Qbricks, un environnement pour la vérification formelle en informatique quantique *

Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perelle, Benoît Valiron

Qbricks, un environnement pour la vérification formelle en informatique quantique*

Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perelle
CEA, LIST, Université Paris-Saclay, France
firstname.lastname@cea.fr

Benoît Valiron
LRI, CentraleSupélec, Université Paris-Saclay, Orsay, France
benoit.valiron@lri.fr

1 Introduction

L'informatique quantique est un domaine actif de recherche, avec comme enjeu un possible bouleversement des capacités de calculs dans certains domaines clés (optimisation, cryptographie, simulation physique, etc). Alors que les progrès sont sensibles côté matériel, il est temps de s'intéresser aux chaînes de développement logiciel qui accompagneront ces machines. Un premier pas dans cette direction est le développement de langages de programmation quantique [1, 7, 11]. Les utilisateurs et développeurs de ce type de langages ont la tâche difficile de créer des langages et des programmes qu'ils ne peuvent pas tester ou déboguer facilement. En effet, de par la nature du calcul quantique, toute observation d'un registre au cours d'une exécution est probabiliste, et par ailleurs elle détruit inévitablement l'état du registre qu'elle observe. Il n'est donc pas possible de tester un programme par des exécutions partielles ou des tests de valeurs sur des variables en cours d'exécution. Enfin, l'informatique quantique est par nature destinée à implémenter des algorithmes à des échelles non simulables classiquement.

Assurer la correction des programmes quantiques est donc un défi d'importance cruciale.

État de l'art. Différentes approches ont été proposées pour répondre à ce défi. Une première approche consiste à étendre des techniques de *model-checking* au domaine quantique [6, 15]. Cette approche utilise des spécifications non paramétrées, elle est donc limitée dans son utilisation par la taille des registres quantiques. Une seconde approche est le langage Qwire [11, 12], un langage fonctionnel embarqué dans l'assistant de preuves Coq. Qwire bénéficie du puissant système de types dépendants de Coq. Un premier problème pour cette approche est le manque d'automaticité : dans Coq, les preuves sont vérifiées automatiquement, mais elles doivent être

*Les travaux présentés dans cet article ont été financés par l'Agence Nationale de la Recherche (ANR), projet SoftQPro, ANR-17-CE25-0009

intégralement écrites par l'utilisateur. La deuxième difficulté vient de la combinaison des types et des spécifications : les annotations de types sont complexes et les preuves doivent être données pour permettre au programme de compiler. Une dernière approche réside dans l'extension de la logique de Hoare (QHL) à la programmation quantique [4, 14]. QHL [14] interprète les opérateurs de densité comme des *prédicats quantiques* qui sont des atomes pour une logique de type Hoare. Des efforts sont actuellement en cours pour la génération automatique d'invariants dans QHL [16] et la preuve de théorèmes pour QHL en Isabelle/HOL [9]. Le défaut de l'approche QHL réside dans son formalisme de spécification, très restrictif puisque les prédicats exprimables y sont limités aux opérateurs positifs.

Objectifs. Notre but est de proposer une approche de vérification formelle des programmes quantiques répondant aux différentes limitations perçues dans les approches existantes. Plus spécifiquement nous voulons :

— des spécifications intuitives, séparant clairement l'écriture d'un programme et sa spécification,

— des outils pour automatiser autant que possible l'écriture de ces spécifications.

