

---

# Around Quantum Decision Diagrams

Report – Research Track – Semester 7

*Student : Malo LEROY*  
*Supervisor : Renaud VILMART*

---

## Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                     | <b>1</b>  |
| 1.1      | Context and objective . . . . .         | 1         |
| 1.2      | State of the art . . . . .              | 1         |
| 1.3      | Contribution . . . . .                  | 3         |
| 1.4      | Structure of the report . . . . .       | 3         |
| <b>2</b> | <b>Modèle théorique</b>                 | <b>4</b>  |
| 2.1      | Arithmétique des intervalles . . . . .  | 4         |
| 2.2      | États abstraits et diagrammes . . . . . | 5         |
| 2.3      | Approximations . . . . .                | 5         |
| 2.4      | Réduction . . . . .                     | 5         |
| 2.5      | Exemple d'approximation . . . . .       | 6         |
| 2.6      | Erreur . . . . .                        | 6         |
| 2.7      | Application de portes . . . . .         | 7         |
| <b>3</b> | <b>Implémentation</b>                   | <b>8</b>  |
| 3.1      | Structure du code . . . . .             | 8         |
| 3.2      | Tests . . . . .                         | 8         |
| 3.3      | Interpréteur QASM . . . . .             | 8         |
| 3.4      | Outils . . . . .                        | 9         |
| <b>4</b> | <b>Conclusion</b>                       | <b>10</b> |
| 4.1      | Travail réalisé . . . . .               | 10        |
| 4.2      | Perspectives . . . . .                  | 10        |

# Chapter 1

## Introduction

### 1.1 Context and objective

**Quantum computing** is a rapidly expanding field. This technology, which enables qubits to be manipulated instead of the bits that form the basis of today's computing, paves the way for algorithms that are more powerful than conventional ones. As the quantum machines running these algorithms are still under development and costly, there is a need for tools to simulate and verify quantum algorithms using classical machines. The aim of this project is to propose a model of the data structure of abstract additive quantum decision diagrams, based on existing work, and to simulate them in order to study their performance.

### 1.2 State of the art

#### Quantum computing

The first postulate of quantum physics, the **principle of superposition**, states that the state space of a quantum system is a Hilbert space. Consequently, while a *bit* can classically only be in a  $|0\rangle$  or  $|1\rangle$  state, its quantum counterpart, the **qubit**, can be in a superposition of these two states.

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

where  $\alpha, \beta$  are complex coefficients. The second postulate, the *measurement principle*, states that when a qubit is measured, it is projected onto one of the base states  $|0\rangle$  or  $|1\rangle$  with probability  $|\alpha|^2$  or  $|\beta|^2$  respectively.

States with  $n$  qubits can be represented by  $2^n$ -dimensional **vectors**, the states of the set consisting of two systems being those obtained by tensor product (*Kronecker product*) of a state of the first system and a state of the second. It is this exponential number of complex parameters for a multi-qubit state that makes the classical study of quantum algorithms difficult, since the states take up exponentially large amounts of memory.

As in classical computing, elementary operations on memory are performed in quantum computing by **gates**. From a mathematical point of view, a gate operating on  $n$  qubits is commonly represented by a matrix of size  $2^n \times 2^n$ . Applying a gate  $M$  to a set of qubits  $v$  then amounts to multiplying the state vector by the gate matrix.

Parallel application of multiple gates to multiple qubits is represented by the **tensor product** (Kronecker product) of the gate matrices. It should be noted, however, that the application of gates to qubits in this way takes exponential time as a function of the number of qubits, making the naive use of quantum algorithms on classical machines inefficient.

The application of gates to qubits is frequently represented as **quantum circuits**, where qubits are represented by lines and gates by boxes.

Any *unitary* and *reversible* matrix can be used as a quantum gate. Among the most common gates are the Hadamard gate ( $H$ ), the “not” gate ( $X$ ), the “controlled not” gate ( $CX$ ), or the swap gate ( $S$ ). These and other gates are used in quantum algorithms such as Deutsch-Josza's [1], Grover's [2] and Shor's [3].

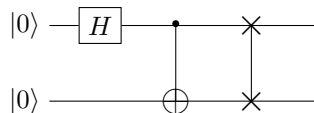


Figure 1.1: 2-qubits circuit with gates  $H$ ,  $CX$  and  $S$

To provide a unified language for developing quantum algorithms from gates that can be interpreted on any hardware, IBM released the **Open QASM** programming language in 2017. [4] It defines qubits on which gates are applied, and the program can then be simulated or executed on a quantum computer as on a conventional computer. The preceding circuit can, for example, be represented by the program 1.2:

```
qubit a;
qubit b;
h a;
cx a b;
s a b;
```

Program 1.2: Example Open QASM code

## Decision diagrams

The **decision diagram** structure is a data structure developed in the 1970s. It has since become widely used in computer science, in particular to make the representation of binary functions more compact.

