

Emulation d'un ordinateur quantique

Laureline Pinault

7 juin 2013

Table des matières

1	Modélisation des composants d'un ordinateur quantique	2
1.1	Le bit quantique ou qubit	2
1.2	Les portes logiques quantiques	3
2	Algorithmes quantiques	5
2.1	Un exemple d'algorithme quantique : l'algorithme de Deutsch-Josza	5
2.2	Mise en oeuvre informatique de cet algorithme	5
A	Notation de Dirac ou notation bra-ket	8
B	Portes courantes	8
C	Extraits du code en Caml Light	9
D	Preuve de la condition nécessaire et suffisante de non intrication d'un qubit de taille 2¹	14
E	Preuve du théorème de décomposition ZY²	16
F	Preuve de la correction de l'algorithme de Deutsch³	18
G	Bibliographie	19

1. Nous avons réalisé et rédigé cette preuve, la bibliographie se contentant de mentionner la propriété.

2. Nous avons réalisé et rédigé cette preuve, la bibliographie estimant qu'il s'agissait d'une trivialité.

3. Pour le détail des calculs, on se référera avec profit à [4], p.33 ou [3], p.48.

Introduction

Avant de nous lancer dans de grands exposés, commençons par définir certains termes :

- Un *calculateur* est une machine effectuant des calculs, qu'ils soient de nature arithmétique, logique ou algébrique. Par exemple, une calculatrice ou un ordinateur peuvent être considérés comme des calculateurs.
- Un *calculateur quantique* est un calculateur qui fonctionne suivant les lois de la mécanique quantique. Même si leur réalisation physique en est encore au stade embryonnaire, la théorie de l'information quantique et les algorithmes quantiques sont, eux, déjà bien développés et font l'objet de recherches intenses depuis des dizaines d'années.
- Par ailleurs, l'*émulation* consiste à remplacer un composant physique – ici, le calculateur quantique – par un logiciel.

Nous avons travaillé à deux sur ce projet. Afin de mieux comprendre le fonctionnement d'un calculateur quantique, nous avons codé en Caml Light⁴ un émulateur. Sur cet émulateur nous avons ensuite pu faire tourner des algorithmes.

Ce document présente en première partie les composants de base d'un calculateur quantique, les qubits ou bits quantiques et les portes logiques quantiques, et la manière dont nous les avons modélisés en restant très proche du formalisme à la fois mathématique et physique. La deuxième partie permet ensuite d'expliquer la façon dont nous pouvons faire tourner des algorithmes quantiques sur cet émulateur. Les conventions d'écriture adoptées sont celles de la notation de Dirac, aussi appelée notation bra-ket. Elles sont introduites au fur et à mesure du rapport et regroupées dans l'annexe A.

1 Modélisation des composants d'un calculateur quantique

Quelle que soit sa réalisation physique, un calculateur se compose de deux types de composants :

- Des unités élémentaires de stockage de l'information. Celles-ci sont appelées bits et peuvent prendre la valeur 0 ou 1 dans les calculateurs que nous côtoyons (calculatrices, ordinateurs, ...) et qui fonctionnent selon les principes de la physique classique. L'analogue du bit pour les calculateurs quantiques est nommé qubit, contraction de *quantum bit*.
- Des opérateurs qui agissent sur ces unités élémentaires. Ceux-ci sont nommés portes logiques, que ce soit pour les calculateurs classiques ou les calculateurs quantiques.

1.1 Le bit quantique ou qubit

Qubit de taille 1⁵ : L'état d'un qubit est une superposition des états classiques $|0\rangle$ et $|1\rangle$. Mathématiquement on le représente comme une combinaison linéaire de ces deux états :

$$|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$$

Si on décide de mesurer $|\varphi\rangle$, on obtiendra $|0\rangle$ avec la probabilité $|\alpha|^2$ et $|1\rangle$ avec la probabilité $|\beta|^2$ (règle de Born). Ceci implique qu'on ait nécessairement (somme des probabilités) :

$$|\alpha|^2 + |\beta|^2 = 1$$

De manière plus formelle, si on note $V = Vect(|0\rangle, |1\rangle)$, \mathbb{C} -ev de dimension 2, un qubit est un vecteur unitaire de V .

Une différence importante entre les qubits et les bits classiques est qu'on ne peut en général pas dupliquer un qubit, sauf s'il est dans un état classique, $|0\rangle$ ou $|1\rangle$ (théorème de non-clonage quantique⁶).

Qubit de taille quelconque : Si on considère deux qubits de taille 1 $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$ et $|\psi\rangle = \gamma|0\rangle + \delta|1\rangle$, on peut considérer l'ensemble qu'ils forment en en faisant le produit tensoriel⁷ :

$$\begin{aligned} |\varphi\rangle \otimes |\psi\rangle &= \alpha\gamma(|0\rangle \otimes |0\rangle) + \alpha\delta(|0\rangle \otimes |1\rangle) + \beta\gamma(|1\rangle \otimes |0\rangle) + \beta\delta(|1\rangle \otimes |1\rangle) \\ &= \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle \end{aligned}$$

L'état obtenu est un vecteur unitaire de $V \otimes V$, \mathbb{C} -ev de dimension 4. En tant que superposition des états de base $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$, tout vecteur unitaire de $V \otimes V$ est un qubit. C'est ce que j'appelle un qubit de taille 2. Cette représentation a de l'intérêt car elle permet de modéliser l'intrication quantique : tout vecteur unitaire de $V \otimes V$ ne peut pas s'écrire sous la forme $|\varphi\rangle \otimes |\psi\rangle$ où $|\varphi\rangle$ et $|\psi\rangle$ sont des vecteurs unitaires de V :

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \text{ n'est pas dans un état intriqué} \Leftrightarrow \alpha_{00}\alpha_{11} = \alpha_{01}\alpha_{10}$$
⁸

4. En raison de ce choix de langage de programmation, les tableaux et matrices sont numérotés à partir de 0.

5. La notion de taille d'un qubit n'est pas officielle, c'est une dénomination personnelle que j'expliquerai par la suite.

6. Voir [4] p.24 et p.532.

7. En tant que vecteur, un qubit peut être vu comme un tenseur d'ordre 1 de V . Le produit tensoriel donne alors un tenseur d'ordre 2 qui peut également être vu comme un tenseur d'ordre 1 (donc assimilable à un vecteur) de l'espace $V \otimes V$.

8. Pour la preuve de ce résultat, voir l'annexe D.

On peut généraliser à un qubit de taille n : un qubit de taille n ⁹ est un vecteur unitaire du \mathbb{C} -ev de dimension 2^n $V^{\otimes n} = \text{Vect}(|\bar{p}^2\rangle, p \in [0; 2^n - 1])$:

$$|\psi\rangle = \sum_{p=0}^{2^n-1} \alpha_p |\bar{p}^2\rangle \text{ où } |\alpha_p|^2 \text{ représente la probabilité de mesurer l'état } |\bar{p}^2\rangle \text{ (règle de Born généralisée)}$$

Implémentation informatique : De manière assez naturelle, nous avons représenté un qubit de taille n par un vecteur colonne de taille 2^n contenant les coordonnées du qubit dans la base corespondante (les α_p) :

Type synonyme : `type qubit == complex vect vect;;`¹⁰

Principales fonctions implémentées :

- Pour réaliser des produits tensoriels entre deux qubits de tailles quelconques : `mult_qubit : complex vect vect → complex vect vect → complex vect vect = <fun>`.
- Pour générer aléatoirement un qubit de taille 1 : `genere_qbit : unit → complex vect vect = <fun>`. La fonction génère aléatoirement les modules et les phases des coefficients.
- Pour générer aléatoirement un qubit de taille n : `genere_qbit_n : int → complex vect vect = <fun>`. La fonction utilise `genere_qbit` pour générer n qubits de taille 1 de manière aléatoire, puis la fonction `mult_qubit` pour en réaliser le produit tensoriel.
- Pour générer l'état de la base $|0\rangle$: `genere_qbit_zero : unit → complex vect vect = <fun>`. La fonction renvoie tout simplement la matrice `[|1|]; [|0|]`¹¹.

