration différente, nous en avons un nombre illimité, et leur énumération n'est pas possible, donc nous utilisons un type parametré, p. ex., **`X Double`**, ou **`Double`** est un type standard, les nombres flottants qui représentent $\mathbb{R}^1$, et **`X`** un simple identificateur, le nom de la balise, comme **`P`** ou **`Ang`** etc.

Dans les exemples ci-dessus nous avons investi quelques connaissances de ces systèmes. Un rotateur peut *aussi* être représenté par une paire (j,m), où j représente le moment cinétique total, et m sa projection sur un axe. À l'oscillateur nous avons ajouté une balise parametrée par un entier : **`N n`** (avec n non-négatif, même si ceci n'est pas explicit), qui constitue la base de Fock, qui spécifie le niveau d'excitation, ou le nombre de quanta élémentaires décrivant un état du système. Pour un informaticien intéressé par les calculs quantiques, l'« orientation » **`Up | Down`** n'est pas significative, il serait préférable d'utiliser une autre notation, comme **`B0 | B1`** dont la connotation avec les Booléens ou avec les bits est plus évidente. Dans tous les cas, ces entités restent abstraites, sans une sémantique *a priori*, sauf la possibilité de les différencier (et reconnaître l'égalité entre deux).

Une définition de ce genre impose bien sûr la connaissance de la structure du système par son auteur. Nous allons créer des instances de ces types, p. ex., **`q = N 6`** dans le programme, mais nous soulignons que *ces types n'ont aucune propriété mathématique*, nous ne pouvons les ajouter ou multiplier par quoi que ce soit. La variable **`q`** n'est qu'une étiquette formelle.

3.2 Comment mettre l'espace de Hilbert dans un PC?

Indépendamment du statut des objets introduits ci-dessus, il est possible, naturel, et *universel* au sens catégorique, de créer des véritables vecteurs basés sur ces objets. Rappelons que toute fonction d'un espace $\mathbb{X}$, dans $\mathbb{R}$ ou dans un autre espace « numérique » peut être considérée comme un vecteur[3], selon les règles :

$$(f+g)x = f(x)+g(x) \quad (1)$$
$$(a\cdot f)x = a\cdot f(x) \quad (2)$$

En Haskell une telle construction est parfaitement possible, et lisible, il faut simplement *surcharger* les opérateurs arithmétiques aux objets fonctionnels. Ceci peut être fait en déclarant les fonctions comme des ***instances*** de la ***classe***[4]. **`Num`** – la catégorie d'objets qui admettent l'addition, ou, pour des raisons de discipline, définir une nouvelle classe, celle de vecteurs abstraits, avec l'addition définie par l'opérateur **`(<+>)`**, et basé sur un corps de scalaires (d'habitude complexes ; nous gardons le paramétrage, car notre paquetage est un peu plus générique que la description simplifiée dans cet article…), comme ci-dessous.

```
class Vspace v where
 (<+>) :: v -> v -> v          --  Aussi : <->
 (*>)  :: Scalar -> v -> v
```

Ensuite on déclare l'instance, en précisant que les fonctions de type **`x->Scalar`** appartiennent à la classe **`Vspace`**:

```
type Hvector b = b->Scalar

instance Vspace (Hvector b) where
 (f <+> g) a = f a + g a       --  Aussi : <->
 (c *> f)  a = c*(f a)
```

Désormais nous pouvons écrire des expressions comme **`2.0*>(sin <+> exp)`**, dont l'évaluation engendre des objets fonctionnels applicables aux nombres. Ces objets (fermetures) sont opaques, et non-décomposables. Nous avons notre abstractio in machina !