Take, for example, the binary function  $f(x_1, x_2, x_3) = x_1 \vee (x_2 \wedge x_3)$ . In the general case, such a function is represented using a *truth table*, i.e. in a size that grows exponentially with the number of binary variables considered. A more compact representation can be achieved using a decision diagram, as shown in Figure 1.3, where left-hand children are indicated by dotted arrows and right-hand children by solid arrows.

Decision diagrams take advantage of the data's internal **structure** (here, a Boolean function). On the one hand, labels are not needed to reconstruct the values taken by the function. On the other hand, in the worst case, i.e. where the function has no structure allowing reduction, the size of the decision diagram (its number of branches) is  $2^{n+1} - 2$  which, like the number of values  $2^n$  to be stored in a truth table, is **exponential** in  $n$ . In the worst case, decision diagrams offer no improvement, but are not asymptotically worse than truth tables either.

## Abstract interpretation

Abstract interpretation is a general method of dealing with the properties of computer programs by abstracting them. It was introduced by Patrick and Radhia Cousot in 1977 [5]. It can also be used to solve problems or compute faster.

Consider the following problem: we're trying to determine the sign of the expression  $-12 \times 7 - 13$ . We could calculate the result of this expression, but we could also note that  $-12$  and  $-13$  are negative and that  $7$  is positive, hence  $-12 \times 7$  is negative, so the expression is negative. More formally, we've taken the concrete elements  $-12$ ,  $7$  and  $-13$  and replaced them with the abstract elements  $\oplus$  "positive" and  $\ominus$  "negative", on which we define sum and product operations according to well-known rules.

There are many uses for the abstract interpretation [6], for example in program compilation. In this project, we'll be using it to reduce quantum decision diagrams, even if this means losing some of the information they contain. This is also the case for the previous example: abstracting elements by their sign does not always allow us to determine the sign of the expression.

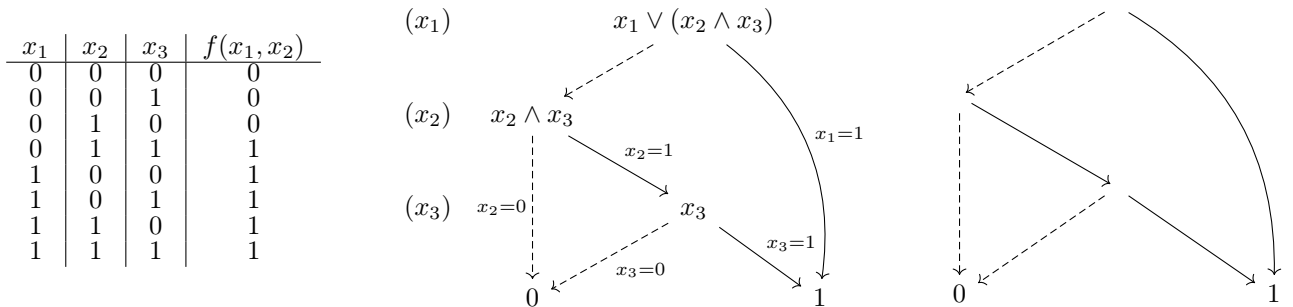


Figure 1.3: (a) Table de vérité (b) Diagramme de décision (c) Diagramme de décision sans labels

## 1.3 Contribution

The various concepts presented in the state of the art can all be used to improve the computation speed or memory size of a dataset. The aim of this project was therefore to combine these concepts, adding an extra dimension, additivity: the fact that a diagram has several right (respectively left) wires, the interpretation being that the “effective right wire” is the sum of the diagram’s right (respectively left) wires. The aim of the project was therefore to propose a model of abstract, additive quantum decision diagrams, and to simulate them in order to study their performance in comparison with other models.

As a preliminary step, a study of polar complex intervals was carried out, which had not been done in the literature until now. A mathematical **model** of the data structure and reduction methods were formalized, and reduction algorithms were developed to reduce these diagrams. In addition, methods for applying quantum gates to the diagrams have been formalized. An **implementation** of these algorithms was carried out, without relying on existing libraries (except for unit tests).

## 1.4 Structure of the report

chapter 2 presents the theoretical model of quantum decision diagrams that has been developed, including the accompanying reduction and gating algorithms. Next, chapter 3 discusses the implementation carried out. Finally, a short conclusion is presented in chapter 4.

A more complete theoretical document is available online. Some theoretical points succinctly presented in this report, or theorems whose proof is not specified, are detailed in this document. [7] In addition, technical documentation of the code is available online. [8]

## Chapter 2

# Modèle théorique

Le modèle théorique proposé est celui des **diagrammes de décision quantiques additifs abstraits**.