1.2 Les portes logiques quantiques

Une porte logique quantique est un opérateur linéaire unitaire¹² qui s'applique à un qubit d'une taille donnée. On note : $U : |\psi\rangle \rightarrow U|\psi\rangle = |U\psi\rangle$. Le caractère unitaire est dû au fait que le vecteur résultant de l'opération U est également un qubit, donc unitaire : l'opérateur doit conserver la norme. Elles sont représentées dans les circuits par des schémas comme celui-ci :

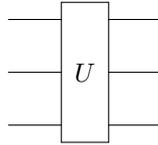


FIGURE 1 – Porte s'appliquant à 3 qubits / un qubit de taille 3.

Implémentation informatique : Du fait de la linéarité des opérateurs, ceux-ci peuvent être décrits par leur action sur les états de la base. On représente donc un opérateur s'appliquant à un qubit de taille n par une matrice de $\mathcal{M}_{2^n}(\mathbb{C})$. Il suffit ensuite pour appliquer une porte logique à un qubit de réaliser un simple produit matriciel.

Type synonyme : `type porte_logique == complex vect vect;;`

Portes contrôlées : Un type particulier de portes sont les portes contrôlées. Elles agissent sur un nombre $k + m$ de qubits où $k \geq 1$ est le nombre de qubits de contrôle et $m \geq 1$ le nombre de qubits cibles. On considère une porte U agissant sur m qubits. La porte Controlled- U agit sur les états de la base en appliquant la porte U aux m qubits cibles si et seulement si les k qubits de contrôle sont dans l'état $|1\rangle$.

On peut, de manière simple, implémenter la porte Controlled- U à partir de la matrice de U , la matrice représentant la porte Controlled- U étant une matrice par bloc : $\begin{pmatrix} I_k & 0 \\ 0 & U \end{pmatrix}$

9. A noter que lorsque je parle de n qubits auxquels s'appliquent une porte par exemple, je pourrais tout aussi bien dire que la porte s'applique à un qubit de taille n .

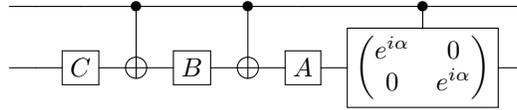
10. Un vecteur colonne de taille 2^n correspond en Caml Light à une matrice de $\mathcal{M}_{2^n,1}(\mathbb{C})$, d'où le type `vect vect`.

11. Cela peut paraître peu satisfaisant mais deux raisons justifient notre choix. D'une part on pourrait tout aussi bien le faire en mesurant un qubit généré aléatoirement, et en appliquant la porte `Not` si on obtient $|1\rangle$. D'autre part la question ne se pose pas en ces termes lors de la réalisation physique d'un calculateur car chaque réalisation physique a ses moyens spécifiques de générer un état de la base (par exemple dans le cas d'une réalisation avec des ions piégés, on peut imaginer un refroidissement des dits-ions par effet Doppler afin de les amener dans leur état fondamental, qui constituerait alors l'état $|0\rangle$).

12. On notera que ceci implique que l'opérateur soit réversible.

Cependant, étant donnée la difficulté de construire physiquement des portes multi-qubits, il peut être intéressant de chercher une décomposition de ces portes¹³. Pour cela on procède en plusieurs étapes : on décompose d'abord les portes s'appliquant à un qubit de contrôle et à un qubit cible, puis on utilise ce résultat pour le cas général.

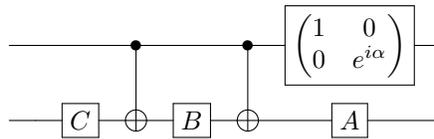
On cherche donc à construire la porte Controlled-U où U est une porte qui s'applique à 1 qubit à partir de portes à 1 qubit et d'une porte contrôlée particulière : la porte Controlled-Not. Pour cela on utilise le théorème de décomposition ZY¹⁴ qui nous assure de l'existence d'opérateurs unitaires A, B et C (agissant sur un seul qubit) tels que $ABC = I$ et $U = e^{i\alpha}AXBXC$, où α est un facteur de phase global. Le circuit équivalent sera alors¹⁵ :



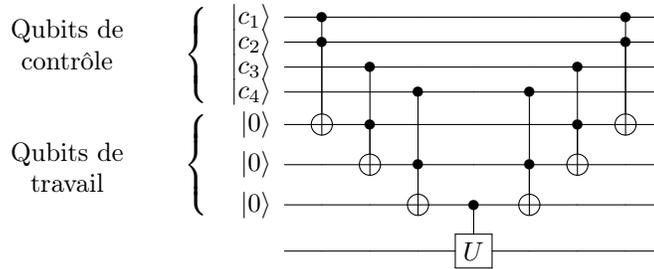
Or le circuit équivalent de la porte Controlled- $e^{i\alpha}I_2$ est : $\begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}$ car sa matrice est :

$$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\alpha} & 0 \\ 0 & 0 & 0 & e^{i\alpha} \end{pmatrix}$$

Le circuit équivalent de la porte contrôlée sera donc :



On peut ensuite construire une porte Controlled-U à 1 qubit cible et m qubits de contrôle de la manière suivante¹⁶ :



De plus, on peut décomposer toute porte en un circuit composé uniquement de portes à 1 qubit U_i et de Controlled-Not¹⁷. En remplaçant chaque porte U_i de ce circuit par une porte Controlled- U_i à m qubits de contrôle, et chaque Cnot par une porte de Toffoli, on obtient le circuit équivalent de la porte contrôlée. Si U_i est la porte de Controlled-Not, on utilise la décomposition de la porte de Toffoli (ou de Controlled-Not généralisé à m qubits de contrôle). Si U_i est une porte élémentaire à 1 qubit, on se ramène au paragraphe précédent.

Une opération particulière, la mesure : La mesure est une opération qui est rangée à tort parmi les portes logiques car elle n'est ni linéaire ni réversible. Si en classique il s'agit d'une opération évidente (il suffit de regarder la valeur du bit), en quantique il ne s'agit pas d'une opération anodine. En effet elle fige le qubit mesuré dans un des états de la base, détruisant ainsi les éventuels états de superposition et d'intrication du qubit sur lequel on est en train de travailler. Toute l'ingéniosité des algorithmes quantiques consiste donc à agencer astucieusement les portes pour récupérer l'information utile au bon moment et au bon endroit à l'aide d'une mesure.

13. Nous avons implémenté cette décomposition.

14. Pour plus de détail, voir l'annexe E.

15. En effet, si le qubit de contrôle est à $|0\rangle$, le qubit cible deviendra $ABC|\psi\rangle$ or $ABC = I$ donc il reste inchangé. Si le qubit de contrôle est à $|1\rangle$, le qubit cible deviendra $e^{i\alpha}AXBXC|\psi\rangle = U|\psi\rangle$, ie U est appliquée au qubit cible.

16. La première porte enregistre le produit $|c_1c_2\rangle$ dans le premier qubit de travail. Ensuite la deuxième porte enregistre le produit $|c_1c_2c_3\rangle$ dans le deuxième qubit de travail, et ainsi de suite. Le dernier qubit de travail ($|c_1c_2\dots c_m\rangle$) vaudra donc $|1\rangle$ si et seulement si tous les qubits de contrôle sont à $|1\rangle$. On applique ensuite Controlled-U avec un seul qubit de contrôle, puis on réalise les mêmes opérations en sens inverse pour que les qubits de travail retournent dans leur état initial.

17. Référence de [4] à retrouver

Pour l'implémentation de cette opération, nous avons écrit une fonction `mesure`. Lorsqu'on veut mesurer un qubit $|\psi\rangle = \sum_{p=0}^{2^n-1} \alpha_p |\bar{p}^2\rangle$, la fonction crée un tableau contenant dans la p-ième case la somme des $|\alpha_i|^2$ pour $i \leq p$. Ensuite elle génère de manière aléatoire un nombre compris entre 0 et 1 et regarde à quelle case (et donc à quel état) il correspond.

2 Algorithmes quantiques

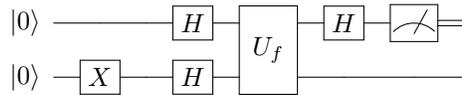
A priori après avoir modélisé informatiquement les composants du calculateur quantique, faire tourner différents algorithmes dessus ne devrait pas être la partie la compliquée. Il se trouve que nous nous sommes retrouvées confrontées à un problème inattendu. Nous n'avons finalement, par manque de temps, implémenté entièrement qu'un seul algorithme, l'algorithme de Deutsch-Josza. Je présente d'abord le principe de cet algorithme puis son implémentation informatique.

