NExt ApplicationS of Quantum Computing

Quantum Rule-Based Systems (QRBS) Models, Architecture and Formal Specification (D6.2)

Document Properties

Contract Number	951821
Contractual Deadline	M08 (30/04/2021)
Dissemination Level	Public
Nature	Report
Edited by :	Vicente Moret-Bonillo
Authors	Vicente Moret-Bonillo (UDC), Eduardo Mosqueira-Rey (UDC), Samuel Magaz-Romero (UDC), Andrés Gómez-Tato (CESGA)
Reviewers	Mohamed Hibti (EDF), Alfons Laarman (ULEI)
Date	15/04/2021
Keywords	Models. Architecture. Formal Specification
Status	Final
Release	1.5

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 951821

History of Changes

Release	Date	Author, Organization	Description of Changes
1.1	24/02/2021	Vicente Moret, UDC	First Draft. Deliverable-6.2, WP-6, Task 6.2
1.2	11/03/2021	Vicente Moret, UDC	Second Draft. Deliverable-6.2, WP-6, Task 6.2
1.3	22/03/2021	Vicente Moret, UDC	For Review. Deliverable-6.2, WP-6, Task 6.2
1.4	15/04/2021	Vicente Moret, UDC	For Review. Deliverable-6.2, WP-6, Task 6.2
1.5	26/04/2021	Vicente Moret, UDC	Final version

Table of Contents

1. Executive Summary

This report is the first deliverable of UDC-CESGA, related to task 6.2 of Work Package 6 of the NEASQC project, UC6. The document incorporates information on the approach to the work carried out so far, from the project start date to the deadline established for this first deliverable.

The report includes a brief description of invasive ductal carcinoma of the breast (IDC), the methodology followed for the modeling of a rule-based system for the diagnosis and treatment of IDC, a preliminary analysis to evaluate the suitability of quantum computing in this domain, a proposal about the quantum approximation that we want to use, and that we will later have to evaluate, and the analysis about the formal requirements of the application that we intend to carry out. We also include a quantum proposal on the uncertainty associated with reasoning in medicine.

A brief summary of the IDC is necessary to place the use case in the context of the project. The description will range from the initial symptoms that allow the clinician to consider the possibility of IDC, the diagnostic process, the degree of severity of the IDC, and the possible associated treatments.

The methodological description of the knowledge engineering used is necessary to understand the architecture of a classical rule-based system, and to be able to formalize the problem in terms of declarative knowledge, procedural knowledge and inferential circuits.

Next, a qualitative analysis of the problem in terms of quantum logical operators is presented to illustrate the possibility of converting a conventional rule-based system into a quantum rule-based system.

Finally, the formal requirements of the quantum rule-based system will be mentioned. Also, we will pay special attention to the imprecision of the information and the uncertainty associated with clinical practice.

2. Context

2.1. Project

In the context of the project this document describes the first stages of this use case in which we intend the development of a quantum rule-based system (QRBS) for Invasive Ductal Carcinoma (IDC) management. An inherent problem with conventional rule-based systems (RBS) is their great sensitivity to the number of hypotheses, data and rules of the system itself. More specifically, selecting which rules are applicable at each moment can greatly slow down the inferential process (Moret-Bonillo, Fundamentos de la Inteligencia Artificial, 2000). This process, called pattern matching, is an unresolved issue in Artificial Intelligence (AI).

In this respect the specific characteristics of quantum computing, such as quantum superposition or entaglement, could increase the computational power of our programs and be useful for AI in general, and for RBS in particular.

The algorithms and their implementation will be used to build a quantum rule-based system that solves a specific problem: diagnosing and treating a specific type of breast cancer known as Invasive Ductal Carcinoma (IDC).

2.2. Work package

In the context of WP6 - "Symbolic AI and graph algorithmics" - the work of the team made up of researchers from the UDC and CESGA is focused on the fields of artificial intelligence, oncology and quantum computing.