### 2.1 Arithmétique des intervalles

L'arithmétique des intervalles dans le cadre de l'interprétation abstraite est proche de l'exemple de calcul sur les signes de la section 1.2. L'arithmétique des intervalles est utilisée pour déterminer un ensemble de valeurs possibles pour une expression mathématique en utilisant des intervalles pour les valeurs des variables.

L'arithmétique des intervalles réels a été étudiée [9], et l'arithmétique des intervalles complexes cartésien a fait l'objet d'études légères par le passé [10]. Au cours de ce projet, un travail a été réalisé pour explorer les possibilités de l'arithmétique des **intervalles complexes cartésiens et polaires**.

Les intervalles complexes cartésiens sont définis par un intervalle pour la partie réelle et un intervalle pour la partie imaginaire. De manière équivalente, ils sont définis comme le plus petit rectangle dans le plan complexe (orienté selon les axes réel et imaginaire) contenant deux complexes. Les intervalles complexes polaires sont définis par un intervalle pour le module et un intervalle pour l'argument. Ces deux types d'intervalles ont des représentations différentes dans le plan complexe, comme le montre la Figure 2.1. Dans un cas comme dans l'autre, l'équivalent abstrait d'une **opération**  $*$  est défini sur des éléments  $\alpha$  et  $\beta$  de l'ensemble des intervalles cartésiens  $\mathcal{A}_0$  ou polaires  $\mathcal{S}_0$  par

$$\alpha * \beta = \bigcap_{\gamma \supset \alpha \otimes \beta \text{ et } \gamma \in \mathcal{A}_0} \gamma \quad \text{où} \quad \alpha \otimes \beta = \{a * b; a \in \alpha, b \in \beta\}$$

Les opérations de somme, de produit et d'union ainsi construites sont **sur-approximées** : elles garantissent que l'ensemble des valeurs possibles est inclus dans l'ensemble abstrait, et d'avoir un intervalle pour résultat. Ces opérations ont des propriétés, de distributivité par exemple, parfois très différentes des nombres complexes qu'ils représentent. Des propriétés sur les intervalles cartésiens et polaires sont énoncées et démontrées dans le document annexe [7].

D'un point de vue pratique, les intervalles cartésiens sont généralement plus simples à manipuler que les intervalles polaires, et sont largement plus adaptés à une structure additive. Les intervalles polaires ont des avantages pour les opérations de multiplication et de division, mais rendent la somme coûteuse en calculs et en perte de précision.

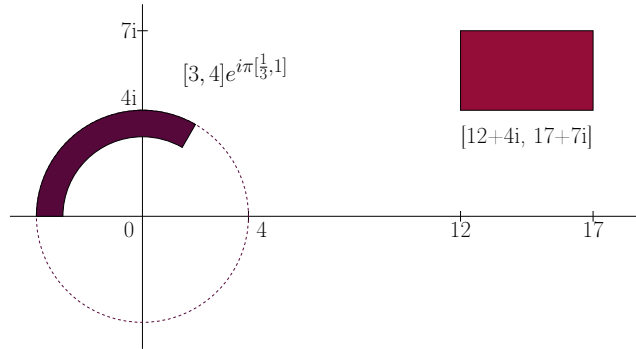


Figure 2.1: Exemple d'intervalles complexes cartésiens et polaires

## 2.2 États abstraits et diagrammes

Les **états** abstraits à  $n$  qubits sont définis comme des  $2^n$ -uplets d'intervalles complexes, on note leur ensemble  $\mathcal{A}_n$ . Il peut s'agir des intervalles cartésiens ou polaires, puisque les opérations sont définies de manière similaire, mais en pratique on se limite souvent aux intervalles cartésiens. On définit sur les états la relation d'ordre d'inclusion des états par l'inclusion des produits cartésiens des intervalles les composant.

Les **diagrammes** sont définis de manière récursive. Le seul diagramme de hauteur 0 est  $\boxed{1}$ , puis si l'ensemble  $\mathcal{D}_n$  des diagrammes de hauteur  $n$  est défini, les diagrammes de hauteur  $n + 1$  peuvent avoir un nombre fini de fils gauches dans  $\mathcal{D}_n$  et un nombre fini de fils droits dans  $\mathcal{D}_n$ , chacun étant associé à une amplitude abstraite sur la branche dans  $\mathcal{A}_0$

$$\mathcal{D}_{n+1} = \mathcal{P}_f(\mathcal{A}_0 \times \mathcal{D}_n) \times \mathcal{P}_f(\mathcal{A}_0 \times \mathcal{D}_n)$$

On peut ainsi définir la fonction d'évaluation  $\mathcal{E} : \mathcal{D}_n \rightarrow \mathcal{A}_n$  sur les diagrammes (une définition similaire est possible en utilisant les intervalles polaires), ce qui permet de définir une relation d'ordre  $\leq$  sur les diagrammes par inclusion des ensembles abstraits qu'ils représentent.