Comment construire de manière naturelle des fonctions-vecteurs associés aux bases quantiques? Encore une fois, il faut exploiter nos connaissances, et parmi d'autres nous savons que pour des bases discrètes dans l'espace des états quantiques, les vecteurs de base sont d'habitude orthogonaux et normalisables. Nous ***imposons*** donc une certaine structure métrique qui caractérise la base définie par les balises **`Qubit`**, **`Oscil`**, etc. Voici une proposition minimaliste. *Tout* système quantique, le **`Qubit`**, le**`Oscil`**, la spécification de l'atome d'hydrogène, etc., noté comme une *variable type* **`a`**, s'il est précisé de manière à pouvoir distinguer deux configurations classiques différentes (il admet la notion d'équivalence, le *contexte* **`Eq`**), permet la construction

```
class (Eq a) => Hbase a where
  bracket :: a -> a -> Scalar
  bracket j k = kdelta j k  --  Kronecker

instance Hbase Qubit
instance Hbase Oscil  --  etc.
```

3. À condition que a appartient à un corps ; sinon on parle de *Modules* plutôt que d'espaces vectoriels

4. Rappelons, pour éviter la confusion avec les paradigmes orientés-objet, qu'en Haskell la classe est une relation satisfaite par les *types*, la constatation que les types qui constituent ses instances disposent d'une famille de fonctions polymorphes sur les données appartenant à ces types

qui signifie : tout système quantique légitime est spécifié par une *base*, ensemble d'objets abstraits auquel *nous* imposons les propriétés métriques, p. ex. **`bracket (N k) (N l)`** égal à 1 si $k = l$, zéro sinon. Bien sûr, la vérité est un peu plus compliquée techniquement.

- Pour un oscillateur il faut que les deux k et l soient non-négatives ; pour un rotateur (j,m) la projection m doit respecter $|m| \leq j$, etc., sinon le **`bracket`** disparaît trivialement.
- Pour les bases alternatives l'orthogonalité n'a pas lieu, par exemple **`bracket (X x) (P p)`** $= \exp(-ipx/\hbar)$ où $\hbar$ est la constante de Planck.
- Plusieurs systèmes admettent des bases non-orthogonales, qui souvent facilitent les calculs (p. ex. la base d'états cohérents pour l'oscillateur).

Le point crucial de la construction est l'abstraction fonctionnelle, la définition qui se réduit à l'introduction d'un nom un peu plus significatif :

```
axis x = \y -> bracket x y
```

où « `\` » est le « lambda » de Church, le constructeur des objets fonctionnels. Sachant que Haskell respecte l'ordre *normal* d'évaluation des formules : **`f a b c`** $\equiv$ **`((f a) b) c`**, et que l'identité syntaxique entre les définitions **`f x = P`**, et **`f = \x -> P`** a lieu, nous pouvons écrire directement **`axis = bracket`**.

Les « axes » constituent nos bases quantiques, des véritables vecteurs, assimilés aux « bras » de Dirac : **`axis (N k)`** est le vecteur de base $\langle k|$ de Dirac pour un oscillateur. Les vecteurs $\langle 0|$ et $\langle 1|$ dans le monde des qubits, circuits quantiques, etc., seront représentés par **`axis B0`** et **`axis B1`**.

Cet article par force majeure doit rester limité et qualitatif, nous ne pouvons élaborer des exemples très concrets, mais remarquons que

1. la définition des objets-balises abstraits au sens de ne pas contenir aucune sémantique naturelle, plus
2. la construction des *abstractions fonctionnelles* sur les objets ci-dessus, plus
3. l'imposition de la métrique selon les principes de la mécanique quantique standard

nous ont permis la construction d'un domaine mathématique parfaitement implémentable, et adapté à la construction de combinaisons linéaires, en accord avec le principe de superposition quantique. Le formalisme, et le paquetage Haskell est universel et pratique, il a été utilisé pour résoudre des problèmes assez concrets.

Notons que l'espace de configurations de zéro dimensions, fini (disons, 2 valeurs décrivant un bit) engendre l'espace vectoriel 2-dimensionnel. Si l'espace classique possède déjà des dimensions non-triviales, p. ex. « x » est un vecteur 1-dimensionnel, l'espace de Hibert bâti sur les configurations classiques est infiniment-dimensionnel, toute valeur de x est un vecteur de base indépendant.

3.3 États et bases duales

Ceci n'est pas la fin de la construction. Les axes sont des « bras », $\langle\ |$, il nous faut encore les « kets » de Dirac, les vecteurs $|\psi\rangle$ qui représentent les états eux-mêmes, et dont le produit scalaire avec un vecteur de base, disons $\langle 0|\psi\rangle$, est l'amplitude de probabilité de trouver la configuration « 0 » dans l'état décrit par ψ.

