CentraleSupélec

université PARIS-SACLAY

Around Quantum Decision Diagrams

Report – Research Track – Semester 7

Student : Malo Leroy
Supervisor : Renaud Vilmart

Contents

February 17th 2025

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 α, β 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].

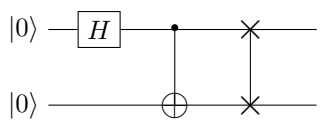

Figure 1.1: 2-qubits circuit with gates H, CX and S

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 -12×7 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.

x_1	x_2	x_3	$f(x_1, x_2)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

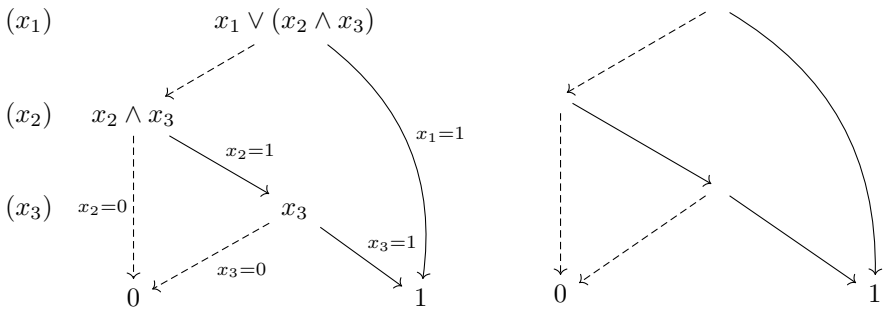

Figure 1.3: (a) Table de vérité (b) Diagramme de décision (c) Diagramme de décision sans labels

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

Theoretical model

The proposed theoretical model is that of **abstract additive quantum decision diagrams**.

2.1 Interval arithmetic

Interval arithmetic in the context of abstract interpretation is similar to the example of calculating the signs of the section 1.2. Interval arithmetic is used to determine a set of possible values for a mathematical expression using intervals for variable values.

Real interval arithmetic has been studied [9], and Cartesian complex interval arithmetic has been lightly studied in the past [10]. In the course of this project, work has been carried out to explore the possibilities of **Cartesian and polar complex interval arithmetic**.

Cartesian complex intervals are defined by an interval for the real part and an interval for the imaginary part. Equivalently, they are defined as the smallest rectangle in the complex plane (oriented along the real and imaginary axes) containing two given complex numbers. Polar complex intervals are defined by an interval for the modulus and an interval for the argument. These two types of interval have different representations in the complex plane, as shown in Figure 2.1. In either case, the abstract equivalent of a **operation** $*$ is defined on elements α and β of the set of Cartesian intervals $\mathcal{A}_0$ or polar intervals $\mathcal{S}_0$ by

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

The sum, product and union operations thus constructed are **over-approximated**: they guarantee that the set of possible values is included in the abstract set, and have an interval as a result. These operations have properties, such as distributivity, which are sometimes very different from the complex numbers they represent. Properties of Cartesian and polar intervals are set out and demonstrated in the accompanying document [7].

From a practical point of view, Cartesian intervals are generally simpler to handle than polar intervals, and are largely more suited to an additive structure. Polar intervals have advantages for multiplication and division operations, but make summation computationally expensive and often result in a huge loss of precision in this case.

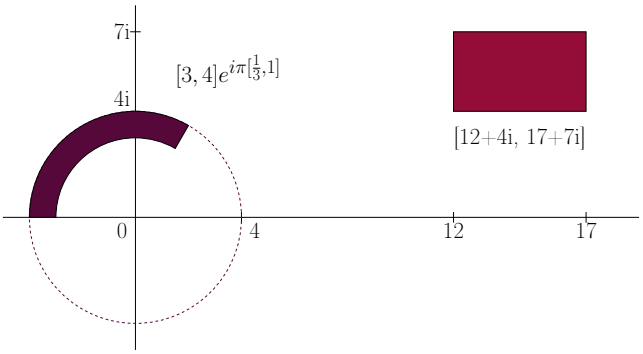

Figure 2.1: Example of Cartesian and polar complex intervals

2.2 Abstract states and diagrams

Abstract n-qubit **states** are defined as 2^n-uplets of complex intervals, and their set $\mathcal{A}_n$ is noted. These can be Cartesian or polar intervals, since the operations are defined in a similar way, but in practice we often restrict ourselves to Cartesian intervals. The inclusion order relation of states is defined by the inclusion of the Cartesian products of their component intervals.