We focus on a specific type of AI program, the so-called rule-based system (RBS). One of the biggest problems with RBS is the pattern-matching that is related to the process of rule selection. This makes RBS very sensitive to the size of the problem and to changing environments, which is at the core of a very significant increase in computational cost. In the field of artificial intelligence, the most difficult problems are informally known as **AI-complete** or **AI-hard**, implying that the difficulty of these computational problems, assuming intelligence is computational, is equivalent to that of solving the central artificial intelligence problem. To call a problem AI-complete reflects an attitude that it would not be solved by a simple specific algorithm. Current AI systems can solve very simple and/or restricted versions of AI-complete problems, but never in their full generality. When AI researchers attempt to "scale up" their systems to handle more complicated, real-world situations, the programs tend to become excessively brittle without commonsense knowledge or a rudimentary understanding of the situation: they fail as unexpected circumstances outside of its original problem context begin to appear. When human beings are dealing with new situations in the world, they are helped immensely by the fact that they know what to expect: they know what all things around them are, why they are there, what they are likely to do and so on. Knowledge-based systems are typically AI-hard.

We place special emphasis on the principle of coherent superposition, on the intrinsically parallel character of quantum computing, on the probabilistic nature of quantum computing, and we describe some quantum operators that must be reversible. Next, we build a quantum architecture equivalent to a conventional inferential circuit and analyze the results.

The algorithms and their implementation will be used to build a quantum rule-based system that solves a specific problem for the field of medicine to support the process of diagnosing and treating a specific type of breast cancer known as Invasive Ductal Carcinoma (IDC).

In this proposal we will describe a quantum method to represent the uncertainty that may appear in the so-called Quantum Rule-Based Systems (QRBS). For this we will consider the following restrictions:

1. We will consider that the rules of the knowledge base are written in a categorical way. For example: A and B → C
2. In the resolution of a problem the facts may be affected by imprecision and the rules of the knowledge base have to be used to obtain valid inferences. For example: we have the fact $\mathbb{A}$, which looks like A but it is not exactly A. We also have the fact $\mathbb{B}$, which looks like B but it is not exactly B. This statement means that, for example, instead of "something is absolutely true" it is "almost sure it is true". The question implies the ability of making inferences with facts of the kind $\mathbb{A}$ and $\mathbb{B}$, and with rules A and B → C.
3. Uncertainty arises as a consequence of the propagation of imprecision through the inferential network.

It is an obvious fact that uncertainty, in the most general sense possible and from any point of view, is a problem of the first magnitude - still unresolved - in the field of artificial intelligence and, more specifically, in Rule-Based Systems. There are lots of different terms and kinds of "representation of inexact knowledge": probability, subjective probability, imprecision, uncertainty, degrees of belief, fuzziness… and all of those terms refer to different concepts and are treated with different mathematical models. Regardless of the completeness of our system, the inherent subjectivity of the uncertainty associated with the information that is used when trying to solve a real case, has involved the development of a multitude of approaches and models to try to solve the problems of inexact knowledge representation and reasoning . The mathematical orientation of the different approaches varies depending on the model in question. Therefore, different models produce different results. In this context we can mention, among others:

1. Categorical approaches such as the so-called Differential Interpretation (Ledley & Lusted, 1959)
2. Probabilistic approaches such as Bayesian Networks (Pearl, 1986)
3. Quasi-statistical approaches such as the Certainty Factors Method (Shortliffe & Buchanan, 1975) or Evidential Theory (Shaffer, 1976)
4. Fuzzy methods such as Fuzzy Logic (Zadeh, 1965)

It appears to be clear that we cannot forget the potential of emerging theories and applications, among which Quantum Computing stands out and it is intrinsically probabilistic. **The question is, therefore, how can we model the subjective uncertainty** (here "uncertainty" refers to the strength of the causal relation between facts.) **of rule-based systems and achieve coherent results using the resources of quantum computing?** In short, it is about establishing synergies between artificial intelligence and quantum computing to solve the problem of uncertainty.