Nous avons donc besoin de la base duale, et nous profitons du fait que les constructions duales sont aussi universelles au sens catégorique. Les kets seront des fonctions linéaires sur les axes. L'application d'une telle fonction **est** la construction du produit scalaire approprié. Les kets *élémentaires* associés aux axes primitifs sont donnés par

```
ket alpha ax = conj ax alpha
```

où **`conj`** est la conjugaison complexe définie sur les fonctions, **`(conj f) x = conjugate (f x)`** (ou : **`conj f = conjugate . f`**, où **`(.)`** est l'opérateur de composition fonctionnelle). Étant des fonctions dont le co-domaine est scalaire, numérique (**`(ket alpha) ax = (conjugate (ax alpha)`**, les kets *automatiquement* deviennent des instances de notre espace vectoriel : **`3.5*>ket (N 1] <-> 1.2*>ket (N 0)`** est parfaitement légal, et la construction est conceptuellement terminé, nous omettons quelques détails techniques.

Le formalisme définit également assez naturellement une transformation de dualité, permettant de transformer un ket en un axe, et ainsi donnant à l'utilisateur la possibilité de définir le produit scalaire de deux kets, ce

qui correspond à l'amplitude de probabilité de recouvrement de deux états. Cette opération, nommée **`dual`** est donnée par une formule très simple

```
dual kt = kt . axis
```

Le produit scalaire de **`kt`** et **`ka`** aura la forme **`kt (dual ka)`**. Voici la vérification quand **`kt=ket alpha`** :

```
(ket alpha) (dual ka) = conj (ka . axis) alpha
   = conjugate ((ka . axis) alpha) = conjugate (ka (axis alpha))
```

ce qui correspond à la propriété exigée du produit scalaire : $\langle a|\alpha\rangle = \overline{\langle\alpha|a\rangle}$. La linéarité généralise la validité de la définition à tous les kets.

3.4 Opérateurs

Les opérateurs linéaires qui agissent sur les états en produisant d'autres, sont les ingrédients fondamentaux de la physique quantique, en fait la *seule chose* que l'on peut faire avec un état (sauf le calcul du produit scalaire avec un vecteur dual), est de lui appliquer un opérateur. Les opérateurs sont des « observables », déterminent l'évolution du système, réalisent les symétries, etc. Ils sont des objets fonctionnels du genre : **`Ket -> Ket`**.

Dans des cas discrets ils peuvent être représentés par des matrices (finies ou infinies), mais ceci implique le choix d'une base concrète, et son instanciation. dans notre formulation toute une famille d'opérateurs peut être construite par les combinaisons linéaires de formes primitives $|\alpha\rangle\langle\beta|$, il faut seulement préciser s'il s'agit d'un opérateur qui agit sur les kets : $|\alpha\rangle\langle\beta| \cdot |\psi\rangle = \langle\beta|\psi\rangle|\alpha\rangle$, ou de son dual (adjoint) applicable aux axes $\langle\phi|$ ce qui donne $\langle\phi|\alpha\rangle\langle\beta|$. Le premier et le second possèdent les définitions différentes (adjointes) :

```
ktdyade alpha beta = \kt -> (kt (axis beta))*>ket alpha
axdyade alpha beta = \ax -> (ax alpha)*>axis beta
```

L'existence des combinaisons linéaires est assuré par le fait que Haskell permet l'instanciation récursive des classes (non seulement pas la définition des types récursifs comme des listes), par exemple

```
type Hmat b = Hvector b -> Hvector b

instance Vspace (Hmat b)
 where
  (f <+> g) a = f a <+> g a
  (c *> f)  a = c*>(f a)
```

c'est-à-dire : une fonctionnelle transformant les vecteurs constitue également l'élément d'un espace vectoriel.

Les 20 dernières années de la théorie des langages de programmation se trouvent sous le signe de la programmation par objets, programmation « donné-centrique ». Les fonctions, transformations, ou méthodes constituent une couche un peu secondaire (même si indispensable), et le concept de « réification », la conversion des entités programmables en objets est très important.