Approche et premiers résultats. Nous développons *Qbricks*, un environnement purement fonctionnel de programmation et de vérification. La vérification étant paramétrée par la taille des registres de données, cette approche a le même coût quelle que soit la taille de ces registres et ne pose donc pas de problème de passage à l'échelle. *Qbricks* est un langage dédié embarqué dans un outil robuste et éprouvé dans différents contextes industriels : Why3 [3, 5], conçu pour prouver des propriétés de programmes classiques. Nous présentons ici de manière générale le modèle de calcul quantique standard (Section 2) et notre méthodologie (Section 3). Pour *Qbricks*, nous avons implémenté la sémantique standard, matricielle, ainsi qu'une sémantique alternative proposée en 2018 et qui repose sur le formalisme des sommes de chemins [2]. Nous avons comparé l'usage de ces deux sémantiques et prouvé formellement leur équivalence (Section 4). À titre de preuve de concept, nous avons par ailleurs développé et prouvé la correction dans chacune de ces deux sémantiques d'une des primitives majeures du calcul quantique : la transformée de Fourier quantique (Section 5).

2 Informatique quantique et modèle Qram hybride

Information quantique. Là où un bit d'un ordinateur classique est toujours dans un état parmi deux possibles (typiquement noté **0** et **1**), son équivalent quantique, un *qubit* peut être préparé et manipulé dans un état qui est une superposition de ces deux états. Plus précisément, la convention d'écriture désignant respectivement les états **0** et **1** par $|0\rangle$ et $|1\rangle$, l'état d'un qubit est représenté par la forme

$$qb_1 := a_0 \, |0\rangle + a_1 \, |1\rangle \tag{1}$$

où a_0 et a_1 sont des valeurs complexes respectant la relation $|a_0|^2 + |a_1|^2 = 1$. Cette particularité révèle son intérêt pour des registres de données comportant plusieurs qubits, à travers le phénomène de la *superposition d'états* : alors que l'état d'un registre classique à n bits correspond à

un élément dans l'ensemble de taille 2^n de toutes les chaînes de n bits, un registre quantique à n qubits correspond à une *combinaison linéaire* (à nombres complexes) de chaînes de n bits : on peut donc avoir les 2^n chaînes de n bits en superposition en même temps. L'état d'un registre quantique s'écrit donc, en étendant la notation employée ci-dessus à des registres $|k\rangle_n$ encodant des entiers k en n qubits, sous la forme suivante :

$$\sum_{k=0}^{2^n-1} a_k \, |k\rangle_n$$

Il en résulte un accroissement potentiellement exponentiel de la quantité d'information stockable dans un registre quantique et manipulable par un ordinateur quantique. Les lois de la physique quantique, auxquelles est soumis le support d'information pour un ordinateur quantique, contraignent cependant sévèrement le calcul. Notamment :

— les opérations possibles sont restreintes à une certaine classe, qui correspond algébriquement aux opérateurs unitaires ;

— l'information contenue dans un registre quantique n'est pas accessible directement, mais via une opération de mesure, qui transforme un registre quantique (ou une partie d'un registre quantique) superposé en registre classique, en le projetant de manière probabiliste sur un des termes de sa superposition. La quantité d'information renvoyée est donc linéaire par rapport à la taille du registre.

C'est cette seconde contrainte qui rend impossible le débogage du code au sens classique. Il n'est pas possible d'arrêter une exécution en cours pour l'analyser, sauf à détruire la superposition d'états du registre. Il faut donc générer intégralement les séquences de calcul quantique avant de les lancer pour des exécutions d'un seul tenant.

Circuits quantiques. Les opérations possibles sur un registre quantique sont approximées, à partir d'un ensemble fini donné de portes élémentaires, par deux compositions :

— la séquence, qui correspond à la succession des opérations sur un registre,

— la composition parallèle, qui permet d'agir sur différentes parties d'un registre.