In this context, we will focus the question from the perspective of the Theory of Quantum Circuits. For this we will build the quantum operators equivalent to the classic {and, or} operators. We will also try to develop a classic inferential circuit and build the equivalent quantum circuit. To model the uncertainty we propose the use of projections of the corresponding quantum states along the Z axis of the Bloch sphere and, to model the subjectivity, the parameter DELTA (δ). This approach makes it possible the construction of a general quantum gate to represent any situation of uncertainty and express it in a probabilistic manner. The proposed approach will be widely evaluated and verified from different points of view. We expect that the results obtained after this exhaustive validation process allow us to conclude that Quantum Computation is an effective and efficient method to solve uncertainty problems in Artificial Intelligence.

3. Invasive Ductal Carcinoma (IDC)

Invasive ductal carcinoma, sometimes referred to as infiltrating ductal carcinoma, is the most common type of breast cancer. About 80% of all cases of breast cancer are invasive ductal carcinomas. Invasive means that the cancer has "invaded" or spread into the surrounding breast tissues. Ductal means that the cancer started in the milk ducts, which are the "pipes" that carry milk from the milk-producing lobules to the nipple. Carcinoma refers to any cancer that begins in the skin or in other tissues that line internal organs, such as breast tissue. Collectively, "invasive ductal carcinoma" refers to cancer that has penetrated the wall of the milk duct and has begun to invade the tissues of the breast. Over time, invasive ductal carcinoma can spread to the lymph nodes and possibly other areas of the body (Figure 1).

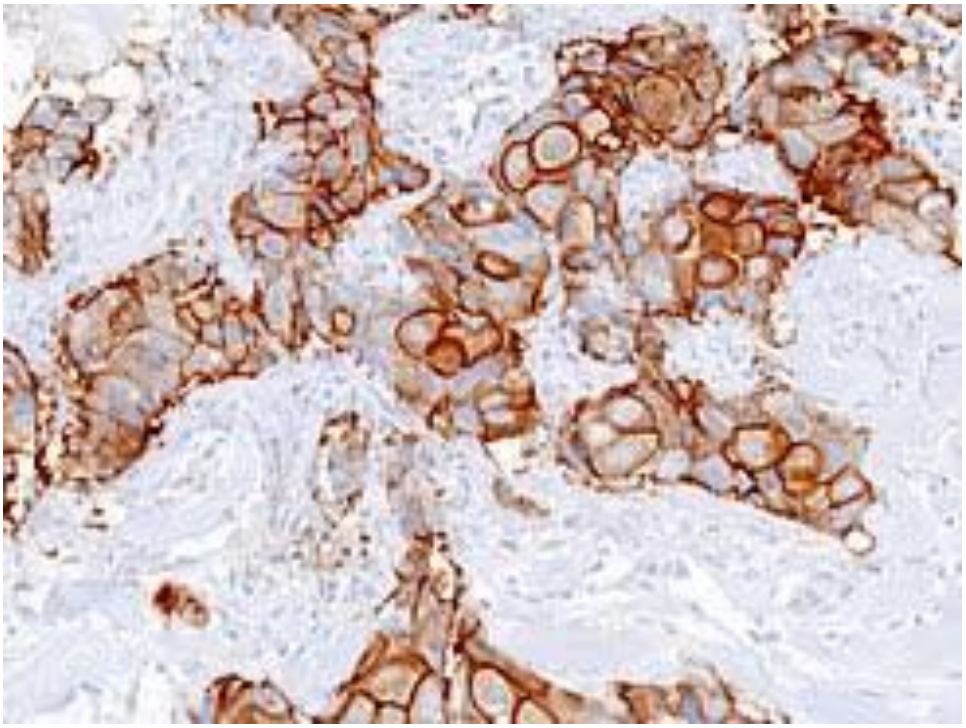

Figure 1: Invasive Ductal Carcinoma

Although this carcinoma can affect women of any age, it becomes more common as a woman ages. According to the American Cancer Society (American Cancer Society, 2019) approximately two-thirds of women who are diagnosed with invasive breast cancer are 55 years of age or older. Invasive ductal carcinoma also affects men.