## 2.3 Approximations

L'un des objectifs pour cette structure de données est de transformer un diagramme en un autre incluant le diagramme initial et de taille plus faible. Sur les diagrammes de décision abstraits ont été développés deux algorithmes, à partir d'une relation de fusion.

Puisque traiter les diagrammes de manière globale est difficile, on réalise les approximations de manière locale. Plus formellement, une **approximation globale** est une fonction  $g : \mathcal{D}_n \rightarrow \mathcal{D}_n$  telle que

$$\forall D \in \mathcal{D}_n, D \leq g(D)$$

En pratique, on a surtout développé une **approximation par fusion**, qui est une fonction  $f : \mathcal{D}_n \times \mathcal{D}_n \rightarrow \mathcal{D}_n$  telle que

$$\begin{cases} \forall A \neq B \in \mathcal{D}_n, A \leq f(A, B) \text{ and } B \leq f(A, B) \\ \forall A \in \mathcal{D}_n, f(A, A) = A \end{cases}$$

Le **théorème de fusion** indique que si l'on dispose d'une approximation par fusion, alors on dispose d'une approximation globale. Il simplifie grandement les preuves ultérieures, puisque l'on peut se contenter de démontrer la propriété d'approximation par fusion pour montrer l'approximation globale.

D'un point de vue computationnel, les approximations par fusion ont aussi l'avantage de pouvoir s'appliquer à des sous-diagrammes : pour réduire un diagramme  $D$ , il suffira alors de réaliser une approximation par fusion sur deux sous-diagrammes de  $D$ . Il est alors possible de réduire un diagramme de manière locale, ce qui est plus efficace que de considérer le diagramme parent directement.

## 2.4 Réduction

Deux algorithmes de réduction ont été développés, utilisant largement l'approximation par fusion  $\text{fm}$  (pour **force merge**) de la Figure 2.2, qui est centrale dans la réduction des diagrammes.

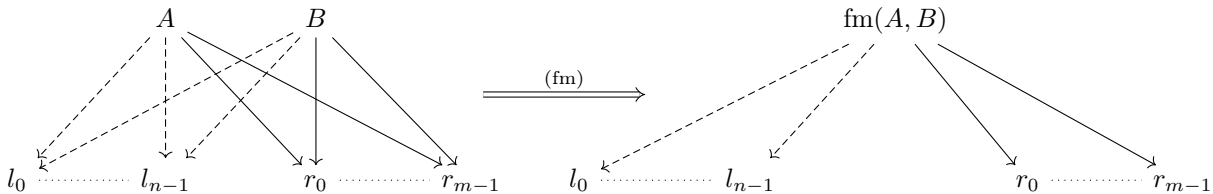


Figure 2.2: Approximation par fusion forcée

où si  $\text{ampl}(A, x)$  est l'amplitude abstraite sur le lien entre  $A$  et  $x$ , alors les nouvelles amplitudes abstraites sont définies par la formule suivante

$$\forall x \in \{l_0, \dots, l_{n-1}, r_0, \dots, r_{m-1}\}, \text{ampl}(\text{fm}(A, B), x) = \text{ampl}(A, x) \sqcup \text{ampl}(B, x)$$

où  $\sqcup$  est l'opération d'union des intervalles complexes (cartésiens ou polaires). Cette approximation par fusion fonctionne en pratique y compris sur des diagrammes n'ayant a priori pas de descendance commune, puisqu'il suffit d'ajouter des branches avec un poids nul pour se ramener à ce cas. On remarque que cette généralisation ne fonctionne que dans le cas où on permet l'utilisation de diagrammes additifs.

## 2.5 Exemple d'approximation

Considérons le diagramme suivant, où tous les fils (gauche ou droit) sont  $\boxed{1}$  et où les branches sans amplitude écrite sont d'amplitude 1. Appliquer  $\text{fm}$  sur  $A$  et  $B$  permet fait passer le diagramme additif initial à un diagramme additif abstrait  $D' \geq D$ .

Ici, la fusion permettrait ensuite de fusionner les deux branches gauches de  $D'$  pour obtenir un diagramme non additif, comme le montre la Figure 2.3.

