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].

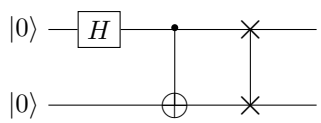

Figure 1.1: 2-qubits circuit with gates H, CX and S

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

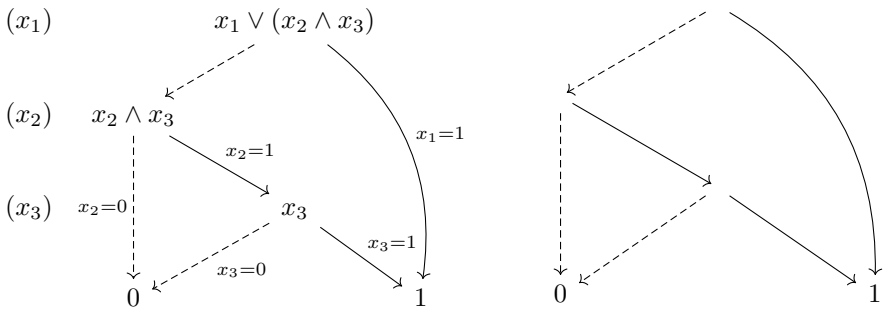

Figure 1.3: (a) Table de vérité (b) Diagramme de décision (c) Diagramme de décision sans labels

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.

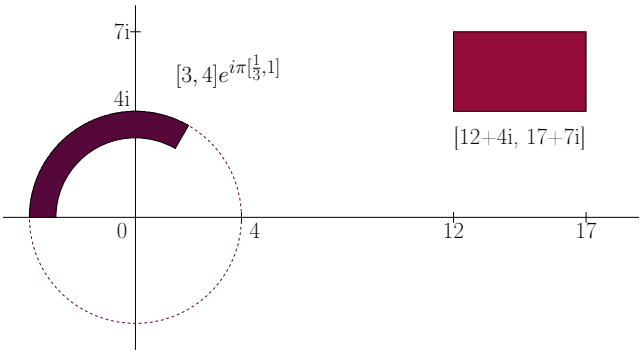

Figure 2.1: Example of Cartesian and polar complex intervals

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.

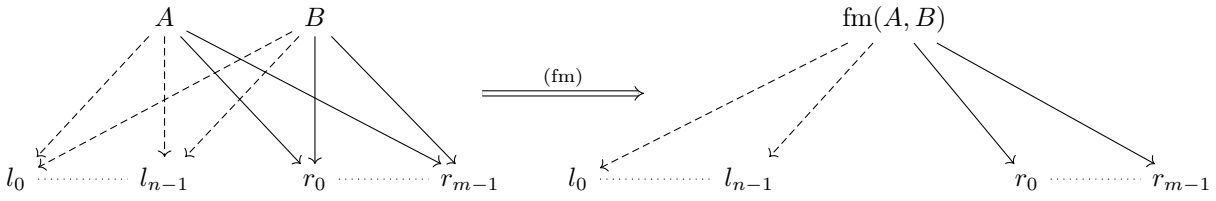

Figure 2.2: Approximation by forced merge

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.

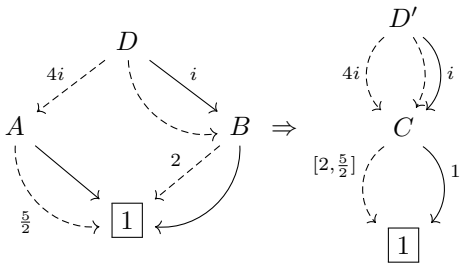

Figure 2.3: Merger of A and B

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).

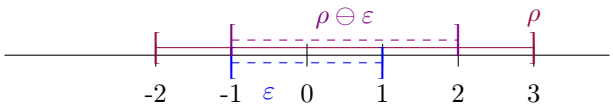

Figure 2.4: "Cropping" operation $\ominus$ in the case of real intervals

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 $D \in \mathcal{D}_n$.

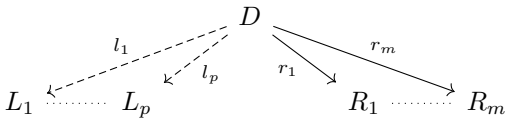

Figure 2.5: Diagramme D

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)

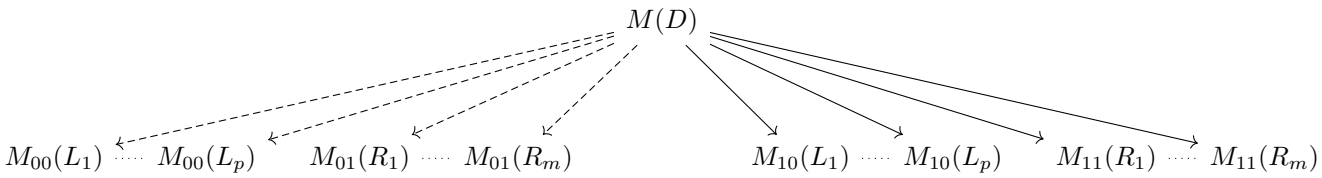

Figure 2.6: Diagram $M(D)$

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

Implementation

The implementation was written in **C++**. This choice was motivated by several factors: performance, the maturity of the language and its use in several historical projects in the field [11] [12], and its proximity to the well-known C language.