3.1. Signs and Symptoms of Invasive Ductal Carcinoma

Invasive ductal carcinoma may not cause any symptoms at first. An abnormal area often appears on a screening mammogram (x-ray of the breast), leading to further testing. In some cases, the first sign of breast cancer is a recent lump or mass in the breast. According to the American Cancer Association (American Cancer Society, 2019) any of the following unusual changes in the breast can be an indication of breast cancer, including invasive ductal carcinoma:

- inflammation of the breast or part of it.
- skin irritation or pitting.
- breast pain, nipple pain or nipple inversion.
- redness, scaling, or thickening of the nipple or breast skin.
- a nipple discharge other than milk.
- a lump in the armpits.

3.2. Diagnosis of Invasive Ductal Carcinoma

The diagnosis of invasive ductal carcinoma usually includes a combination of procedures, physical examinations and imaging studies and PET.

- Physical examination of the breasts: The physicians may notice a small lump in the breast during the physical examination. They will also feel the lymph nodes under the armpit and above the collarbone to check for swelling or other abnormalities.
- Mammogram: Invasive ductal carcinoma is usually found with a mammogram, a test that takes X-ray images of the breast. Mammograms are performed in apparently healthy women to detect early signs of breast cancer. A key feature of invasive breast cancer is the presence of spiculated margins, which means that on the mammography the physician sees an abnormality that presents projecting finger-like projections. These projections reveal the "invasion" of the cancer into other tissues. If a screening mammogram highlights a suspicious area, additional mammograms are usually done to gather more information about the area in question. Mammography is done on both breasts.
- Ultrasounds: Sound waves from the breast to obtain additional images of the tissue. This study can be used to complement mammography.
- Breast MRI: MRI uses magnetic fields, radio waves, and a computer to generate images of tissues inside the body. In certain cases, the physician may use breast MRI to gather more information about a possibly affected area.
- Biopsy: If the results of a mammogram or other imaging test suggest an abnormality, the physician will probably want to do a biopsy. A biopsy involves removing some or all of the abnormal-looking tissue so that a pathologist (a doctor trained to diagnose cancer from biopsy samples) can look at it under a microscope.

3.3. Staging Invasive Ductal Carcinoma Severity

Staging is the process used to estimate the extent of invasive ductal carcinoma spread from its original location. The stage of the cancer is based on three pieces of information:

- the size of the tumor
- whether the cancer has spread to the lymph nodes and, if so, to what degree
- whether the cancer has spread to other parts of the body

Invasive ductal carcinoma is described on a scale from stage I (the earliest stage) to stage IV (the most advanced stage).

3.3.1. Stage I

Stage I describes invasive breast cancer (cancer cells take in or invade the normal breast tissue around them). Stage I is divided into subcategories, known as IA and IB.

Stage IA describes invasive breast cancer in which the tumor is up to 2 cm and the cancer has not spread beyond the breast; no lymph nodes are affected.

Stage IB describes invasive breast cancer in which: there is no tumor in the breast; in contrast, small groups of cancer cells greater than 0.2 mm but less than 2 mm are observed in the lymph nodes or there is a breast tumor smaller than 2 cm and small groups of cancer cells larger than 0.2 mm but smaller than 2 mm in the lymph nodes.

3.3.2. Stage II

Stage II is divided into subcategories IIA and IIB.

Stage IIA describes invasive breast cancer in which: there is no tumor in the breast, but cancer cells (larger than 2mm) are found in 1-3 axillary lymph nodes (under the arm) or in lymph nodes near the

breastbone (found during a sentinel node biopsy) or the tumor is 2 cm or smaller and has spread to the axillary lymph nodes or the tumor is 2 to 5 cm and has not spread to the axillary lymph nodes.

Stage IIB describes invasive breast cancer in which: the tumor is between 2 and 5 cm, and small groups of cancer cells larger than 0.2 mm but smaller than 2 mm are seen in the lymph nodes or the tumor is 2 to 5 cm, and the cancer has spread to 1-3 axillary lymph nodes or lymph nodes near the breastbone (found during a sentinel node biopsy) or the tumor is larger than 5 cm but has not spread to the axillary lymph nodes.