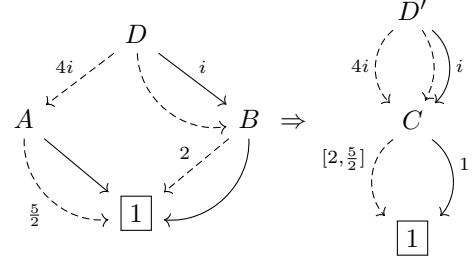


Figure 2.3: Fusion de  $A$  et  $B$

## 2.6 Erreur

S'il est possible de réaliser des fusions sur n'importe quelle paire de diagrammes de même hauteur, savoir lesquels fusionner afin de réduire le diagramme sans causer de trop grande perte de précision est un enjeu majeur. On a donc développé une grandeur d'**erreur**, qui permet de déterminer si une fusion est intéressante ou non. Cette grandeur se distingue en deux intervalles :  $\rho$  et  $\varepsilon$ , définis inductivement sur les diagrammes de hauteur  $n$  par

$$\rho(\boxed{1}) = \{1\}$$

$$\varepsilon(\boxed{1}) = \{0\}$$

$$\forall G, D \in \mathcal{P}_f(\mathcal{A}_0 \times \mathcal{D}_n), \rho((G, D)) = \left( \sum_{(l, L) \in G} l \rho(L) \right) \sqcup \left( \sum_{(r, R) \in D} r \rho(R) \right)$$

$$\forall G, D \in \mathcal{P}_f(\mathcal{A}_0 \times \mathcal{D}_n),$$

$$\varepsilon((G, D)) = \left( \sum_{(l, L) \in G} l \max |\rho(L) \ominus \varepsilon(L)| + \varepsilon(L) \right) \sqcup \left( \sum_{(r, R) \in D} r \max |\rho(R) \ominus \varepsilon(R)| + \varepsilon(R) \right)$$

où  $\alpha^c$  est la version centrée d'un intervalle cartésien et où  $\ominus$  est l'opération de « rognage » illustrée Figure 2.4 et définie lorsque  $\alpha \subset \beta$  telle que

$$\forall \alpha, \beta \in \mathcal{A}_0, (\alpha \ominus \beta) + \beta = \alpha$$

Arriver à cette définition pour la grandeur d'erreur a nécessité des recherches conséquentes au cours du semestre. Plusieurs autres définitions ont été mises à l'épreuve, mais celle-ci a été retenue comme la plus prometteuse.

Ceci pourra faire l'objet de recherches ultérieures plus expérimentales, fondées sur des *benchmarks* de réduction de diagrammes. Le choix de cette définition fait respecter à ces grandeurs plusieurs propriétés intéressantes, en particulier  $\rho$  contient tous les autres intervalles de l'évaluation

$$\forall D \in \mathcal{D}_n, \forall i \in \{0, \dots, 2^n - 1\}, \mathcal{E}(D)[i] \subset \rho(D)$$

ou le fait que l'intervalle  $\varepsilon$  soit toujours centré. Le calcul de ces grandeurs, s'il est correctement stocké dans la structure de données, n'induit pas de temps de calcul rédhibitoire puisqu'il ne nécessite pas de calculer l'évaluation entière à chaque changement (il suffit de « propager » un changement dans un diagramme fils aux parents).

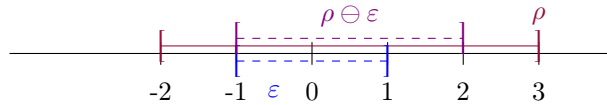


Figure 2.4: Opération de « rognage »  $\ominus$  dans le cas des réels

## 2.7 Application de portes

Il n'existe pas par défaut de formalisation des portes quantiques dans le cas de diagrammes de décision quantiques additifs abstraits, puisque cette structure de données est nouvelle. On a donc défini une application de portes  $M \in \mathcal{M}_{2^n, 2^n}(\mathbb{C})$  à un diagramme  $D \in \mathcal{D}_n$  préservant ce qu'on attend comme effet sur l'évaluation des diagrammes, c'est-à-dire

$$\mathcal{E}(M(D)) = M \cdot \mathcal{E}(D)$$

où  $\cdot$  est le produit matriciel basé sur le produit dans  $\mathcal{A}_0$ . On note que, sans perte de généralité, on peut supposer que les coefficients de  $M$  sont eux-mêmes des intervalles complexes. Réalisons un exemple d'application de porte sur un diagramme  $D \in \mathcal{D}_n$ .

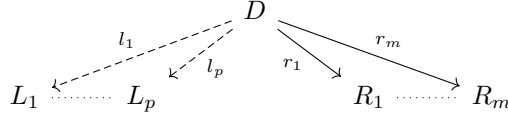


Figure 2.5: Diagramme  $D$

En pratique, pour appliquer une porte  $M$  à un diagramme  $D$

1. On sépare  $M$  en 4 sous-matrices

$$M = \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix}$$

2. On crée pour chaque branche gauche de  $D$  une branche droite de même amplitude et pour chaque branche droite de  $D$  une branche gauche de même amplitude.
3. On applique  $M_{00}$  à chaque branche gauche (hors celles créées à l'étape 2)
4. On applique  $M_{01}$  à chaque branche gauche (uniquement celles créées à l'étape 2)
5. On applique  $M_{10}$  à chaque branche droite (uniquement celles créées à l'étape 2)
6. On applique  $M_{11}$  à chaque branche droite (hors celles créées à l'étape 2)