The **diagrams** are defined recursively. The only diagram of height 0 is $\boxed{1}$, then if the set $\mathcal{D}_n$ of diagrams of height n is defined, diagrams of height $n+1$ can have a finite number of left-hand threads in $\mathcal{D}_n$ and a finite number of right-hand threads in $\mathcal{D}_n$, each associated with an abstract amplitude on the branch in $\mathcal{A}_0$.

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

That way, a diagram in $\mathcal{D}_n$ represents n qubits. We can thus define the evaluation function $\mathcal{E} : \mathcal{D}_n \to \mathcal{A}_n$ on the diagrams (a similar definition is possible using polar intervals), which makes it possible to define a relation of order $\leq$ on the diagrams by inclusion of the abstract sets they represent.

2.3 Approximations

One of the aims of this data structure is to transform one diagram into another, including the initial diagram, but of smaller size. Two algorithms have been developed for abstract decision diagrams, based on a merge relation.

Since dealing with diagrams globally is difficult, approximations are performed locally. More formally, a **global approximation** is a function $g : \mathcal{D}_n \to \mathcal{D}_n$ such that

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

In practice, we have mainly developed a **approximation by fusion**, which is a function $f : \mathcal{D}_n \times \mathcal{D}_n \to \mathcal{D}_n$ such that

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

The **fusion theorem** states that if we have a fusion approximation, then we have a global approximation. It greatly simplifies subsequent proofs, since we can simply demonstrate the fusion approximation property to show the global approximation.

From a computational point of view, fusion approximations also have the advantage of being applicable to subdiagrams: to reduce a D diagram, it will then suffice to perform a fusion approximation on two subdiagrams of D. It is then possible to reduce a diagram locally, which is more efficient than considering the parent diagram directly.

2.4 Reduction

Two reduction algorithms have been developed, making extensive use of the fm fusion approximation (for **force merge**) of the Figure 2.2, which is central to diagram reduction.

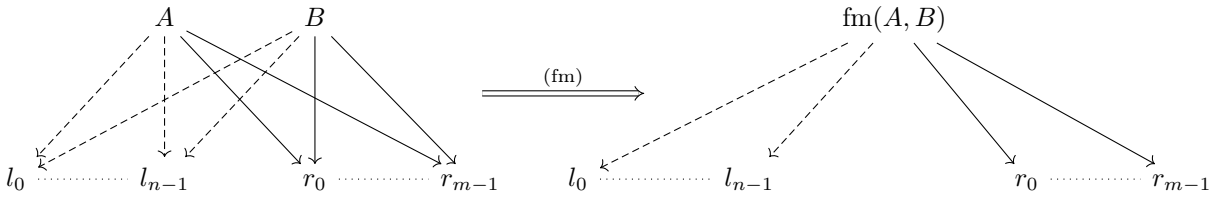

Figure 2.2: Approximation by forced merge

where if ampl(A, x) is the abstract amplitude on the link between A and x, then the new abstract amplitudes are defined by the following formula

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

where $\sqcup$ is the complex interval union operation (Cartesian or polar). In practice, this fusion approximation works even on diagrams with no common descent, since it's enough to add branches with zero weight to bring us back to this case. Note that this generalization only works when additive diagrams are allowed.

2.5 Example of approximation

Consider the following diagram, where all children (left or right) are $\boxed{1}$ and branches with no written amplitude are amplitude 1. Applying fm to A and B turns the initial additive diagram into an abstract additive diagram $D' \geq D$.

Here, merging would then merge the two left-hand branches of D' to obtain a non-additive diagram, as shown in Figure 2.3.

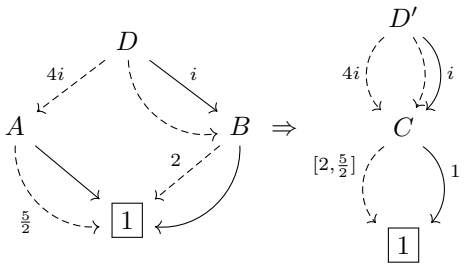

Figure 2.3: Merger of A and B

2.6 Erreur

If it's possible to perform merges on any pair of diagrams of the same height, knowing which ones to merge in order to reduce the diagram without causing too great a loss of accuracy is a major issue. We have therefore developed a scale of **error**, which can be used to determine whether a merge is interesting or not. This variable can be divided into two intervals: ρ and ε, defined inductively on diagrams of height n by

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

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

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

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

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

where α^c is the centered version of a Cartesian interval and where $\ominus$ is the "cropping" operation illustrated in Figure 2.4 and defined when $\alpha \subset \beta$ such that

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

Arriving at this definition for the error quantity required considerable research over the course of the semester. Several other definitions were put to the test, but this was selected as the most promising.