3.3.3. Stage III

Stage III is divided into subcategories IIIA, IIIB, and IIIC.

Stage IIIA describes invasive breast cancer in which: there is no tumor in the breast or the tumor may be any size, and cancer was found in 4-9 axillary lymph nodes or lymph nodes near the breastbone (found during imaging studies or physical examination) or the tumor is larger than 5 cm, and small clusters of cancer cells larger than 0.2 mm but smaller than 2 mm are seen in the lymph nodes or the tumor is larger than 5 cm, and the cancer has spread to 1-3 axillary lymph nodes or lymph nodes near the breastbone (found during a sentinel node biopsy).

Stage IIIB describes invasive breast cancer in which: the tumor is indefinite in size and has spread to the chest wall or skin of the breast, causing inflammation or an ulcer and may have spread to 9 axillary lymph nodes or may have spread to lymph nodes near the breastbone.

Stage IIIC describes invasive breast cancer in which: there may be no evidence of disease in the breast or, if a tumor is present, it may be any size and may have spread to the chest wall or skin of the breast and cancer has spread to 10 or more axillary lymph nodes or cancer has spread to lymph nodes above or below the collarbone or cancer has spread to axillary lymph nodes or lymph nodes near the breastbone.

3.3.4. Stage IV

Stage IV describes invasive breast cancer that has spread beyond the breast and surrounding lymph nodes to other organs in the body, such as the lungs, distant lymph nodes, skin, bones, liver, and brain.

3.4. Treatment for Invasive Ductal Carcinoma

Treatments for invasive ductal carcinoma fall into two broad categories:

- Localized treatments for IDC: surgery and radiation therapy. These treatments treat the tumor and surrounding areas, such as the chest and lymph nodes.
- Systemic treatments for IDC: chemotherapy, hormone therapy, targeted therapies. Drugs used in systemic treatments travel through the body to kill cancer cells that may have spread and help reduce the risk of cancer recurrence.

4. Modelling Rule-Based Systems

Knowledge engineering is a discipline that is a part of Artificial Intelligence whose purpose is the design and development of expert systems (Kendal & Creen, 2007). This is supported by instructional methodologies, trying to represent the human knowledge and reasoning in a certain domain, within an artificial system. The work of knowledge engineers consists of extracting the knowledge of human experts, and in coding said knowledge so that it can be processed by a system. The problem is that the knowledge engineer is not an expert in the field that tries to model, while the expert in the subject has no experience modeling his knowledge (based on heuristics) in a way that can be represented generically in a system. Knowledge engineering encompasses the scientists, technology and methodology required to process knowledge. The goal is to extract, articulate and computerize knowledge from an expert.

Since the task of acquiring knowledge is a difficult task, several stages have been identified. In this context, the development of a RBS has to be incremental and cyclic. This permits to face the RBS construction in a more systematic way (Nalepa, 2008). There are different methodologies for RBS development, but the most accepted in the literature includes the following steps:

- Identification of the problem.
- Conceptualization.
- Formalization.
- Implementation.
- Testing.

4.1. Identification of the problem

In this phase, it must first be determined whether the problem can or should be addressed through the AI techniques. To classify the problem as adequate, it cannot be solved algorithmically, since if it were possible in this way, it makes no sense to start such an expensive task. In fact, expert systems can be considered as “computational models of intelligent knowledge-based behaviour”. It must also be necessary to have access to sufficient sources of knowledge to complete a good design. Finally, the problem to be addressed must be of an adequate size so that it is not an unapproachable task due to its complexity.

The next step is to find the sources of knowledge that will be necessary for the development of the system, the most common are:

- Human experts in the domain of the problem.
- Books and manuals that explain the problem and resolution techniques.
- Examples of solved cases.