3.1 Structure du code

The code is the subject of extensive documentation, available online. [8] The implementation makes extensive use of **object-oriented programming**, defining classes for the objects handled. Some of the main classes are as follows, from lowest to highest level:

- Real intervals (any, of positive reals or modulo 2π), and complex intervals (Cartesian or polar)
- **Diagrams** (by default, additive and abstract)
- Branches, which contain a link (**pointer**) to a destination digram and a complex interval (Cartesian or polar)

Classes and functions are organized in **namespaces** to avoid name conflicts, and defined in separate files. **reduction** functions are then used to reduce diagrams, using selection functions.

The definition of functions and methods and their implementation are separated into header files and source code files, respectively. To take advantage of **separate compilation**, header files are included in source code files, and source code files are compiled into static libraries.

During this semester, an effort was made to improve **interchangeability** between Cartesian and polar intervals, and to make the code more generic, for example by defining adapted and optimized types for gate matrices or states (containing complexes or intervals).

3.2 Tests

To guarantee code quality, **unit tests** have been written for most functions and methods. These tests are written using the open-source *Google Test* library, and are organized into **test suites** for each class. [13] The 46 tests, which run in less than half a second and include several thousand assertions, are used to validate code and detect bugs or regressions.

3.3 QASM interpreter

An **interpreter** for the Open QASM language was developed during this semester. It can be used to read a QASM file defining qubits and applying quantum gates to them, generate a diagram and make the modifications corresponding to the gates you wish to apply to it, and display the evaluation.

This interpreter can also operate in interactive mode, i.e. reading instructions from standard input. It does not support all QASM instructions, only basic logic gates (X, H, CX, S, phase gates). However, this interpreter's *back-end* is capable of applying any gate using the method detailed in section 2.7. This interpreter consists of a library that can be included with the header `qasm.h` including a `qasm` namespace and an `prompt` executable.

As QASM is already a quantum circuit description language, oppositely to a high-level programming language, it was not considered necessary to build a true compiler comprising a front-end, a middle-end performing

optimizations and a back-end transforming the intermediate representation into an executable. The interpreter has been successfully tested on several examples of quantum circuits.

The example program 1.2 runs in 2.6 ms. **benchmarks** on a larger number of qubits, or on more complex circuits, have not been carried out. Furthermore, it is likely that a significant part of the execution time is due to reading the file or interpreting the QASM, and not to generating the diagram and modifying it.

3.4 Tools

The source code is versioned using *Git*, and is available on the *GitHub* platform. [14] [15] [7] This project is the subject of a **continuous integration** using *GitHub Actions*, automating test validation.

This semester saw the arrival of *Clang* as a compiler instead of *GCC*, and *Ninja* instead of *Make*. [16] [17] These changes have reduced compilation time and improved the readability of error messages, as well as facilitating the use of recent language features (dating back to C++23). These tools are orchestrated by *CMake*. [18]

Compiling the project (excluding tests) takes about 8 seconds on a recent laptop. It takes a further 5 seconds to compile the tests. The project has around 5,000 lines of code, including around 1,000 lines of tests.

The project is also the subject of a **documentation**, written in the comments of the header files. Documentation web pages are automatically generated by *Doxygen*. [19] and automatically published on *GitHub Pages* each time they are modified using *GitHub Actions*. [8]

Chapter 4

Conclusion

4.1 Work completed

The model that has been developed, with reduction algorithms, makes it possible to limit the size of a state in memory as much as desired, at the cost of a loss of precision. A mathematical framework has been clearly defined, and reduction algorithms have been proven on this data structure.

The implementation has not yet produced any significant experimental results as thorough benchmarks have not been run yet, but it is already capable of simulating and reducing quantum decision diagrams. Its robustness is ensured by unit tests.

During this semester, efforts were made on the implementation, among other things to make Cartesian and polar complex intervals interchangable, but also more generally to improve code quality and reusability. Code documentation has been improved, and the main document detailing the theoretical aspects of the project has been completed.

A QASM interpreter was produced during this semester, based on theoretical work which enabled the implementation of the application of gates to diagrams.

The work carried out over this period is therefore promising, and follows on from the work carried out last semester: despite the difficulties encountered, both from a technical point of view in programming and in defining the model, which sometimes led us to go backwards before finding a good definition of the error or a convincing way to apply quantum gates, for example, the theoretical results obtained are promising.

4.2 Perspectives

Several perspectives can be envisaged to extend the work carried out. On the one hand, since several choices in the definition of the model have been made arbitrarily among several possible options, it is relevant to carry out numerous simulations in different configurations in order to determine the most pertinent choices.

In particular, we can carry out performance tests on several error functions, or observe the effect of qubit exchanges on performance in terms of lack of precision. Generally speaking, we lack **experimental results** and in particular comparisons with other implementations in the literature on usual examples.

We could also extend the model with other concepts, such as tree automata for **verification** or locally invertible decision diagrams and applications. [20] [21] Finally, the implementation would benefit from being more user-friendly, for example with a **graphical user interface** that could enhance the QASM interpreter. If the project is extended, these focus areas can be explored in the future.

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.