La programmation fonctionnelle renverse un peu ce courant, et psychologiquement ces manipulations « fonctio-centriques » sont souvent difficiles à accepter (p. ex. par les physiciens. C'est un fait établi…). Ceci est un phénomène intéressant, en particulier car il devient très vite évident que les fonctions sont également des données, les fermetures lexicales sont des vrais objets fonctionnels, et plusieurs paradigmes de la programmation par objets (p. ex. l'héritage) y trouvent leur place.

La compréhension de quelques détails mathématiques profonds est très importante. Nous avons vu que les transformations de dualité sont simples. Nous pouvons aisément construire des opérateurs très pratiques comme la négation dans le domaine des qubits par

```
qnot = dyade B0 B1 <+> dyade B1 B0      --  Où :

dyade = axdyade
```

ou l'opérateur d'« annihilation » qui décremente le niveau d'excitation d'un oscillateur, $(\sqrt{n}) \cdot |n\rangle\langle n-1|$, voir p. ex. le très bon livre [11] :

```
oa ax (N n) = sqrt n * ax (N (n-1))
```

agissant sur les axes, etc. Cependant, l'application (*mapping*) entre un domaine X et le domaine des fonctions sur X est un ***foncteur contravariant***. Ceci signifie que si F est une transformation qui effectue la transition $F : X \to Y$, et f : une fonction agissant sur les $x \in X$, nous aurons « gratuitement » un opérateur induit F^* qui agit sur les fonctions, mais il est du genre $F^* : Y \to X$, nous construisons en fait l'opérateur adjoint ! Sa construction est le *pullback* catégorique classique : $(F^* f)x = f(Fx)$.

Nous pouvons en déduire que dans le domaine des kets notre opérateur d'annihilation devient en fait l'opérateur de création (il augmente le niveau d'excitation), et que heureusement la négation est un opérateur *self-adjoint*, utilisable sans changement dans les deux bases (mais il ne faut pas oublier que $(AB)^* = B^* A^*$). Ceci est très facile à prouver, mais essentiel est que de telles propriétés sont universelles, indépendantes de tous les détails d'implantation.

3.5 Mesures

Dans le cadre de notre formalisme nous avons résolu plusieurs petits problèmes pédagogiques trouvés dans les livres sur la mécanique quantique. Cependant le résultat était d'habitude « théorique » : les niveaux approximatifs d'énergie d'un système perturbé, ou les valeurs d'un produit scalaire entre les états qui correspondent à un modèle.

Or, dans le cadre de la simulation l'observateur ne voit aucun produit scalaire. Si la théorie nous donne la valeur de, disons $\langle 0|\Psi\rangle$, où $\langle 0|$ est la représentation de **`axis B0`**, un *simulateur* doit construire $p_0 = |\langle 0|\Psi\rangle|^2$, et physiquement engendrer l'entité **`B0`** stockée dans une structure de données classique, avec la probabilité p_0 ou l'objet formel **`B1`** avec la probabilité $p_1 = 1 - p_0$.

Donc, notre paquetage est équipé d'un générateur de nombres aléatoires général, capable de produire des résultats aléatoires en accord avec une distribution de probabilité donnée.

Pour un système plus complexe, p. ex., l'oscillateur, dont la base est infiniment dimensionnelle, la situation peut paraître difficile. Mais Haskell est un langage paresseux qui permet aisément la définition des structures de données *potentiellement* infinies, par exemple une liste de tous les nombres entiers non-négatifs. L'objet

```
nats = liste_de 0   where
  liste_de n = n : liste_de (n+1)
```

où l'opérateur **`(:)`** est le « **`cons`** » dans Lisp est une liste **`[0, 1, 2, 3, ...]`** etc., et un tel objet peut être manipulé sans craintes si on *regarde* effectivement seulement un segment fini de cette structure. Si un système quantique se trouve dans un état normal, avec l'énergie totale finie, la probabilité de trouver des excitations lointaines décroît vers zéro assez rapidement, et tout échantillon fini construit une distribution « infinie » réalisable, par exemple la distribution de Poisson (distribution des excitations d'un oscilateur qui simule un laser) : $p_n = \mu^n / n! \cdot \exp(-\mu)$ pour $n = 0{,}1, \ldots \infty$. Notre paquetage fait cette simulation sans aucun problème, sauf si on choisi l'excitation moyenne μ si grande, que la mémoire déborde. . . .