The latter will be important especially in the final validation phase, but can also be used with automatic knowledge acquisition techniques to obtain the basic elements involved and their relationships.

With these sources of information, necessary data to solve the problem and the criteria to evaluate the solution can be determined as well as the steps that allow the resolution and its subsequent evaluation.

At this time the knowledge engineer and the expert will be able to make a first description of the problem. The main goal will be to specify:

- Objectives.
- Motivations.
- Resolution strategies and their justification.
- Knowledge sources.
- Types of tasks.

This scheme will be the starting point to propose the following phases:

- Conceptualization.
- Formalization.
- Implementation.
- Testing.

4.2. Conceptualization

Before going into the global characteristics of the problem, it is essential to detail its basic elements and discover the relationships between them. In particular, it is necessary to observe how the expert solves typical problems and abstracts from them general principles that can be applied in different contexts. It is also necessary to obtain a decomposition of the problem into subproblems, performing an analysis by successive refinements until the knowledge engineer can get an idea of the hierarchical relationships of the different resolution phases to the most elementary reasoning operators.

Another required element is to discover the flow of reasoning in solving the problem and specify when and how knowledge items are needed. With this hierarchical decomposition and the flow of reasoning, the knowledge engineer can characterize the blocks of higher reasoning and the main concepts that define the problem. It will be mandatory to distinguish between evidence, hypotheses and necessary actions in each of the blocks and determine the difficulty of each one resolution subtasks. In this way it will be possible to capture the structure of the domain and the different relationships between its elements.

4.3. Formalization

We have to consider the different reasoning schemes that can be used to model the different problem solving needs identified in the previous phases. At this point, it is necessary to understand the nature of the search space and the type of search that will have to be done. For this, it can be compared with different prototypical mechanisms of resolution of problems such as classification, data abstraction, temporal reasoning, causal structures, etc.

At this stage, the certainty and completeness of the available information, the temporal dependencies, or the reliability and consistency of the information will also have to be analyzed. It should be discovered which parts of the knowledge are certain facts and which are not. For the latter, some methodology must be adapted for dealing with uncertainty, so that it can be modeled within the system (Lindley, 2014).

4.4. Implementation

At this point, decisions must be made about specifying the resolution and flow control of the information. Decisions must be made about the specific way of representing knowledge so that it can be adapted to the resolution strategies needed and the relationships between the different sets of knowledge. In this phase the rules will be defined, and inevitably problems and incompleteness will be discovered that will force a review of previous phases.

4.5. Testing

A set of representative resolved cases has to be chosen in order to check the operation of the system with these. In this phase, errors will be discovered that will allow correcting previous pitfalls; in general problems will appear due to lack of rules, incompleteness, lack of correction, and possible failures in the analysis of the pre-established rules.

4.6. Use case proposed methodology

We propose a concrete methodology for the acquisition of knowledge, in which we can distinguish the following phases:

- Initial.
- Methodological.
- Structured.
- Evaluation.

These four phases really constitute a complete knowledge engineering methodology - not just acquisition - since the final result obtained after its application should be a perfectly operational expert system.

- Scheme of the Initial Phase
  - Carrying out unstructured or directed interviews.
  - Obtaining examples for joint analysis by human experts and knowledge engineers.
  - Establishment of an initial set of rules.
- Scheme of the Methodological Phase
  - Macroscopic structuring of the knowledge obtained in the previous phase.
  - Global organization, and tentative, of the system under development.
  - Establishment of classes, and classification of all those information elements that share characteristics.
- Scheme of the Structured Phase
  - Organization and microscopic structuring of the information sets that share characteristics.
  - Design and construction of prototypes and individual modules.
  - Evaluation and refinement, separately, of each and every one of the prototypes and modules built.
- Scheme of the Evaluation Phase
  - Integration of modules.
  - Optimization of the control structures.
  - Adequacy of the interfaces and the mechanisms of explanation and justification.
  - Validation of the system in the laboratory.
  - Validation of the system in its real working environment.
  - Going back (when going back is no longer necessary, we will have obtained an operating expert system, ready for eventual commercialization.)