2.1 Un exemple d'algorithme quantique : l'algorithme de Deutsch-Josza

Problème et algorithme de Deutsch : Le problème et l'algorithme de Deutsch sont une version simplifiée du problème et de l'algorithme de Deutsch-Josza.

Problème de Deutsch : On considère une fonction $f : \{0;1\} \rightarrow \{0;1\}$. Elle est soit constante, soit équilibrée¹⁸ (c'est-à-dire qu'elle prend deux valeurs distinctes en 0 et en 1). Le but¹⁹ est de savoir si f est constante ou équilibrée.

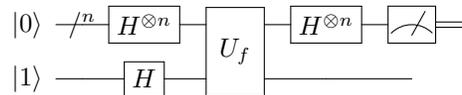
Algorithme de Deutsch : Avec un calculateur classique on n'a pas d'autre choix que d'évaluer successivement la valeur de f en 0 et en 1 pour avoir la réponse au problème de Deutsch. L'algorithme de Deutsch utilise le parallélisme quantique afin de trouver la solution à partir d'une seule mesure de f . Il repose sur la construction d'une transformation U_f qui effectue l'opération $|x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$ où le \oplus représente l'addition modulo 2.²⁰ Le circuit quantique correspondant à l'algorithme est le suivant²¹ :



Problème et algorithme de Deutsch-Josza : Il s'agit d'une généralisation de l'algorithme de Deutsch, vu précédemment.

Problème de Deutsch-Josza : On s'intéresse à une fonction $f : \{0;1\}^n \rightarrow \{0;1\}$ dont on sait par avance qu'elle est soit constante, soit équilibrée (c'est-à-dire égale à 0 sur la moitié des valeurs et à 1 sur l'autre moitié). On cherche à savoir si la fonction est constante ou équilibrée.

Algorithme de Deutsch-Josza : Tout comme pour l'algorithme de Deutsch, l'algorithme quantique ne nécessite qu'une seule évaluation de f grâce à l'oracle quantique, $U_f : |x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$ où le \oplus représente l'addition modulo 2 et où $|x\rangle$ représente un qubit de taille n ²². Le circuit quantique correspondant à l'algorithme est le suivant :



2.2 Mise en oeuvre informatique de cet algorithme

Principe : Nous ne pensions pas rencontrer de problèmes particuliers pour implémenter cet algorithme puisqu'il suffit a priori d'appliquer successivement les différentes portes en faisant des produits matriciels. Il devait juste servir de test pour valider notre implémentation avant qu'on ne se lance dans des algorithmes plus complexes. Nous avons cependant été confrontées à un problème inattendu. En effet, pour appliquer la porte U_f nous avons réalisé le produit tensoriel des $n+1$ qubits. Or si, d'un point de vue théorique, parler de $n+1$ qubits ou d'un qubit de taille $n+1$ est analogue tant que ces qubits ne sont pas intriqués, avec notre implémentation il n'est pas possible de séparer des qubits non intriqués sans avoir connaissance du système, ce qui est exclu au regard des principes gouvernant notre calculateur.

18. Le terme anglais est «balanced».

19. Qui peut a priori paraître artificiel, mais prend tout son sens s'il s'agit d'une petite étape de calcul parmi des milliers d'autres !

20. On peut se demander pourquoi on n'effectue pas tout simplement une transformation $|x\rangle \mapsto |f(x)\rangle$. La réponse est que cette transformation n'est a priori pas unitaire (en fait elle ne l'est pas si f n'est pas bijective). U_f , au contraire, l'est puisque son carré est l'identité ($f(x) \oplus f(x) = 0$).

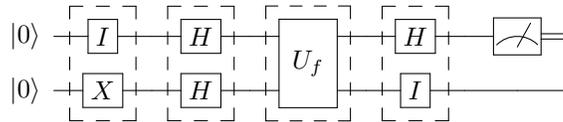
21. Pour la description des portes quantiques utilisées, se reporter à l'annexe B

22. La porte U_f s'applique donc à un qubit de taille $n + 1$.

Nous avons finalement résolu ce problème grâce à cette simple constatation : toutes les portes étant des opérateurs linéaires, le circuit global est lui-même un opérateur linéaire qui peut donc être décrit par son action sur les états de la base. Nous avons donc créé une fonction qui crée la matrice équivalente à un circuit donné en entrée, nous permettant ainsi de faire tourner sur notre émulateur n'importe quel algorithme donné sous la forme d'un circuit quantique. Pour cela nous procédons en deux étapes :

- Nous commençons par "découper" le circuit en zones verticales contenant au maximum une porte par qubit. Les endroits où il n'y a pas de portes seront ensuite remplacés par des portes identités. Nous réalisons ensuite la matrice équivalente de chaque zone en réalisant des produits tensoriels entre les matrices. On utilise pour cela la fonction `matrix_of_gates`.
- Nous réalisons ensuite des produits matriciels des matrices ainsi obtenues afin d'avoir la matrice équivalente à tout le circuit. On utilise pour cela la fonction `matrix_of_circuit`.

Exemple avec l'algorithme de Deutsch : Voici par exemple la division en zones du circuit représentant l'algorithme de Deutsch :



Implémentation :

`matrix_of_gates` : La fonction prend en entrée une des zones du circuit, donnée sous la forme d'un tableau de matrices (les matrices étant la représentation des portes). Par exemple pour lui demander la matrice équivalente de la première zone du circuit de l'algorithme de Deutsch, on lui donne en entrée : $[[I; X]]$.

`matrix_of_circuit` : La fonction prend en entrée le circuit en entier, donné sous la forme d'un tableau de zones (ie. un tableau de tableau de matrices). Par exemple pour lui demander la matrice équivalente du circuit de l'algorithme de Deutsch, on lui donne en entrée : $[[[I; X]]; [H; H]]; [U_f]; [H; I]]$.

Conclusion

Nous avons pu tester le fonctionnement de notre émulation et même faire quelques petites applications de l'algorithme de Deutsch. Les résultats sont satisfaisants. Cependant, notre travail est loin de couvrir l'ensemble de ce que nous aurions pu faire pour plusieurs raisons :

- Nous avons implémentés un seul algorithme, qui est très simple : l'algorithme de Deutsch, l'implémentation d'algorithmes plus complexes comme l'algorithme de Grover et l'algorithme de Shor restant en chantier.
- Nous ne nous sommes pas intéressés à la correction d'erreur, un problème qui se pose en informatique en général et de manière plus prononcé encore en quantique du fait de l'échelle du système²³ d'une part et de la décohérence d'autre part. Ce problème est d'autant plus important qu'il ne peut pas se résoudre de la même manière qu'en classique, la duplication et la mesure des qubits n'étant pas possible en cours d'algorithme.
- Nous n'avons pas approfondi la partie physique importante liée à ce type, nous n'avons notamment regardé les réalisations physiques que sous un angle qualitatif sans rentrer dans les détails calculatoires.
- Nous avons fait le choix d'une implémentation entièrement matricielle, restant ainsi très proches du formalisme mathématique et physique, malgré les coûts (temporels et spatiaux) et les limitations auxquels nous risquons d'être confrontés²⁴. Nous ne nous sommes en effet pas du tout intéressés aux questions d'optimisation qui auraient pu être étudiées²⁵. Cependant pour ce que nous avons implémenté, l'algorithme de Deutsch-Jozsa, nous n'avons pas eu de problèmes liés à notre modélisation. Le temps de réponse était quasi-instantané.
- Nous n'avons pas, contrairement au but premier d'une émulation, développé séparément un module qui simule le fonctionnement d'un ordinateur quantique et un module qui comprendrait les algorithmes et serait directement lisible par un ordinateur quantique réel. Il nous a manqué pour cela la connaissance de l'architecture qui serait adoptée par un ordinateur quantique réel. L'idéal aurait été de créer un langage quantique (bas niveau) correspondant à cette architecture.
- Si on peut, en théorie, implémenter n'importe quel algorithme quantique sur notre émulateur, cette implémentation nécessite d'avoir accès au circuit quantique représentant cet algorithme, ce qui n'est a priori pas facile à obtenir pour un algorithme quelconque²⁶. Pour autant, un certain nombre d'algorithmes ont une représentation directe sous forme

23. En informatique classique, l'état d'un bit est lié au mouvement d'environ 10^5 particules tandis qu'en informatique quantique il s'agit d'un changement d'état d'une particule, celle-ci pouvant être un atome, un ion ou un photon.