This could be the subject of further, more experimental research, based on *benchmarks* of diagram reduction. The choice of this definition gives these variables several interesting properties, in particular ρ contains all the other evaluation intervals

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

and ε is always centered on zero. The calculation of these quantities, if correctly stored in the data structure, doesn't induce any prohibitive calculation time, since it doesn't require calculating the entire evaluation for each change (it's enough to "propagate" a change in a child diagram to the parents).

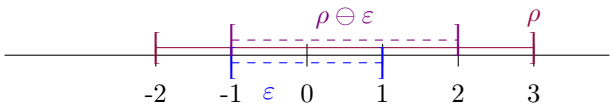

Figure 2.4: "Cropping" operation $\ominus$ in the case of real intervals

2.7 Gate application

By default, there is no formalization of quantum gates in the case of abstract additive quantum decision diagrams, since this data structure is new. We therefore defined an application of gates $M \in \mathcal{M}_{2^n,2^n}(\mathbb{C})$ to a diagram $D \in \mathcal{D}_n$ preserving what we expect to be the effect on diagram evaluation, i.e.

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

where $\cdot$ is the matrix product based on the product in $\mathcal{A}_0$. Note that, without loss of generality, we can assume that the coefficients of M are themselves complex intervals. Let's take an example of a gate application on a diagram $D \in \mathcal{D}_n$.

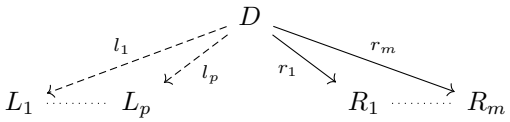

Figure 2.5: Diagramme D

In practice, to apply a gate M to a diagram D

1. We split M into 4 sub-matrices
$$M = \begin{pmatrix} M_{00} & M_{01} \\ M_{10} & M_{11} \end{pmatrix}$$
2. For each left-hand branch of D, a right-hand branch of the same amplitude is created, and for each right-hand branch of D, a left-hand branch of the same amplitude is created.
3. We apply M_{00} to each left branch (except those generated at step 2)
4. We apply M_{01} to each left branch (only those generated at step 2)
5. We apply M_{10} to each right branch (only those generated at step 2)
6. We apply M_{11} to each right branch (except those generated at step 2)

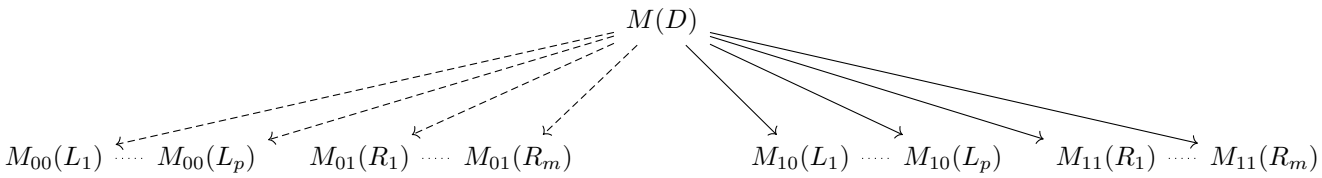

Figure 2.6: Diagram $M(D)$

The evaluation of $M(D)$ with this algorithm is correct, since like described by Figure 2.6, we have

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

which is the expected effect of applying the M gate to the D diagram. The basic case (a matrix of size 1, i.e. a scalar) is treated simply by multiplying the branch weights by this scalar.

We have dealt here with the case of a gate applying to all qubits. More generally, consider a gate applying only to k contiguous qubits (say the $Q = q, ..., q + k - 1$ qubits). For a state v of n qubits, applying a gate $P \in \mathcal{M}_{2^k,2^k}(\mathbb{C})$ to the qubits Q of v returns to applying to the whole v state vector the matrix

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

We can therefore see that the application of a gate to a restricted number of qubits can be seen as the application of a gate to all qubits, but with identity matrices instead of gate matrices. Moreover, using S gates to swap the place of qubits enables the algorithm presented above to handle all cases of logic gates being applied to any subset of the circuit's qubits.

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 2π), 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 étés 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 interchangables 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'incrit 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 étés 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 pourrons 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 Debize 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.LO].

[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.