4 Qubits et circuits quantiques

4.1 Systèmes composites et produits tensoriels

Le statut de la composition de deux systèmes indépendants en physique quantique est si spécifique que provoque jusqu'aujourd'hui des discussions et tentatives de ré-interprétation.

La première leçon est : en mécanique classique un système composite est l'assemblage formel, par le produit cartésien, des sous-systèmes. Donc, en principe nous pouvons prendre une configuration résultante de cette composition cartésienne, et en faire une base quantique. Nous obtiendrons ainsi la *somme directe* des espaces vectoriels composants.

La seconde leçon est moins gentille : ceci ne sert à rien, et structurellement ne correspond à aucun mécanisme permettant de dériver les propriétés d'un système quantique. De plus, une telle construction permettrait la séparation locale de la base en composantes, ce qui serait illégal.

Si nous voulons simuler un système quantique complexe, il faut exploiter des faits établis (voir [11, 12] ou tout livre complet sur la mécanique quantique), et construire des ***produits tensoriels*** des bases, des kets, etc. Le produit tensoriel $|\psi\rangle \otimes |\phi\rangle$, noté aussi comme $|\psi\rangle|\phi\rangle$ ou $|\psi; \phi\rangle$, n'est pas un vecteur dans la somme

directe des espaces de Hilbert appropriés, ce qui corresponderait à la situation classique, rejetée ci-dessus. Si un vecteur dans l'espace de Hilbert est une fonctionnelle linéaire, le produit tensoriel de deux tels vecteurs sera une forme *bi-linéaire*, une fonction qui prend deux arguments. Alors l'ensemble d'un millier de sous-systèmes serait décrit par des fonctions qui prennent un millier d'arguments, ce qui semble un peu ridicule à un informaticien « orthodoxe ». Mais conceptuellement, si nous acceptons des listes avec des milliers d'éléments, alors pourquoi pas?...

Si nous voulons construire une quantité observable à partir de $|\psi;\phi\rangle$ il faut construire un *bracket* de Dirac : $\big(\langle\alpha|\langle\beta|\big)|\psi;\phi\rangle \equiv \langle\alpha|\psi\rangle\cdot\langle\beta|\phi\rangle$.

La construction en Haskell est délicate, mais assez évidente. Nous définissons la *classe de tenseurs* contenant l'opérateur **(<*>)**, la multiplication tensorielle

```
class Tensor v1 v2 v3  | v1 v2 -> v3
 where
  (<*>) :: v1 -> v2 -> v3
```

où la clause **| v1 v2 -> v3** précise au système que le type (l'arité) du résultat **v3** est *uniquement* défini par les types des arguments **v1, v2**. Nous pouvons ensuite construire quelques instances, en commençant par le fait que pour les scalaires **(<*>)** se réduit à la multiplication normale. Plus génériquement, si *un* argument est scalaire, il suffit d'appliquer la multiplication extérieure, déjà connue :

```
instance Tensor Scalar v v
 where
  s <*> v = s *> v
```

et la clause la plus importante, récursive, est :

```
instance (Tensor v1 v2 v3) => Tensor (a->v1) v2 (a->v3)
 where   u <*> v = \x -> u x <*> v
```

Même si ceci est difficile à digérer par les scientifiques conditionnés par Fortran, l'appareillage technique est facile à comprendre, et correspond à la construction des tenseurs dans les livres de géométrie. Nous avons construit des tenseurs duals, et nous avons montré comment construire et multiplir les opérateurs agissant sur les états composites. Ceci est indispensable pour toute simulation des systèmes « calculatoires », les circuits quantiques qui opèrent sur des séquences de qubits. Nous aurons donc les opérateurs qui agissent sur des multi-kets, étant des fonctions de plusieurs axes.