On forme ainsi un *circuit quantique*, qui correspond à une instruction structurée de calcul pour un ordinateur quantique. Un exemple en est donné par la Figure 1. Il s'agit du circuit pour la Transformée de Fourier Quantique (QFT), dont nous reparlerons dans la Section 4. L'instance de la Figure 1 agit sur un registre à six qubits, représentés par les lignes horizontales indicées. Nous avons par ailleurs souligné, par des tracés pointillés, la structure récursive de cette construction. Les sous-circuits imbriqués correspondent respectivement aux instances de la transformée de Fourier quantique pour des registres à 5,4,3,2 puis 1 qbits. Deux opérations agissent sur ces qubits :

— une opération unaire, la transformation de Hadamard (H), qui opère une superposition d'états sur un qubit,

— une opération binaire, la rotation contrôlée (CR_k), dont le paramètre k indique l'angle de rotation $\frac{2\pi}{2^k}$. Par exemple, la porte dessinée en traits épais indique une rotation d'angle $\frac{\pi}{2}$ du qubit 2 si le qubit 5 est dans l'état $|1\rangle$.

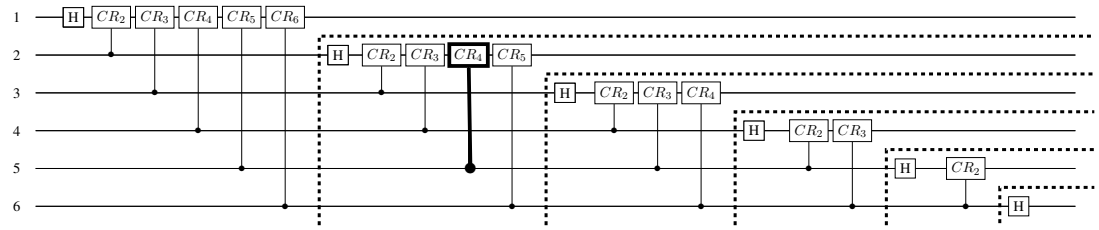

FIGURE 1: Circuit pour la transformée de Fourier quantique

Modèle de calcul hybride (co-processeur quantique). Dans le modèle standard d'architecture quantique, le modèle *Qram* [8], des instructions de calcul sont envoyées sous forme de circuit quantique depuis un ordinateur classique vers un co-processeur quantique. Ce co-processeur effectue le calcul encodé par ce circuit et renvoie les résultats à l'ordinateur classique qui les traite. L'ensemble du contrôle de flot est donc géré par l'ordinateur classique. L'interaction homme-machine s'effectue également entièrement avec l'ordinateur classique.

Dans cet échange d'informations et ce processus de calcul, maîtriser précisément la relation entre un circuit quantique envoyé au co-processeur quantique et la fonction liant les sorties de ce circuit à ses entrées est donc d'une importance cruciale. C'est à ce besoin que nous souhaitons répondre par le développement du formalisme Qbricks.

3 Méthodologie

Notre idée maîtresse est de développer un langage minimal (*Qbricks*) de construction de circuits quantiques avec une sémantique formelle qui interprète ces circuits comme des fonctions transformant des registres quantiques. Cette sémantique étant définie récursivement sur la structure des circuits, on peut calculer, pour chaque circuit, son interprétation sémantique. À l'aide de macros adéquates, on construit ainsi, dans une syntaxe minimale de construction de circuits, des circuits tels que celui de la Figure 1. Ces constructions peuvent éventuellement être paramétrées, de manière à correspondre à des *familles de circuits*.

Sémantique duale. Parallèlement nous développons des outils d'interprétation sémantique pour *Qbricks*. Pour chacun des termes du langage, il s'agit d'interpréter formellement les calculs qu'il permet. Pour *Qbricks* nous développons parallèlement deux sémantiques formelles distinctes : la sémantique matricielle, qui est la sémantique standard, et la sémantique des sommes de chemins. Nous y reviendrons dans la Section 4.

Vérification. En vérification déductive de programmes, les programmes sont décorés par des assertions telles que des pré- ou postconditions ou des invariants de boucles. Pour l'utilisateur, ces décorations constituent des contrats assurant le respect des postconditions par les *outputs* du programme, pour tout *input* qui en vérifie les préconditions. Inférer la satisfaction des postcon-

ditions en sortie de la satisfaction des préconditions en entrée constitue alors une obligation de preuve pour le développeur.