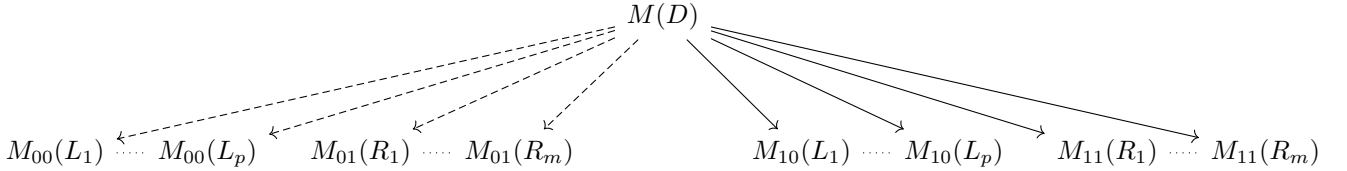


Figure 2.6: Diagramme  $M(D)$

L'évaluation de  $M(D)$  avec cet algorithme est correcte, puisque comme décrit sur la Figure 2.6, on a

$$\mathcal{E}(M(D)) = \begin{pmatrix} \sum l_i M_{00} \mathcal{E}(L_i) + \sum r_j M_{01} \mathcal{E}(R_j) \\ \sum l_i M_{10} \mathcal{E}(L_i) + \sum r_j M_{11} \mathcal{E}(R_j) \end{pmatrix} = M \cdot \mathcal{E}(D)$$

ce qui correspond bien à l'effet attendu de l'application de la porte  $M$  sur le diagramme  $D$ . Le cas de base (une matrice de taille 1, c'est-à-dire un scalaire) est traité simplement en multipliant les poids des branches par ce scalaire.

On a traité ici le cas d'une porte s'appliquant à tous les qubits. Plus généralement, considérant une porte s'appliquant seulement à  $k$  qubits contigus (disons les qubits  $Q = q, \dots, q + k - 1$ ). Pour un état  $v$  à  $n$  qubits, appliquer la porte  $P \in \mathcal{M}_{2^k, 2^k}(\mathbb{C})$  à aux qubits  $Q$  de  $v$  revient à appliquer à  $v$  tout entier la porte

$$M = \left( \bigotimes_{i=1}^{q-1} I \right) \otimes P \otimes \left( \bigotimes_{i=1}^{n-k-q-1} I \right)$$

On remarque donc que l'application d'une porte à un nombre restreint de qubits peut être vue comme l'application d'une porte à tous les qubits, mais avec des matrices identités à la place des matrices de la porte. De plus, utiliser des portes  $S$  afin d'échanger la place de qubits permet de traiter avec l'algorithme présenté précédemment tous les cas d'application de portes logiques à un sous-ensemble quelconque des qubits du circuit.



# Chapter 3

## Implémentation

L'implémentation a été réalisée en **langage C++**. Ce choix a été motivé par plusieurs raisons : la performance, la maturité du langage et son utilisation dans plusieurs projets historiques du domaine [11] [12], et la proximité avec le langage C, déjà bien connu.

### 3.1 Structure du code

Le code fait l'objet d'une documentation extensive, disponible en ligne. [8] L'implémentation utilise largement la **programmation orientée objet**, en définissant des classes pour les objets manipulés. Les classes principales sont les suivantes, du plus bas au plus haut niveau :

- Intervalles réels (quelconques, de réels positifs ou modulo  $2\pi$ ), et complexes (cartésiens ou polaires)
- **Diagrammes** (par défaut, additifs et abstraits)
- Les branches, qui contiennent un lien (**pointeur**) vers un digramme de destination et un intervalle complexe (cartésien ou polaire)

Les classes et fonctions sont organisées en **espaces de noms** (*namespaces*) pour éviter les conflits de noms, et définies dans des fichiers séparés. Des fonctions de **réduction** servent ensuite à réduire les diagrammes, en utilisant les fonctions de sélection.

Les définitions des fonctions et méthodes et leur implémentation sont séparées dans des fichiers d'en-tête (*header*) et des fichiers de code source (*source*), respectivement. Afin de tirer parti de la **compilation séparée**, les fichiers d'en-tête sont inclus dans les fichiers de code source, et les fichiers de code source sont compilés en bibliothèques statiques.

Au cours de ce semestre, un effort a été fait pour améliorer l'**interchangeabilité** entre les intervalles cartésiens et polaires, et pour rendre le code plus générique, par exemple en définissant des types adaptés et optimisés pour les matrices de portes ou les états (contenant des complexes ou des intervalles).