4.2 Circuits primitifs

L'évolution d'un système quantique est unitaire, réversible. Ceci signifie qu'aucune opération du genre NAND, XOR, etc., qui combinerait deux qubits en un : $|x\rangle|y\rangle \to |z\rangle$ n'est légale. De même, une fonction classique d'un bit : $x \to f(x)$ qui n'est pas explicitement réversible ne peut être « quantisée » directement, convertie en opérateur. Parmi les quatres fonctions d'un bit seulement l'identité et la négation sont utilisables. Mais il a été démontré que toute transformation peut induire un opérateur reversible, si on lui ajoute des « lignes » : arguments d'entrée et de sortie dans la représentation graphique d'un tel opérateur, comme sur la Fig. 1.

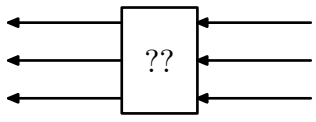

FIG. 1 – *Operateur multi-argument*

Ainsi, l'équivalent de l'opération de différence symétrique XOR est réalisé comme la *négation contrôlée*, CNOT, dont la sémantique est $|x\rangle|y\rangle \to |x\rangle|x\oplus y\rangle$, représentés sur la Fig. 2, et implémentée par le programme suivant

```
cnot k x y = p B0 + p B1 where
    p b = k (qproj x b) (xor (axis b) y)
```

où

```
qproj x b = x b *> axis b
xor r = r B0 *> id <+> r B1 *> qnot
```

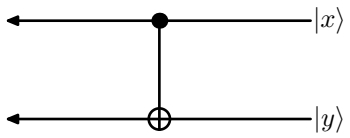

FIG. 2 – *« Controlled-not »*

Notons que nous avons engendré des « miettes », des lignes de sortie qui ne servent vraiment à rien (x), mais qui sont indispensables pour pouvoir inverser le flot de contrôle. La construction du simulateur devient chère, le modèle contient des lignes de transfert d'information qui peuvent facilement se multiplier hors de tout contrôle. Heureusement il existe une technique d'optimisation, découverte par Bennet [13] – on effectue une partie du processus, on inverse le flot de contrôle après avoir mémorisé le résultat principal, et en revenant à la configuration initiale on peut « libérer » et réutiliser les lignes qui constituent le lest. Cette opération est exécutée périodiquement. Nous travaillons actuellement sur l'implantation de cette technique, qui est facilitée par la paresse du langage (on n'instancie pas les objets dont on n'a pas besoin). Mais nous voyons que la simulation d'un ordinateur quantique[5] exploite des astuces qui n'ont aucune raison d'être dans l'informatique classique ; on joue même avec la « flèche du temps », et ceci toujours au nom de la pragmatique réaliste…

4.3 Un simple exemple, problème de Deutsch

Supposons qu'une fonction classique inconnue $f(x)$ définie sur un bit et produisant un bit, doit être analysée, l'utilisateur doit vérifier au moins si c'est une fonction constante : $f(0) = f(1)$ ou non. Classiquement il faut la mesurer deux fois. Deutsch a montré qui si on arrive la transformer en un opérateur quantique, nous pouvons l'appliquer à une superposition de deux qubits, et obtenir le résultat final après un pas (accompagné d'un filtrage). Le résultat doit être « classique » au sens : déterministe ; le système doit répondre **`B0` ou `B1`** avec la probabilité 1 (concrètement, le vecteur résultant est proportionnel soit à **`ket B0`**, soit à **`ket B1`**, et non pas à une superposition ambiguë). Le circuit sur la Fig. 3 réalise cet algorithme. On commence par la préparation de deux qubits d'entrée, figés à $|0\rangle$ et $|1\rangle$. Ensuite on leur applique la transformation de Hadamard, qui effectue les transformations $|0\rangle \to \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, et $|1\rangle \to \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. (La représentation matricielle souvent citée de cette transformation est $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$). Ce mélange est transmis au module décisif qui est la généralisation de notre négation contrôlée : $|x\rangle|y\rangle \to |x\rangle|f(x) \oplus y\rangle$, et ensuite le résultat est obtenu par la transformation de Hadamard (qui est une involution, c'est-à-dire, est son propre inverse) sur une ligne, tandis que l'autre est abandonnée.