C'est cette méthodologie que nous adoptons dans le développement de *Qbricks*, en annotant les programmes de construction de circuits par des spécifications sur la sémantique formelle. Les annotations constituent donc des contrats de correction du calcul effectué par le circuit créé par le programme.

Why3. L'outil Why3 [3] est un environnement de spécification qui fournit un langage de type ML à la fois pour la programmation et pour l'écriture des spécifications. À partir d'un programme spécifié, Why3 génère un ensemble d'obligations de preuves qui doivent être remplies afin de certifier ce programme. Ces preuves peuvent être manipulées à travers une interface graphique dédiée. Why3 fournit aussi un assistant de preuves interactif. Pour prouver un théorème, on peut ainsi appeler un ensemble de prouveurs SMT automatiques (CVC, Z3, Alt-ergo,etc), accessibles depuis l'interface. Il est aussi possible d'appeler des lemmes pour fournir aux prouveurs des étapes de la preuve, ainsi que de directement simplifier les objectifs de preuve par différentes commandes de transformation de termes.

4 Sémantique matricielle et sommes de chemins

La sémantique standard interprète les circuits quantiques en termes de matrices. On donne ci-dessous cette sémantique pour les opérations de base :

$$M_{Had} := \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \qquad M_{Id} := \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \qquad M_{CRk} := \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & e^{\frac{2i\pi}{2^k}}\end{pmatrix} \qquad M_{Swap} := \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

La composition séquentielle est alors interprétée par le produit matriciel, et la composition parallèle par le produit tensoriel de matrices. On représente de plus un registre quantique par un vecteur – $\begin{pmatrix}1\\0\end{pmatrix}$ pour le registre $|0\rangle$ et $\begin{pmatrix}0\\1\end{pmatrix}$ pour le registre $|1\rangle$ par exemple –. Un calcul quantique effectué par un circuit C de sémantique matricielle M_c sur un registre V est alors interprété par la multiplication matricielle $M_c \cdot V$. Ci-dessous à titre d'illustration, nous représentons l'action d'un circuit élémentaire constitué de la seule porte de Hadamard sur le qubit qb_1 de l'équation 1 :

$$M_{Had}\cdot qb_1 = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\cdot\left(a_0\begin{pmatrix}1\\0\end{pmatrix}+a_1\begin{pmatrix}0\\1\end{pmatrix}\right) = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}\cdot\begin{pmatrix}a_0\\a_1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}a_0+a_1\\a_0-a_1\end{pmatrix}$$

Nous avons implémenté la sémantique matricielle pour *Qbricks* en Why3. Travailler avec cette sémantique nous a cependant conduit à chercher une expression sémantique plus abstraite et donc plus adéquate pour les preuves automatiques. La manipulation des matrices présente en effet plusieurs inconvénients, parmi lesquels :

— les matrices sont utilisées dans ce domaine pour représenter, via la multiplication matricielle à un vecteur représentant le registre d'entrée du calcul, des fonctions transformant

des registres quantiques. On leur préférera une expression directe des fonctions des vecteurs vers les vecteurs ;

— les matrices sont une représentation pour des fonctions bornées, de $(\mathbb{N} \times \mathbb{N})$ dans $\mathbb{C}$. La gestion des indices peut s'avérer fastidieuse et parasiter le développement. On préférera donc manipuler des objets moins contraints ;

— enfin, comme l'illustrent à leur niveau les matrices M_{CR_k} et M_{Swap} présentées ci-dessus, les matrices utilisées dans le calcul quantique peuvent être très creuses. On leur préférera des structures de données plus compactes et qui ne tiennent pas compte des zéros.