24. Caml s'arrête à 2^{30} , soit 30 qubits... !

25. Que ce soit pour les calculs de multiplications matricielles où des algorithmes avec mémorisation existent ou pour la compression de données.

26. C'est une autre raison pour laquelle le développement d'un langage quantique aurait pu être intéressant.

- de circuits quantiques. C'est le cas de la plupart des algorithmes s'appuyant sur des oracles.
- Bien que nous ayons pu appréhender certaines différences entre la logique quantique et la logique classique, et comprendre comment le parallélisme quantique agissait par exemple dans le cas de l'algorithme de Deutsch, nous n'avons pas réellement réussi à dégager des critères pour qu'un algorithme se résolve plus efficacement en quantique qu'en classique, ce qui était le but premier de notre Tipe. Il s'agit en fait d'un problème ouvert.

Annexe A : Notation de Dirac ou notation bra-ket

Aussi appelée notation bra-ket, elle est très utilisée en mécanique quantique. Dans mon rapport, il s'agit d'une simple convention d'écriture, je la présente donc en tant que telle.

La notation de Dirac consiste fondamentalement à écrire :

- Un vecteur de l'espace des états sous la forme $|u\rangle$, ce qui est appelé un ket.
- Un vecteur de l'espace dual sous la forme $\langle u|$, ce qui s'appelle un bra.
- Le produit scalaire hermitien de deux vecteurs $|\varphi\rangle$ et $|\phi\rangle$ sous la forme $\langle\varphi|\psi\rangle$.

Le produit tensoriel de deux qubits $|\varphi\rangle$ et $|\psi\rangle$ peut s'écrire aussi bien $|\varphi\rangle \otimes |\psi\rangle$ que $|\varphi \otimes \psi\rangle$, $|\varphi\rangle|\psi\rangle$ ou $|\varphi\psi\rangle$.

L'action d'un opérateur linéaire (unitaire) U sur un qubit $|\psi\rangle$ peut s'écrire aussi bien $U|\psi\rangle$ que $|U\psi\rangle$.

Annexe B : Portes courantes

Porte identité : La porte identité n'a, comme son nom l'indique, absolument aucune action sur le circuit.

Son schéma est le suivant : 

Sa matrice est tout simplement la matrice identité : $I = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

Porte X / Not : La porte X est l'équivalent du Not classique (c'est pourquoi elle est également appelée porte Not) : elle agit sur les états de la base comme Not.

Son schéma est le suivant : 

Sa matrice est : $X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

Porte de Hadamard : La porte de Hadamard permet de mettre les états de la base à équiprobabilité : $\begin{cases} |0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{cases}$

Son schéma est le suivant : 

Sa matrice est : $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$

Porte Controlled-Not / Cnot : Très utilisée, elle agit sur un qubit de contrôle et un qubit cible. Sur les états de la base, elle applique X (Not) au qubit cible si et seulement si le qubit de contrôle est à $|1\rangle$.

Son schéma est le suivant (le rond noir représente le qubit de contrôle, le + entouré représente le qubit cible) : 

Sa matrice est : $Cnot = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$

Porte de Fredkin / Controlled-Swap : Elle agit sur 3 qubits : un qubit de contrôle et deux qubits cible. Son action sur les états de la base est d'échanger les qubits cibles si et seulement si le qubit de contrôle est à $|1\rangle$.

Son schéma est le suivant : 

Sa matrice est Fredkin = $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$

Porte mesure : Bien que ce ne soit pas une porte logique à strictement parler, l'opération qui consiste à mesurer un qubit est usuellement rangée parmi celles-ci, d'où sa place dans cette section. Voici son schéma : 

Annexe C : Extraits du code en Caml Light

Je n'en donne ici que les fonctions intéressantes, on ne trouvera pas par exemple l'implémentation des nombres complexes ou du produit matriciel.

mult_qubit : multiplication de qubits.

La fonction `mult_qubit` réalise le produit tensoriel de deux qubits de tailles quelconques p et q donnés en entrée sous la forme de vecteurs colonnes de tailles respectives 2^p et 2^q . Elle renvoie un vecteur colonne de taille 2^{p+q} .

```

1 (* mult_qubit : complex vect vect -> complex vect vect -> complex vect vect = <fun> *)
2 let mult_qubit p q =
3   let l = taille_qubit p in let m = taille_qubit q in
4   let k = vect_length p in let n = vect_length q in
5   let res = make_matrix (k*n) 1 {Re=0. ; Im=0.} in
6   for i = 0 to (k-1) do
7     for j = 0 to (n-1) do
8       res.(n*i+j).(0) <- mult p.(i).(0) q.(j).(0) ;
9     done ;
10  done ;
11  res ;;

```

Fonctionnement de mult_qubit : On multiplie chaque composante du premier tenseur par les composantes du deuxième tenseur, selon le schéma ci-dessous :

$$\begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{2^p-1} \end{pmatrix} \otimes \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{2^q-1} \end{pmatrix} = \begin{pmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \vdots \\ \alpha_0\beta_{2^q-1} \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \\ \vdots \end{pmatrix}$$

FIGURE 2 – Produit tensoriel de deux qubits

Formellement, si on note $(p_i)_{1 \leq i \leq n}$ le premier tenseur et $(q_j)_{1 \leq j \leq m}$, la composante s_i du tenseur $s = p \otimes q$ sera $p_k q_r$ où on a $i = km + r$ (division euclidienne de i par m).

genere_qbit et genere_qubit_n : génération aléatoire de qubits.

La fonction `genere_qbit` permet de générer aléatoirement un qubit de taille 1, tandis que la fonction `genere_qubit_n` génère un qubit d'une taille n donnée en entrée.

```

1 (*genere_qbit : unit -> complex vect vect = <fun>*)
2 let genere_qbit () =
3   let al = random_float 1. in (*limite non comprise*)
4   let pi = 4. *. atan 1. in
5   let theta = random_float 2. *. pi in
6   let phi = random_float 2. *. pi in
7   let a = sqrt al in let b = sqrt (1. -. al) in
8   [| [|{Re = a *. cos theta ; Im = a *. sin theta}]] ; [|{Re = b *. cos phi ; Im = b *. sin phi}]] || ;;

```

```

1 (*genere_qubit_n : int -> complex vect vect = <fun>*)
2 let genere_qubit_n n =
3   let res = ref (genere_qbit()) in
4   for i = 2 to n do
5     res := (mult_qubit !res (genere_qbit())) ;

```

```
6 done ; !res ;;
```

Fonctionnement de genere_qbit : On cherche à générer deux coefficients complexes aléatoires, α et β , qui doivent cependant vérifier la relation $|\alpha|^2 + |\beta|^2 = 1$. On génère aléatoirement un réel a_1 dans $[0;1]$, et deux angles θ et ϕ dans $[0;2\pi]$ grâce à la fonction `random_float`. a_1 sera le carré du module a de α dont on déduit le module b de β à partir de la condition de normalisation. On a alors deux nombres réels (deux modules) qui vérifient cette condition, il suffit donc de rajouter deux phases quelconques : ce sont les angles θ et ϕ .

Fonctionnement de genere_qubit_n : On se contente de générer n qubits de taille 1 avec `genere_qubit`, puis de réaliser un produit tensoriel entre eux (avec `mult_qubit`)²⁷.

mesure_aux et mesure : mesure d'un qubit.