### 3.2 Tests

Afin de garantir la qualité du code, des **tests unitaires** ont été écrits pour la plupart des fonctions et méthodes. Ces tests sont écrits en utilisant la bibliothèque open-source *Google Test*, et sont organisés en **suites de tests** pour chaque classe. [13] Les 46 tests, qui s'exécutent en moins d'une demi-seconde et comptent plusieurs milliers d'assertions, permettent de valider le code et de détecter les bugs ou régressions.

### 3.3 Interpréteur QASM

Un **interpréteur** pour le langage Open QASM a été réalisé au cours de ce semestre. Il permet de lire un fichier QASM définissant des qubits et y appliquant des portes logiques, d'en générer un diagramme et d'y faire les modifications correspondant aux portes qu'on souhaite lui appliquer, et d'en afficher l'évaluation.

Cet interpréteur peut aussi fonctionner en mode interactif, c'est-à-dire en lisant les instructions depuis l'entrée standard. Il ne supporte pas l'ensemble des instructions QASM, seulement les portes logiques de base ( $X$ ,  $H$ ,  $CX$ ,  $S$ , portes de phase). Toutefois, le *back-end* de cet interpréteur est capable d'appliquer n'importe quelle porte grâce à la méthode détaillée en section 2.7. Cet interpréteur est constitué d'une bibliothèque qu'on peut inclure avec l'en-tête `qasm.h` comportant un *namespace* `qasm` et d'un exécutable `prompt`.

Le QASM étant déjà un langage de description de circuits quantiques, il n'a pas été jugé nécessaire de réaliser véritable compilateur comportant un front-end, un middle-end réalisant des optimisations et un back-end transformant la représentation intermédiaire en exécutable. L'interpréteur a été testé avec succès sur plusieurs exemples de circuits quantiques.

L'exemple du programme 1.2 s'exécute en 2,6 ms. Des **benchmarks** sur un plus grand nombre de qubits, ou sur des circuits plus complexes, n'ont pas été réalisés. De plus il est probable qu'une partie non négligeable du temps d'exécution soit due à la lecture du fichier ou à l'interprétation du QASM, et non à la génération du diagramme et à sa modification.

### 3.4 Outils

Le code source est versionné à l'aide du logiciel *Git*, et est disponible sur la plateforme *GitHub*. [14] [15] [7] Ce projet fait l'objet d'une **intégration continue** utilisant *GitHub Actions*, automatisant la validation des tests.

Ce semestre a vu l'arrivée de l'utilisation de *Clang* comme compilateur à la place de *GCC*, et de *Ninja* à la place de *Make*. [16] [17] Ces changements ont permis de réduire le temps de compilation, et d'améliorer la lisibilité des messages d'erreur, ainsi que de faciliter l'utilisation de fonctionnalités récentes du langage (datant de C++23). Ces outils sont orchestrés par *CMake*. [18]

Compiler le projet (tests exclus) prend environ 8 secondes sur un ordinateur portable récent. Il faut 5 secondes supplémentaires pour compiler les tests. Le projet compte environ 5 000 lignes de code, dont environ 1 000 lignes de tests.

Le projet fait aussi l'objet d'une **documentation**, écrite dans les commentaires des fichiers d'en-tête. Les pages web de documentation sont générées automatiquement par *Doxygen*. [19] et publiée automatiquement sur *GitHub Pages* à chaque modification à l'aide de *GitHub Actions*. [8]

# Chapter 4

## Conclusion

### 4.1 Travail réalisé

Le modèle qui a été développé, avec algorithme de réduction, permet de limiter autant que souhaité la taille d'un état en mémoire, au prix d'une perte de précision. Le cadre mathématique de celui-ci a été défini, et des algorithmes de réduction ont été prouvés sur cette structure de données.

L'implémentation, n'a pas encore fourni de résultats expérimentaux significatifs, mais permet déjà de simuler des diagrammes de décision quantiques de manière performante. Sa robustesse est assurée par des tests unitaires.

Au cours de ce semestre, des efforts sur l'implémentation ont été réalisés, entre autres afin de rendre interchangeables les intervalles complexes cartésiens et polaires mais aussi plus généralement pour améliorer la qualité et réusabilité du code. La documentation du code a été améliorée, et le document principal détaillant les aspects théoriques du projet a été complété.

Un interpréteur QASM a été réalisé au cours de ce semestre, se reposant sur des travaux théoriques ayant permis une implémentation de l'application de portes à des diagrammes.

Le travail réalisé sur cette période est donc prometteur, et s'inscrit dans la continuité du travail réalisé au semestre dernier : malgré des difficultés rencontrées, tant d'un point de vue technique en programmation que dans la définition du modèle, ayant parfois amené à revenir en arrière avant de trouver une bonne définition de l'erreur par exemple, les résultats théoriques obtenus sont prometteurs.