4.7. RBS Architecture

In computer science, RBS are used to store and manipulate knowledge to interpret information in a useful way. It is often used in artificial intelligence applications and research.

Normally, the term *rule-based system* is applied to systems involving human-crafted or curated rule sets. Rule-based systems constructed using automatic rule inference, such as rule-based machine learning, are normally excluded from this system type.

A typical rule-based system (Figure 2) has four basic components (Nalepa, 2008):

- **The list of rules,** which is a specific type of knowledge base.
- **The inference engine** or semantic reasoner, which infers information or takes action based on the interaction of input and the rule base. The interpreter executes a production system program by performing the following match-resolve-act cycle:
  - In the first phase, “matching”, the left-hand sides of all productions are matched against the contents of the “working memory”. The working memory is a module in which the declarative knowledge (initial information, proven facts, working hypothesis) that represents the actual state of the problem under consideration is stored. It interacts with the knowledge base to infer new facts. This is a dynamic process. When the working memory does not change then the inferential process ends. As a result of the matching process, a conflict set is obtained,

which consists of “instantiations” of all satisfied productions. An instantiation of a production is an ordered list of working memory elements that satisfies the left-hand side of the production.

- In the second phase, “conflict-resolution”, one of the production instantiations in the conflict set is chosen for execution. If no productions are satisfied, the interpreter halts.
- In the third phase, “action”, the actions of the production selected in the conflict-resolution phase are executed. These actions may change the contents of the working memory. At the end of this phase, execution returns to the first phase. Although the user/expert can solve the conflicts, there are specific and automatic ways for conflict resolution. This is usually linked with the search strategy, and can be implemented in the form of meta-rules. For example: IF we have two rules that can be fired THEN apply first the more specific one.

- **The temporary working memory**, that contains all the pieces of information the rule-based system is working with. It can hold both the premises and the conclusions of the rules.
- **The user interface** or other connection to the outside world through which input and output signals are received and sent.