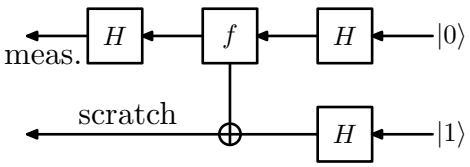

FIG. 3 – *Problème de Deutsch*

L'implantation du circuit est simple.

```
had = sqrt 0.5 *>(qnot <+> dyade B0 B0 <-> dyade B1 B1)
```

5. Il faut préciser que la technique a été proposée dans le cadre des calculs reversibles pas forcément quantiques

```
inp = (had <*> had) (ket B0 <*> ket B1)
-- functions :
fmut = id
fcst = const B0

-- Résultats :
xout = (had <*> id) (fcnot fmut inp)
yout = (had <*> id) (fcnot fcst inp)
```

où fcnot généralise cnot

```
fcnot f k x y = p B0 + p B1 where
    p b = k (qproj x b) (xor (axis (f b)) y)
```

et est représenté par le sous-circuit similaire à **cnot**, avec le disque noir remplacé par la boîte contenant f. Pour pouvoir tester la validité de ce programme il faut compléter la définition des opérateurs tensoriels, ce qui aurait pris trop d'espace dans cet article. Pour nos buts il suffit d'avoir une définition limitée du produit tensoriel de deux opérateurs individuels, qui agissent sur les axes, et qui produit un opérateur agissant sur bi-kets :

```
(aop1 <*> aop2) biket  = \ax1 ax2 -> biket (aop1 ax1) (aop2 ax2)
```

Le lecteur doit reconnaître que si toutes ces définitions auxiliaires (et universelles) sont placées dans une librairie, la programmation devient *extrêmement courte et lisible*, elle correspond à la spécification graphique du système. Les résultats, **xout** et **yout** sont appliquées à des axes **axis 0** et **axis 1**, et le résultat final est zéro pour la combinaison « mauvaise ».

Nous pensons donc que les techniques fonctionnelles seront les meilleurs outils de programmation des ordinateurs quantiques futurs, et actuellement constituent le meilleur outil de simulation de tels objets. Le style est clair, compact, et – ce qui peut être très important – nous ne confondons pas les entités classiques (*nombres* 0 et 1), et les bits abstraits, qui seront transmutés en qubits. L'abstraction aide à garder la vision réaliste du processus.

5 Conclusions

Nous ne pouvons nous défendre si le lecteur trouve que notre « simulation » ne l'est vraiment pas. Comme nous avons souligné, l'ontologie quantique est obscure (sauf si on accepte le modèle d'Everett, mais celui-ci n'est pas implémentable…).

Notre ambition est donc limitée. Nous avons repris les concepts établis dans la physique quantique standard et enseignée, des objets mathématiques reconnus, et dont la correspondance avec les sciences expérimentales est « respectable ». Nous ne savons toujours pas ce qui est une particule[6], mais nous savons comment préciser le *minimum* de propriétés satisfaites par son *état*. Puisque nous n'avons pas d'autres, l'état, et les opérateurs (observables et autres) sont nos seuls acteurs participant dans la simulation. Il s'agit d'une approche méthodologique minimaliste, qui profite au maximum des propriétés formelles de la théorie, sans – dans la mesure du possible – figer la représentation des objets quantiques par leur observations.

Nous avons implémenté les objets quantiques : les états et les opérateurs dans le cadre de la programmation fonctionnelle pure. Ceci nous a permis d'établir une plate-forme conceptuelle commune entre le monde du calcul assisté par l'ordinateur, et l'épistémologie typique d'un physicien théoricien réaliste. Les statuts d'un ket quantique, et d'un objet fonctionnel implémenté sur l'ordinateur, présentent beaucoup d'affinités, la réalisation fonctionnelle explicite le caractère de connaissances dont dispose l'observateur mieux, que toute réalisation « concrète », par les tableaux de bits, listes de nombrex complexes, etc., même si dans des cas concrets une réalisation de cet autre genre peut être *beaucoup* plus efficace numériquement.