La fonction `mesure` permet de mesurer un qubit de taille quelconque n . Pour cela elle se sert d'une fonction auxiliaire : `mesure_aux` qui renvoie l'état mesuré sous la forme d'une chaîne de caractère, ce qui est le type choisi pour représenter des états²⁸. La fonction `mesure` récupère cette information et fige le qubit dans l'état mesuré.

```
1 (* mesure_aux : complex vect vect -> string = <fun> *)
2 exception trouve of string;;
3 let mesure_aux x =
4   let n = taille_qubit x in
5   let proba = make_vect (vect_length x) (norm2 x.(0).(0)) in
6   let m = random__float 1. in
7   try
8     for i=1 to (vect_length x)-1 do
9       if m <= proba.(i-1) then raise (trouve (bin_of_dix (i-1)));
10      proba.(i) <- (norm2 x.(i).(0)) +. proba.(i-1);
11 done; (doctor [|bin_of_dix ((vect_length x)-1)|] n).(0)
12 with trouve s -> (doctor [|s|] n).(0)
```

```
1 (* mesure : complex vect vect -> string = <fun> *)
2 let mesure x =
3   let s = mesure_aux x in
4   for i=0 to (vect_length x)-1 do
5     match i with
6     | a when a = dix_of_bin s -> x.(i).(0) <- un
7     | _ -> x.(i).(0) <- zero;
8 done; s;;
```

Fonctionnement de mesure_aux : On crée un tableau `proba`. La case p du tableau contiendra la somme des $|\alpha_i|^2$ pour $i \leq p$ ce qui correspond à la probabilité que la mesure du qubit renvoie un des états situés avant p . La dernière case contiendra donc automatiquement 1. Ensuite, on génère aléatoirement un réel m entre 0 et 1, et on cherche dans le tableau des probabilités cumulées à quel état il correspond (le plus petit p tel que m soit inférieur au contenu de la case p du tableau). Par souci d'économie on ne fait qu'une boucle `for` qui se charge en même temps de tester la case i du tableau et de remplir la case $i+1$ si ce n'est pas la case i qui correspond à l'état mesuré. On interrompt cette boucle avec une exception nommée `trouve` lorsqu'on trouve l'état mesuré et on renvoie ce dernier.

Fonctionnement de mesure : On récupère l'état mesuré (sous forme de chaîne de caractères) avec `mesure_aux`. On parcourt ensuite le tableau représentant le qubit mesuré; à chaque fois, si l'indice correspond à l'état qui a été mesuré, on met le coefficient à 1, sinon à 0.

decomposition et controlled_one : implémentation des portes contrôlées à un qubit de contrôle et un qubit cible.

La fonction `decomposition` renvoie α , A , B et C en utilisant les expressions obtenues à la fin de la preuve. La fonction `controlled_one` s'en sert ensuite pour réaliser le circuit équivalent et renvoyer sa matrice (grâce à la fonction `matrix_of_circuit`).

```
1 (* decomposition : complex vect vect -> float * complex vect vect * complex vect vect * complex vect vect =
   <fun> *)
2 let decomposition U =
```

27. A noter que la fonction `genere_qubit_n` ne génère donc pas de qubits dans un état intriqué.

28. Le choix d'une chaîne de caractère plus que d'un entier vient de la nécessité de différencier l'état $|1\rangle$ de l'état $|01\rangle$ par exemple.

```

3 let a,b,c,d = zy_decomposition U in
4 let A = mult_matrix (Rz b) (Ry (c/. 2.)) in
5 let B = mult_matrix (Ry (-. c /. 2.)) (Rz (-. (b +. d) /. 2.)) in
6 let C = Rz ((d -. b) /. 2.) in
7 (a,A,B,C);;

```

```

1 (* controlled_one : complex vect vect -> complex vect vect = <fun> *)
2 let controlled_one U =
3   let a,A,B,C = decomposition U in
4   let aux = [|[un;zero|];|[zero;exp_complex {Re=0.;Im=a}|]|] in
5
6   matrix_of_circuit [|[I;C|];|[CNOT|];|[I;B|];|[CNOT|];|[aux;A|]|];;

```

matrix_of_gates et matrix_of_circuit : construction de la matrice équivalente à un circuit.

La fonction `matrix_of_circuit` construit la matrice équivalent d'un circuit donné sous la forme d'un tableau de tableau de portes en utilisant une sous fonction `matrix_of_gates` qui construit la matrice équivalente d'une zone donnée sous la forme d'un tableau de portes.

```

1 (* matrix_of_gates : complex vect vect vect -> complex vect vect = <fun> *)
2 let matrix_of_gates tab =
3   let n = vect_length tab in (* nombre de portes *)
4
5   (* Tableau contenant le nombre de qubits des portes en correspondance *)
6   let lengths = make_vect n 0 in
7   for i=0 to (n-1) do
8     lengths.(i) <- taille_qubit tab.(i);
9   done;
10
11   (* Calcul du nombre total de qubits + Transposition des portes pour acceder aux colonnes en premier
12     indice *)
13   let m = ref 0 in (* m = nombre de qubits *)
14   for i=0 to (n-1) do
15     tab.(i) <- transpose tab.(i);
16     m:= !m + (lengths.(i));
17   done;
18   let p = deux_puissance !m in (* Nombre d'etats *)
19   let resultat = make_matrix p p zero in
20
21   let indice = indices !m lengths in (* Indices des colonnes a multiplier entre elles *)
22
23   for i=0 to (p-1) do (* parcourt les etats et les ensembles d'indices *)
24     let temp = ref (transpose tab.(0).(indice.(i).(0))) in
25     for j=1 to (n-1) do (* produit vectoriel des colonnes correspondant aux indices *)
26       temp := mult_qubit !temp (transpose tab.(j).(indice.(i).(j)));
27     done;
28     copy (transpose !temp) resultat.(i); (* enregistrement du resultat *)
29   done;
30   transpose resultat;; (* on a enregistre les colonnes-resultat comme lignes, donc on transpose *)

```

```

1 (* Transforme un circuit, divise en zones verticales, en matrice *)
2 (* matrix_of_circuit : complex vect vect vect vect -> complex vect vect = <fun> *)
3 let matrix_of_circuit tab =
4   let n = vect_length tab in
5   let resultat = ref (matrix_of_gates tab.(0)) in
6
7   let m = vect_length tab.(0) in (* on regarde la taille des matrices representant les zones *)
8   let fct = [|[mult_matrix;strassen|]|] in
9   let indice = ref 0 in
10  if m <= 4 then indice := 0 else indice := 1; (* on applique strassen si m est trop grand : limite a
11    definir plus precisement *)
12
13  for i=1 to (n-1) do
14    resultat := fct.(!indice) (matrix_of_gates tab.(i)) !resultat;
15  done;
16  !resultat;;

```

Fonctionnement de `matrix_of_gates` : Il s'agit uniquement de faire un produit tensoriel entre les matrices contenues dans le tableau passé en entrée. Pour cela on commence par transposer ces matrices afin d' avoir accès aux colonnes en premier

indice. Ensuite, la fonction `indices` (voir plus bas pour le fonctionnement) renvoie un tableau dont chaque sous-tableau représente les indices des colonnes (des matrices de chaque porte) dont on doit faire le produit tensoriel pour obtenir une colonne de la matrice finale. Pour finir, on fait diverses boucles afin de réaliser ces produits tensoriels et d'enregistrer à l'aide de `copy` (qui copie un tableau dans un autre) les colonnes obtenues dans la matrice `resultat`.

Fonctionnement de `matrix_of_gates` : On considère un circuit (`tab`) représenté sous forme de tableau de zones. On va créer les matrices correspondant à chacune de ces zones à l'aide de `matrix_of_gates` et les multiplier entre elles, successivement, pour obtenir la matrice finale, correspondant au circuit entier.

Uf et deutsch : réalisation informatique de l'algorithme de Deutsch.