### 4.2 Perspectives

Plusieurs perspectives peuvent être envisagées pour prolonger les travaux réalisés. D'une part, puisque plusieurs choix dans la définition du modèle ont été faits de manière arbitraire parmi plusieurs options possibles, il est pertinent de réaliser de nombreuses simulations dans des configurations différentes afin de déterminer les choix les plus pertinents.

On pourra notamment réaliser des tests de performance sur plusieurs fonctions d'erreur, ou observer l'effet d'échanges de qubits sur les performances en termes de manque de précision. De manière générale, il manque des **résultats expérimentaux** et en particulier des comparaisons à d'autres implémentations de la littérature sur des exemples usuels.

Nous pourrions aussi étendre le modèle avec d'autres concepts, comme les automates d'arbres pour la **vérification** ou les diagrammes de décisions et applications localement inversibles. [20] [21] Enfin, l'implémentation aurait à gagner d'être plus facile d'utilisation, par exemple avec une **interface graphique** pouvant agrémenter l'interpréteur QASM. Ces axes de travail pourront être explorés au cours de la suite du projet s'il est prolongé.

# Bibliography

- [1] D. Deutsch and R. Jozsa. “Rapid solution of problems by quantum computation”. In: *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences* 439 (1907 1992), pp. 553–558. DOI: 10.1098/rspa.1992.0167.
- [2] Lov K. Grover. *A fast quantum mechanical algorithm for database search*. 1996. arXiv: quant-ph/9605043 [quant-ph].
- [3] Peter W. Shor. “Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer”. In: *SIAM Journal on Computing* 26.5 (Oct. 1997), pp. 1484–1509. ISSN: 1095-7111. DOI: 10.1137/s0097539795293172.
- [4] Andrew W. Cross et al. *Open Quantum Assembly Language*. 2017. arXiv: 1707.03429 [quant-ph].
- [5] P. Cousot and R. Cousot. “Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints”. In: *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. Los Angeles, California: ACM Press, New York, NY, 1977, pp. 238–252.
- [6] Mads Rosendahl. *Introduction to abstract interpretation*. CS University of Copenhagen, 1995.
- [7] Malo Leroy. *Abstract additive quantum decision diagrams*. Feb. 2025. URL: <https://github.com/Firefnix/coto/>.
- [8] Malo Leroy. *Coto Documentation*. Feb. 2025. URL: <https://firefnix.github.io/coto/>.
- [9] Teruo Sunaga. “Theory of an interval algebra and its application to numerical analysis [Reprint of Res. Assoc. Appl. Geom. Mem. 2 (1958), 29–46]”. In: *Japan Journal of Industrial and Applied Mathematics* 26.2-3 (2009), pp. 125–143.
- [10] J. Rokne and P. Lancaster. “Complex interval arithmetic”. In: *Commun. ACM* 14.2 (Feb. 1971), pp. 111–112. ISSN: 0001-0782. DOI: 10.1145/362515.362563.
- [11] Benjamin Bichsel et al. “Abstraqt: Analysis of Quantum Circuits via Abstract Stabilizer Simulation”. In: *Quantum* 7 (Nov. 2023), p. 1185. ISSN: 2521-327X. DOI: 10.22331/q-2023-11-20-1185.
- [12] Jean-Baptiste Debrize Martin Olivier and Théo Fourcat. *QTranslator – Assembly to quantum assembly*. June 2022. URL: <https://github.com/PoCInnovation/QTranslator>.
- [13] Google. *Google Test v1.16.0*. Feb. 2025. URL: <https://google.github.io/googletest/>.
- [14] Git. *Git v2.47.1*. 2005–2025. URL: <https://git-scm.com/>.
- [15] GitHub. *GitHub*. 2008–2025. URL: <https://github.com/>.
- [16] Apple Inc. *Clang v19.1.7*. 2007–2025. URL: <https://clang.llvm.org/>.
- [17] Evan Martin. *Ninja v1.12.1*. 2012–2024. URL: <https://ninja-build.org/>.
- [18] Kitware. *CMake v3.31.5*. 2000–2025. URL: <https://gitlab.kitware.com/cmake/cmake>.
- [19] Doxygen Dimitri van Heesch. *Doxygen v1.13.2*. 1997–2024. URL: <https://www.doxygen.nl>.
- [20] Yu-Fang Chen et al. *An Automata-based Framework for Verification and Bug Hunting in Quantum Circuits (Technical Report)*. 2023. arXiv: 2301.07747 [cs.LG].
- [21] Lieuwe Vinkhuijzen et al. “LIMDD: A Decision Diagram for Simulation of Quantum Computing Including Stabilizer States”. In: *Quantum* 7 (Sept. 2023), p. 1108. ISSN: 2521-327X. DOI: 10.22331/q-2023-09-11-1108.