Récemment, Matthew Amy a proposé une interprétation sémantique du calcul quantique qui remplit ces objectifs [2]. Les circuits y sont directement représentés comme des fonctions de transformation des registres quantiques. La multiplication d'une matrice par un vecteur y est remplacée par une décomposition en somme de produits de vecteurs. On représente ainsi les circuits comme des sommes de vecteurs, appelées des sommes de chemins. On en donne ici l'expression générale pour un registre $|j\rangle_n$ de taille n encodant un entier j :

$$\textit{Path-sum}\ (\ |j\rangle_n) = \frac{1}{2^r} \sum_{l=0}^{2^r-1} e^{\frac{2\pi i(p(j,l))}{c}}\ |k(j,l)\rangle_n$$

où r et c sont des constantes entières et p et k sont des fonctions de $(\mathbb{N} \times \mathbb{N})$ dans $\mathbb{N}$. Il ne s'agit bien sûr que de suggérer une intuition du gain apporté. Il apparaît en effet clairement que cette expression prend en charge une fois pour toutes des éléments structurels communs à tous les circuits quantiques. Par ailleurs cette expression ne contient qu'un unique opérateur de somme, là où l'interprétation par les matrices ajoute une somme imbriquée pour chaque multiplication (donc, selon la sémantique, pour chaque transition d'une séquence). Enfin, les constantes r et c et les fonctions p et k ont des expressions simples et peuvent être chacune définie récursivement sur la structure du circuit concerné.

Nous avons donc implémenté également cette sémantique pour *Qbricks*.

5 Cas d'étude : la transformée de Fourier

En l'état actuel de la recherche, les algorithmes quantiques reposent sur un ensemble de *routines* de calcul. L'une des principales est la *transformée de Fourier quantique*. Cette opération est réalisée par le circuit montré dans la Figure 1. Elle permet de transformer un registre superposé $|x\rangle_n = \sum_{k=0}^{2^n-1} x_k\ |k\rangle_n$ en le registre

$$\textit{QFT}\ (\ |x\rangle_n) = \frac{1}{2^n} \sum_{k=0}^{2^n-1} \sum_{j=0}^{2^n-1} x_j e^{\frac{2\pi i * jk}{2^n}}\ |k\rangle_n \tag{2}$$

en un nombre polynomial d'étapes de calcul ($\frac{n(n+1)}{2}$ étapes pour le circuit de la Figure 1). Cette transformation (voir [10] pour une présentation détaillée) est au cœur de nombreux algorithmes quantiques emblématiques, tels que l'algorithme de Shor pour la factorisation de nombres premiers [13] ou l'estimation de phase.

La Figure 1 illustre le caractère récursif de ce circuit. Il est possible de l'implémenter de manière paramétrée, de façon à définir une famille de circuits indicés par la taille du registre quantique. C'est ce type d'invariants d'échelle que nous exploitons à travers *Qbricks*, afin de fournir des méthodes de développement certifiées quelle que soit la taille d'une instance de calcul.

Résultats obtenus. Afin d'éprouver notre méthodologie, nous avons implémenté cette transformation dans *Qbricks*. Nous avons ensuite prouvé sa correction quant à l'expression de l'équation 2, séparément pour la sémantique matricielle et pour la sémantique des sommes de chemins. Cette construction et ces preuves sont paramétrées. Elles sont donc insensibles à l'échelle.

À notre connaissance, il s'agit de la première proposition pour une preuve formelle de correction d'un calcul quantique écrite dans un langage général de génération de circuits quantiques.

Dans l'état actuel, l'ensemble contient environ 10 000 lignes de codes, principalement pour les bibliothèques relatives à la sémantique du calcul quantique (opérations matricielles de sommes, produit tensoriel, etc., opérations binaires, opérations sur les nombres complexes, etc.). Il contient environ 300 définitions et 1000 preuves de lemmes intermédiaires et théorèmes.