Ce travail est loin d'être terminé. La philosophie présentée ici ne se réduit d'ailleurs pas aux systèmes quantiques. L'approche fonctionnelle « abstraite » à l'implantation des objets géométriques peut être exploitée dans un cadre beaucoup plus classique ; depuis plusieurs années nous enseignons le graphisme et la synthèse

6. … et avec le progrès de la physique la situation s'empire ; la notion d'« élémentarité » est de plus en plus floue, et dans l'espace-temps courbe même l'existence des particules devient la fonction de l'observateur, le vide pour les uns (ceux qui tombent dans un trou noir) est plein de radiation (de Hawking) pour les observateurs inertiaux lointains…

d'images de façon la plus *invariante* possible, où les vecteurs, les surfaces, etc. sont des objets fonctionnels indépendants du système de coordonnées (p. ex., lié à la description de la caméra virtuelle ou d'un autre dispositif du rendu). Ceci fait que les formules sont universelles et plus faciles à mémoriser.

Cependant c'est le domaine quantique où cette universalité acquiert une nouvelle dimension ; tandis que dans le graphisme c'est un choix qui favorise la clarté de présentation sur l'efficacité du rendu, dans la description des systèmes quantiques c'est une démarche méthodologique presque obligatoire, qui refuse l'attachement au *système* simulé des connaissances accessoires qui définissent le *contexte d'observation*.

Dans les mathématiques modernes, y compris leurs applications (donc, la physique encore une fois), le statut des équivalences, homo- ou isomorphismes, l'existence de foncteurs permettant de jeter un pont entre des modèles mathématiques structurellement différents, etc. est devenu extrêmement important. Ceci constitue une des motivations du développement de la théorie des catégories, et la motivation primaire pour que les Naturalistes dont l'éducation mathématique n'est pas « abstraite » du tout, commencent à s'intéresser par les catégories.

Même si ces Naturalistes restent fascinés par la vitesse des calculs informatisés, codés par des méthodes les plus rapides (et brutales…) existantes, on ne peut échapper au trend global, le fait que la programmation, y compris les techniques de simulation, devient de plus en plus abstraite, générique, universaliste.

Références

[1] Julia Wallace, site **`www.dcs.ex.ac.uk/~jwallace/simtable.html`** ; (2002).

[2] Bernhard Ömer, *Procedural Formalism for Quantum Computing*, (1998), disponible à **`tph.tuwien.ac.at/~oemer`**.

[3] Peter W. Shor, *Polynomial-Time Algorithm for Prime Factorization and Discrete Logarithms on a Quantum Computer*, SIAM Review **41**, n° 2, (1999), pp. 303–332.

[4] L.K. Grover, *Quantum Mechanics Helps In Searching For a Needle in a Haystack*, Phys. Rev. Lett. **79**, (1997), p. 325. Aussi : L.K. Grover, *A fast quantum mechanical algorithm for database search*, Proc. 28th ACM Symp. on Theory of Computation, (1996), p. 212.

[5] Richard P. Feynman, *Simulating physics with computers*, Int. J. Theor. Phys. **21**, (1982), p. 467.

[6] John Preskill, *Quantum Information and Computation*, Lecture Notes for Physics 229, California Institute of Technology, (1988).

[7] C. Zalka, C. *Efficient simulation of quantum systems by quantum computers*. Online preprint quant-ph/9603026, (1996).

[8] S. Brandt, H.-D. Dahmen, *Quantum Mechanics on the Personal Computer*, Springer, (1994).

[9] W.K. Wootters, W.H. Zurek, *A single quantum cannot be cloned*, Nature **299**, (1982), p. 802.

[10] **`http://www.haskell.org`** .

[11] Daniel Kastler, *Introduction à l'électrodynamique quantique*, Dunod, Paris, (1960).

[12] Diederik Aerts, Ingrid Daubechies, *Physical justification for using the tensor product to describe two quantum systems as one joint system*, Helvetica Physica Acta, **51**, (1978), pp. 661–675.

[13] C. H. Bennet, IBM J. Res. Develop. **17**, (1973), p. 525. Voir aussi : C. H. Bennet, SIAM J.Comput. **18**, (1989), p. 766.