Pour réaliser l'algorithme de Deutsch, il suffit de faire générer deux qubits dans l'état $|0\rangle$ et de leur appliquer le circuit dont on obtient la matrice équivalente grâce à `matrix_of_circuit`. Mais, dans l'algorithme, U_f est considérée comme une donnée du problème²⁹. Afin de pouvoir appliquer directement l'algorithme de Deutsch à une fonction f ³⁰, nous avons écrit une fonction `Uf` qui prend `f` en argument et renvoie la matrice représentant U_f ³¹. `Uf` est la fonction qui masque le coût de l'algorithme de Deutsch : comme nous n'utilisons pas de calculateur quantique, il nous est en effet nécessaire de calculer $f(0)$ et $f(1)$ explicitement. La fonction `deutsch` prend ainsi en argument la fonction f , puis renvoie 0 si f est constante et 1 si elle est équilibrée (elle renvoie en fait $f(0) \oplus f(1)$).

```
1 (*Uf : (int -> int) -> complex vect vect = <fun>*)
2 let Uf f =
3   let U = make_matrix 4 4 zero in
4   let F = [| f 0 ; f 1 |] in
5   U.(F.(0)).(0) <- un ;
6   U.(1-F.(0)).(1) <- un ;
7   U.(2+F.(1)).(2) <- un ;
8   U.(3-F.(1)).(3) <- un ;
9   U ;;
```

```
1 (* deutsch : (int -> int) -> int = <fun> *)
2 let deutsch f =
3   let p = genere_qubit_zero () in
4   let q = genere_qubit_zero () in
5   let tab = [| [| I;X |]; [| H;H |]; [| Uf f |]; [| H;I |] |] in
6   let resultat = apply (matrix_of_circuit tab) (mult_qubit p q) in
7   int_of_string (string_of_char (mesure resultat).[0]);; (* Il faut regarder le premier qubit seulement.
   Renvoie un int. *)
```

Fonctionnement de `Uf` : On commence par calculer $f(0)$ et $f(1)$ qu'on range dans un vecteur `F` puis on remplit la matrice de U_f en regardant l'action de f sur les quatre vecteurs de la base. U_f est en fait constituée de 0 et de 1, avec un 1 par ligne et par colonne. L'enjeu est de savoir à quelle ligne est placé le 1 dans chaque colonne. Pour cela on procède comme indiqué par le tableau ci-dessous (la troisième colonne donne la case où est placée le 1) :

Colonne 0	$U_f(00\rangle) = 0\rangle f(0)\rangle$	$U.\underbrace{(f(0))}_{\text{ligne } f(0)}.\underbrace{(0)}_{\text{colonne 0}}$
Colonne 1	$U_f(01\rangle) = 0\rangle 1 \oplus f(0)\rangle$	$U.\underbrace{(1-f(0))}_{\text{cf (*)}}.\underbrace{(1)}_{\text{colonne 1}}$
Colonne 2	$U_f(10\rangle) = 1\rangle f(1)\rangle$	$U.\underbrace{(2)}_{\text{Bloc 2x2 en bas à droite}} + f(1).\underbrace{(2)}_{\text{colonne 2}}$
Colonne 3	$U_f(11\rangle) = 1\rangle 1 \oplus f(1)\rangle$	$U.\underbrace{(2)}_{\text{Bloc 2x2 en bas à droite}} + \underbrace{1-f(1)}_{\text{cf (*)}}.\underbrace{(3)}_{\text{colonne 3}}$

(*) Additionner 0 ou 1 à 1 modulo 2 revient à soustraire respectivement 0 ou 1 à 1.

Uf_Jozsa et deutsch_jozsa : réalisation informatique de l'algorithme de Deutsch-Jozsa.

L'algorithme de Deutsch-Jozsa fonctionne de manière très analogue à l'algorithme de Deutsch. La fonction `Uf_Jozsa` est celle

29. La bibliographie ne détaille jamais la réalisation de U_f dans le cas où f est une fonction quelconque donnée en entrée, se contentant d'indiquer qu'il s'agit d'une «boîte noire».

30. Cela nous paraissait plus parlant.

31. La matrice est de taille $4*4$ car elle prend en entrée et renvoie en sortie deux qubits.

qui se charge de construire la matrice U_f à partir de la donnée de f et du nombre de qubits auxquels elle s'applique³². Il s'agit peu ou prou de la même porte que dans l'algorithme de Deutsch, à ceci près que, ici, f s'applique à n qubits et que U_f renvoie les n premiers qubits inchangés. Tout comme dans l'algorithme de Deutsch, U_f masque le coût de l'algorithme. La fonction `deutsch_jozsa` renvoie 0 si f constante et 1 si elle est équilibrée. Il est à noter que, à la différence de ce qu'on a fait jusqu'ici, le code suppose que $f : \{0;1\}^{n-1} \rightarrow \{0;1\}$.

```

1 (* Uf_Jozsa : (string -> complex) -> int -> complex vect vect vect = <fun> *)
2 let Uf_Jozsa f n =
3   let m = deux_puissance (n+1) in (* n |0> et 1 |1> *)
4   let resultat = make_matrix m m zero in
5   let In = identite (n+1) in
6   let tab = make_vect (m/2) zero in (* tableau de taille 2^n *)
7   (* Calcule les différentes valeurs de f et les enregistre dans tab *)
8   for i=0 to (m/2 - 1) do
9     tab.(i) <- f ((doctor [|bin_of_dix i|] n).(0)); (* f(bin_of_dix i) *)
10  done;
11  for i=0 to (m-1) do (* un peu beaucoup de string_of_int ici, a revoir *)
12    let k = ((doctor [|bin_of_dix i|] (n+1)).(0)) in (* indice i -> etat k ie k = bin_of_dix i *)
13    let j = tab.(dix_of_bin (extract k 0 n)) in (* j = f(n premiers caracteres de k) *)
14    (* n+1 ieme caractere + j mod 2 *)
15    k.[n] <- char_of_string (string_of_int ((int_of_string (string_of_char k.[n]) + (int_of_float j.Re))
16      mod 2));
17    resultat.(i) <- In.(dix_of_bin k); (* colonne de In correspondant a l'etat k modifie *)
18  done;
19  resultat;;

```

```

1 (*deutsch_jozsa : (string -> complex) -> int -> int = <fun> *)
2 let deutsch_jozsa f n =
3   let qubits = make_vect n (genere_qubit_zero()) in
4   qubits.(n-1) <- apply X (genere_qubit_zero());
5   let Htab = make_vect n H in
6   let Uftab = [|Uf_Jozsa f (n-1)|] in
7   let Htab2 = make_vect n H in
8   Htab2.(n-1) <- I;
9   let qubits_mult = mult_tab_qubits qubits in
10  let resultat = apply (matrix_of_circuit [|Htab;Uftab;Htab2|]) qubits_mult in
11  let s = extract (mesure resultat) 0 (n-1) in (* On ne mesure que les (n-1) premiers qubits *)
12  find_ones s;; (* Si on ne mesure que des zeros, alors constante, sinon balancee *)

```

Fonctionnement de `Uf_Jozsa` : Pour calculer la matrice de U_f , on enregistre dans `tab` les différentes valeurs de $f(x)$ ³³, puis on va parcourir tous les états de la base avec `k` et ajouter au dernier qubit $f(n$ premiers qubits), modulo 2. On place dans `resultat` la colonne de l'identité `In` correspondant à l'état modifié.

Fonctionnement de `Deutsch_Jozsa` : On commence par générer $n-1$ qubits dans l'état $|0\rangle$ et un qubit dans l'état $|1\rangle$, qu'on enregistre dans le tableau `qubits`. Ensuite, `Htab` contient n portes de Hadamard, `Uftab` contient la matrice de U_f , et `Htab2` contient $(n-1)$ portes de Hadamard et une porte Identité. On utilise `matrix_of_circuit` pour générer la matrice équivalente au circuit. On applique cette matrice au produit tensoriel des $n-1$ qubits dans l'état $|0\rangle$ et du qubit dans l'état $|1\rangle$ contenus dans le tableau `qubits`³⁴. On mesure ensuite le résultat obtenu, on s'intéresse aux $n - 1$ premiers états : on regarde s'ils sont tous nuls grâce à une fonction auxiliaire `find_ones` qui permet de regarder la présence d'un 1 dans une chaîne de caractère.

32. C'est une matrice de taille $2^{n+1} * 2^{n+1}$

33. Certes, en classique, il existe des algorithmes plus performants qui ne nécessitent pas de calculer toutes les valeurs de $f(x)$. Cependant, ici notre objectif est avant tout de simuler le calculateur quantique - dans un premier temps, l'efficacité est reléguée au second rang.

34. Pur cela on utilise une fonction auxiliaire `mult_tab_qubits` qui réalise le produit tensoriel de tous les qubits contenus dans un tableau.