6 Conclusion

Nous développons *Qbricks*, un langage noyau pour la programmation et la vérification de programmes quantiques. Nous nous appuyons pour cela sur l'outil Why3, qui permet à la fois l'écriture de programmes dans un langage fonctionnel de type ML, l'écriture d'annotations pour ces programmes et des techniques de preuves formelles du respect de ces annotations. Nous avons implémenté ce langage ainsi que l'ensemble des librairies nécessaires pour sa sémantique (algèbre, théorie des ensembles, opérations binaires,etc). Nous avons implémenté deux sémantiques pour *Qbricks* : d'une part, la sémantique matricielle, standard pour l'interprétation des programmes quantiques et d'autre part, la sémantique des sommes de chemins, proposition récente, avantageuse pour la manipulation des structures de données et la preuve formelle. Nous avons prouvé la correction du calcul pour une routine fondamentale du calcul quantique : la transformée de Fourier quantique. La preuve de correction de la transformée de Fourier quantique a été développée indépendamment pour chacune des deux sémantiques de *Qbricks*. Nous avons par ailleurs prouvé intégralement en Why3 l'équivalence entre ces deux sémantiques. Nous disposons ainsi à la fois d'un cadre formel adapté à la vérification déductive (la sémantique des sommes de chemins), et d'une méthode de traduction des résultats obtenus dans la sémantique standard pour l'informatique quantique (la sémantique matricielle).

Dans un futur proche, nous prévoyons d'étendre notre travail sur la transformée de Fourier à l'algorithme d'estimation de phases. Nous disposerons alors d'un algorithme quantique entièrement vérifié.

Remerciements. Nous remercions Chantal Keller et Dongho Lee pour la relecture attentive de cet article et les discussions constructives durant sa rédaction.

Références

[1] The Q# programming language. https://docs.microsoft.com/en-us/quantum/language/?view=qsharp-preview, 2017.

[2] Matthew Amy. Towards large-scale functional verification of universal quantum circuits. *CoRR*, 2018.

[3] François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. Why3 : Shepherd Your Herd of Provers. In *Boogie 2011 : First International Workshop on Intermediate Verification Languages*, pages 53–64, Wroclaw, Poland, 2011.

[4] Yuan Feng, Runyao Duan, Zheng-Feng Ji, and Mingsheng Ying. Proof rules for the correctness of quantum programs. *Theor. Comput. Sci.*, 386(1-2) :151–166, 2007.

[5] Jean-Christophe Filliâtre and Andrei Paskevich. Why3 - where programs meet provers. In *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013*, pages 125–128, 2013.

[6] Simon J. Gay, Rajagopal Nagarajan, and Nikolaos Papanikolaou. QMC : A model checker for quantum systems. In *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, pages 543–547, 2008.

[7] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper : a scalable quantum programming language. In *PLDI '13, Seattle, WA, USA, June 16-19, 2013*, pages 333–342, 2013.

[8] E. Knill. Conventions for quantum pseudocode. Los Alamos National Laboratory. Technical report, 1996.

[9] Tao Liu, Yangjia Li, Shuling Wang, Mingsheng Ying, and Naijun Zhan. A theorem prover for quantum hoare logic and its applications. *arXiv :1601.03835*, 2016.

[10] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information : 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011.

[11] Jennifer Paykin, Robert Rand, and Steve Zdancewic. QWIRE : a core language for quantum circuits. *ACM SIGPLAN Notices*, 2017.

[12] Robert Rand, Jennifer Paykin, and Steve Zdancewic. QWIRE practice : Formal verification of quantum circuits in coq. *arXiv :1803.00699*, 2018.

[13] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5), 1997.

[14] Mingsheng Ying. Floyd–hoare logic for quantum programs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 33(6) :19, 2011.

[15] Mingsheng Ying, Yangjia Li, Nengkun Yu, and Yuan Feng. Model-checking linear-time properties of quantum systems. *ACM Trans. Comput. Log.*, 15(3) :22 :1–22 :31, 2014.

[16] Mingsheng Ying, Shenggang Ying, and Xiaodi Wu. Invariants of quantum programs : characterisations and generation. *ACM SIGPLAN Notices*, 2017.