Annexe D : Preuve de la condition nécessaire et suffisante de non intrication d'un qubit de taille 2^{35}

Enoncé de la condition : Tout vecteur de \mathbb{C}^4 ne se décompose pas en produit tensoriel de deux vecteurs de \mathbb{C}^2 . Une condition nécessaire et suffisante pour que $\begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix}$ puisse s'écrire sous la forme $|\varphi\rangle \otimes |\psi\rangle$ est $\boxed{\alpha_{00}\alpha_{11} = \alpha_{01}\alpha_{10}}$. (*)

Quand elle n'est pas remplie, on dit que les deux qubits sont intriqués.

Preuve de la nécessité de la condition : Supposons que l'on ait effectivement $\begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} = |\varphi\rangle \otimes |\psi\rangle$.

On note $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$ et $|\psi\rangle = \gamma|0\rangle + \delta|1\rangle$.

Alors par produit tensoriel on aura :

$$|\varphi\rangle \otimes |\psi\rangle = \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle$$

$$\text{donc } \alpha_{00}\alpha_{11} = \alpha\gamma * \beta\delta = \alpha\delta * \beta\gamma = \alpha_{01}\alpha_{10}.$$

Preuve de la suffisance de la condition : On procède par analyse et synthèse.

Analyse : On garde les mêmes notations que précédemment et on cherche à exprimer $\alpha, \beta, \gamma, \delta$ en fonction de $\alpha_{00}, \alpha_{01}, \alpha_{10},$ et α_{11} .

On note $\alpha_{ij} = |\alpha_{ij}|e^{i\varphi_{ij}}$ où $i, j \in \{0; 1\}$.

S'il existe i, j tels que $\alpha_{ij} = 0$: On peut supposer que $\alpha_{00} = 0$. Alors d'après (*) soit $\alpha_{01} = 0$, soit $\alpha_{10} = 0$. On peut supposer que $\alpha_{01} = 0$. Il suffit alors de prendre $\alpha = 0, \beta = 1, \gamma = \alpha_{10}$ et $\delta = \alpha_{11}$.

Sinon : alors $\alpha_{ij} \neq 0$ pour tous $i, j \in \{0; 1\}$.

On aura par unicité de la décomposition sur une base :

$$\alpha_{00} = \alpha\gamma, \alpha_{01} = \alpha\delta, \alpha_{10} = \beta\gamma, \alpha_{11} = \beta\delta$$

d'où :

$$\alpha = \frac{\alpha_{00}}{\gamma} = \frac{\alpha_{01}}{\delta}, \beta = \frac{\alpha_{10}}{\gamma} = \frac{\alpha_{11}}{\delta}, \gamma = \frac{\alpha_{00}}{\alpha} = \frac{\alpha_{10}}{\beta}, \delta = \frac{\alpha_{01}}{\alpha} = \frac{\alpha_{11}}{\beta}$$

Comme $|\varphi\rangle$ et $|\psi\rangle$ sont normés :

$$\begin{cases} 1 = |\alpha|^2 + |\beta|^2 = \frac{|\alpha_{00}|^2}{|\gamma|^2} + \frac{|\alpha_{10}|^2}{|\gamma|^2} = \frac{|\alpha_{01}|^2}{|\delta|^2} + \frac{|\alpha_{11}|^2}{|\delta|^2} \\ 1 = |\gamma|^2 + |\delta|^2 = \frac{|\alpha_{00}|^2}{|\alpha|^2} + \frac{|\alpha_{01}|^2}{|\alpha|^2} = \frac{|\alpha_{10}|^2}{|\beta|^2} + \frac{|\alpha_{11}|^2}{|\beta|^2} \end{cases}$$

d'où on tire :

$$\begin{cases} |\alpha| = \sqrt{|\alpha_{01}|^2 + |\alpha_{00}|^2} \\ |\beta| = \sqrt{|\alpha_{10}|^2 + |\alpha_{11}|^2} \\ |\gamma| = \sqrt{|\alpha_{00}|^2 + |\alpha_{10}|^2} \\ |\delta| = \sqrt{|\alpha_{01}|^2 + |\alpha_{11}|^2} \end{cases}$$

Il reste à s'occuper des arguments : on a

$$\begin{cases} \varphi_{00} = \varphi_\alpha + \varphi_\gamma [2\pi] \\ \varphi_{01} = \varphi_\alpha + \varphi_\delta [2\pi] \\ \varphi_{10} = \varphi_\beta + \varphi_\gamma [2\pi] \\ \varphi_{11} = \varphi_\beta + \varphi_\delta [2\pi] \end{cases}$$

qui impliquent

35. Nous avons réalisé et rédigé cette preuve, la bibliographie se contentant de mentionner la propriété.

$$\begin{cases} \varphi_{00} - \varphi_{01} = \varphi_\gamma - \varphi_\delta [2\pi] \\ \varphi_{00} - \varphi_{10} = \varphi_\alpha - \varphi_\beta [2\pi] \end{cases}$$

On peut choisir $\varphi_\beta = 0$ (seule la différence de phase importe)³⁶. On a alors $\varphi_\delta = \varphi_{11}$ et $\varphi_\gamma = \varphi_{10}$, d'où on déduit $\varphi_\alpha = \varphi_{00} - \varphi_{10} = \varphi_{01} - \varphi_{11}$ ³⁷.

Synthèse : On pose :

$$\begin{cases} \alpha = \sqrt{|\alpha_{01}|^2 + |\alpha_{00}|^2} e^{i(\varphi_{00} - \varphi_{10})} \\ \beta = \sqrt{|\alpha_{10}|^2 + |\alpha_{11}|^2} \\ \gamma = \sqrt{|\alpha_{00}|^2 + |\alpha_{10}|^2} e^{i\varphi_{10}} \\ \delta = \sqrt{|\alpha_{01}|^2 + |\alpha_{11}|^2} e^{i\varphi_{11}} \end{cases}$$

On peut vérifier que ces valeurs conviennent. Il existe donc $|\varphi\rangle$ et $|\psi\rangle$ tels que $\begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} = |\varphi\rangle \otimes |\psi\rangle$.

36. On n'a donc pas unicité malgré le raisonnement analyse/synthèse.

37. Cette dernière égalité est bien confirmée par l'hypothèse.

Annexe E : Preuve du théorème de décomposition ZY³⁸

Enoncé du théorème : Soit U une porte unitaire agissant sur un seul qubit. Alors il existe des réels α, β, γ et δ tels que :

$$U = e^{i\alpha} R_z(\beta) R_y(\gamma) R_z(\delta)$$

Notations :

$$U = \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} |a|e^{i\theta} & |b|e^{i\phi} \\ |c|e^{i\varphi} & |d|e^{i\psi} \end{bmatrix}$$

Preuve du théorème : U unitaire $\implies U^\dagger U = U U^\dagger = I$

Or :

$$U^\dagger = \begin{bmatrix} \bar{a} & \bar{c} \\ \bar{b} & \bar{d} \end{bmatrix}$$

Donc, en multipliant U et U^\dagger à droite et à gauche :

$$U U^\dagger = I \Leftrightarrow \begin{bmatrix} |a|^2 + |b|^2 & a\bar{c} + b\bar{d} \\ \bar{a}c + \bar{b}d & |c|^2 + |d|^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$U^\dagger U = I \Leftrightarrow \begin{bmatrix} |a|^2 + |c|^2 & \bar{a}b + \bar{c}d \\ \bar{a}b + \bar{c}d & |b|^2 + |d|^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

D'où les équations :

$$\begin{cases} |a|^2 + |b|^2 = 1 & (1) \\ |c|^2 + |d|^2 = 1 & (2) \\ a\bar{c} + b\bar{d} = 0 & (3) \\ \bar{a}c + \bar{b}d = 0 & (4) \end{cases}$$

et :

$$\begin{cases} |a|^2 + |c|^2 = 1 & (5) \\ |b|^2 + |d|^2 = 1 & (6) \\ \bar{a}b + \bar{c}d = 0 & (7) \\ \bar{a}b + \bar{c}d = 0 & (8) \end{cases}$$

Commençons par faire vivre γ (n'oublions pas que $|c| \geq 0$ et $|d| \geq 0$) :

$$(1) \implies \exists \gamma, |a| = \cos \frac{\gamma}{2} \text{ et } |b| = \sin \frac{\gamma}{2}$$

$$(5) \text{ et } \exists \gamma, |a| = \cos \frac{\gamma}{2} \implies |c| = \sin \frac{\gamma}{2}$$

$$(6) \text{ et } \exists \gamma, |b| = \sin \frac{\gamma}{2} \implies |d| = \cos \frac{\gamma}{2}$$

Maintenant, faisons vivre α, β et δ : remarquons déjà que (3) = $\overline{(4)}$ et (7) = $\overline{(8)}$. Dans toute la suite, on se place modulo 2π .

$$(3) \implies |a| \times |c| e^{i(\theta-\varphi)} + |b| \times |d| e^{i(\phi-\psi)} = 0$$

$$(7) \implies |a| \times |b| e^{i(\theta-\phi)} + |c| \times |d| e^{i(\varphi-\psi)} = 0$$

38. Nous avons réalisé et rédigé cette preuve, la bibliographie estimant qu'il s'agissait d'une trivialité.

Cas $\cos \frac{\gamma}{2} \times \sin \frac{\gamma}{2} \neq 0$: Comme $|a| \times |c| = |b| \times |d| = |a| \times |b| = |c| \times |d| = \cos \frac{\gamma}{2} \times \sin \frac{\gamma}{2}$, il vient :

$$(3) \implies e^{i(\theta-\varphi)} + e^{i(\phi-\psi)} = 0 \implies \theta - \varphi = \phi - \psi + \pi$$

$$(7) \implies e^{i(\theta-\phi)} + e^{i(\varphi-\psi)} = 0 \implies \theta - \phi + \pi = \varphi - \psi$$

Posons $-\beta = \theta - \varphi$, $-\delta = \theta - \phi + \pi$ et $\theta + \psi = \varphi + \phi + \pi = 2\alpha$ (cette dernière équation s'obtient avec celle qui définit β).

On a le système :

$$\begin{cases} \theta - \varphi = \phi - \psi + \pi = -\beta & (9) \\ \theta - \phi + \pi = \varphi - \psi = -\delta & (10) \\ \theta + \psi = \varphi + \phi + \pi = 2\alpha & (11) \end{cases}$$

D'où on tire facilement :

$$\begin{cases} \theta = \varphi - \beta = \phi - \delta - \pi = 2\alpha - \psi & (12) \\ \phi + \pi = \psi - \beta & (13) \\ \varphi = \psi - \delta & (14) \\ \varphi + \pi = 2\alpha - \phi & (15) \end{cases}$$

Il reste à exprimer les arguments θ , etc en fonction de α , β , etc.

On commence par θ en prenant (9) et (12) + (14) :

$$\begin{cases} \theta - \varphi = -\beta & (16) \\ \theta + \varphi = 2\alpha - \psi + \psi - \delta & (17) \end{cases}$$

\Leftrightarrow

$$\begin{cases} \theta - \varphi = -\beta & (18) \\ \theta + \varphi = 2\alpha - \delta & (19) \end{cases}$$

On fait (18) + (19) : $2\theta = 2\alpha - \delta - \beta \implies \theta = \alpha - \frac{\beta}{2} - \frac{\delta}{2}$

On déduit les autres angles de (12) :

$$\theta = \varphi - \beta \implies \varphi = \alpha + \frac{\beta}{2} - \frac{\delta}{2}$$

$$\theta = \phi - \delta - \pi \implies \phi = \alpha - \frac{\beta}{2} + \frac{\delta}{2} + \pi$$

$$\theta = 2\alpha - \psi \implies \psi = \alpha + \frac{\beta}{2} + \frac{\delta}{2}$$

d'où le résultat.

Cas $\cos \frac{\gamma}{2} \times \sin \frac{\gamma}{2} = 0$: Alors $\cos \frac{\gamma}{2} = 0$ ou $\sin \frac{\gamma}{2} = 0$.

Cas $\cos \frac{\gamma}{2} = 0$: Alors $\gamma = \pi \pmod{2\pi}$ et $a = d = 0$, $|b| = |c| = 1$. Il suffit de poser $2\alpha = \varphi + \phi + \pi$ et $\beta - \delta = \varphi - \phi + \pi$. Seule la valeur de $\beta - \delta$ est fixée, donc β et δ ne sont pas fixés. Par exemple, $\beta = \varphi - \phi + \pi$ et $\delta = 0$ convient.

Cas $\sin \frac{\gamma}{2} = 0$: Alors $\gamma = 0 \pmod{2\pi}$. Les rôles sont inversés, on trouve de même des valeurs pour α , β et δ qui conviennent, ie $2\alpha = \theta + \psi$ et $\theta - \psi = -\beta - \delta$. Par exemple, $\beta = \psi - \theta$ et $\delta = 0$ conviennent.

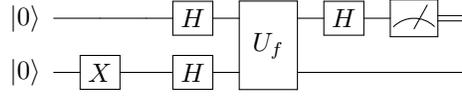
Corollaire : Soit U une porte unitaire agissant sur un seul qubit. Alors il existe des opérateurs unitaires A , B et C (agissant sur un seul qubit) tels que $ABC = I$ et $U = e^{i\alpha}AXBXC$, où α est un facteur de phase global.

Preuve du corollaire³⁹ : il suffit de prendre $A = R_z(\beta)R_y(\gamma/2)$, $B = R_y(-\gamma/2)R_z(-(\delta + \beta)/2)$ et $C = R_z((\delta - \beta)/2)$.

39. Pour plus de détails, voir [4], p.175-176.

Annexe F : Preuve de la correction de l'algorithme de Deutsch⁴⁰

Rappel de l'algorithme de Deutsch : L'algorithme de Deutsch permet de déterminer si une fonction $f : \{0;1\} \rightarrow \{0;1\}$ est constante ou équilibrée en opérant une seule évaluation de f . Pour cela il utilise un opérateur $U_f : |x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle$ où le \oplus représente l'addition modulo 2. Le circuit quantique représentant l'algorithme est rappelé ci-dessous :



Rappel de l'action de la porte de Hadamard :

$$H : \begin{cases} |0\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \\ |1\rangle \mapsto \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \end{cases}$$

Preuve de la correction de l'algorithme :

Etape 1 : $(I \otimes X)(|0\rangle \otimes |0\rangle) = |0\rangle \otimes |1\rangle$

Etape 2 : $(H \otimes H)(|0\rangle \otimes |1\rangle) = \left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$

Etape 3 : $U_f : (|x\rangle, |y\rangle) \mapsto (|x\rangle, |y \oplus f(x)\rangle)$

Lemme : $U_f |x\rangle \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) = |x\rangle \left(\frac{|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}}\right) = (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$

Ici, on aura :

$$\begin{aligned} U_f \left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) &= U_f \frac{|0\rangle}{\sqrt{2}} \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) + U_f \frac{|1\rangle}{\sqrt{2}} \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) \\ &= (-1)^{f(0)} \frac{|0\rangle}{\sqrt{2}} \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) + (-1)^{f(1)} \frac{|1\rangle}{\sqrt{2}} \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) \\ &= \begin{cases} \pm \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{si } f(0) = f(1) \\ \pm \left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{si } f(0) \neq f(1) \end{cases} \end{aligned}$$

Etape 4 : On applique H au premier qubit :

– Si $f(0) = f(1)$: $(H \otimes I) \left(\pm \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right) = \pm |1\rangle \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$

– Si $f(0) \neq f(1)$: $(H \otimes I) \left(\pm \left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)\right) = \pm |0\rangle \left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right)$

Etape 5 : La mesure du premier qubit donne ensuite le résultat.

40. Pour le détail des calculs, on se référera avec profit à [4], p.33 ou [3], p.48.

Annexe G : Bibliographie

- [1] S. Bettelli, Luciano Serafini, and T. Calarco. Toward an architecture for quantum programming. *CoRR*, cs.PL/0103009, 2001.
- [2] Michel Le Bellac. *Introduction à l'information quantique*. Belin, 2005.
- [3] David Mermin. *Calculs et algorithmes quantiques, méthodes et exemples*. CNRS Editions, 2010.
- [4] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.
- [5] Krysta M. Svore, Alfred V. Aho, Andrew W. Cross, Isaac Chuang, and Igor L. Markov. A layered software architecture for quantum computing design tools. *Computer*, 39(1) :74–83, January 2006.