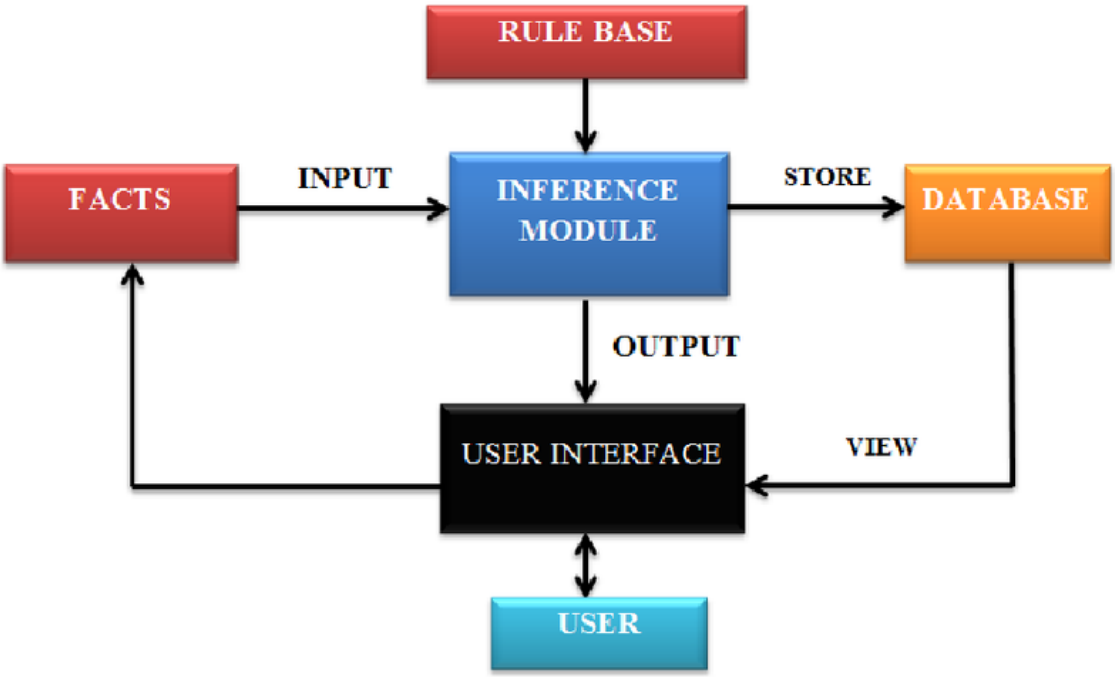

Figure 2: Typical architecture of a rule-based system

5. Quantum Rule-Based Systems

Quantum Rule-Based Systems (QRBS) are defined as those Rule-Based Systems (RBS) that use the formalism of Quantum Computing (QC) for representing knowledge and for making inferences.

5.1. General issues in QRBS

Let us consider the following set of rules:

a. R1: IF A and B THEN X
b. R2: IF X or C THEN Y
c. R3: IF Y and (D or E) THEN R

In conventional RBS, any categorical rule can be represented by the logical operators {and, or, not} that relate statements that are always true. Thus, rule R1 should be interpreted as follows: If statement A is true, and statement B is true, then we can conclude without uncertainty that statement X is true. The three previously defined rules can be represented classically by means of the inferential circuit of Figure 3. In any case, if we consider imprecision, these rules can be fired when, e.g. A=almost certain, B=probably yes and C=perhaps.

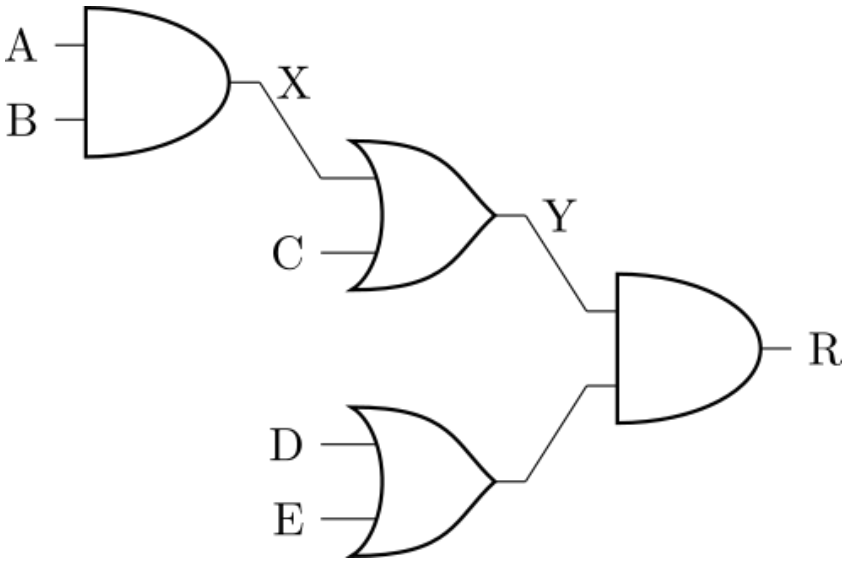

Figure 3: Classical representation of the inferential circuit of the example

However, if we choose the formalism of Quantum Computing we need reversible quantum gates to represent the previous inferential circuit.

In conventional RBS, any categorical rule can be represented by the logical operators {and, or, not} that relate statements that are always true. The truth tables of these conventional logical operators are the following (Table 1).

X	Y	not X	X and Y	X or Y
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Table 1: Truth tables of the classical logical operators {not, and, or}

In order to translate the classical RBS inferencial circuit of Figure 3 to a quantum environment, we need to rewrite the logical operators {not, and, or}, following the restrictions imposed by quantum computing. For that, we need reversible quantum gates such as the CNOT gate (Figure 4), or the Toffoli gate (Figure 5) also named CCNOT gate. Figure 6 shows how to build an {and} gate using a Toffoli gate. Figure 7 shows the same for the {not} gate and Figure 8 shows how to build an {or} gate using the Toffoli and the CNOT gates.

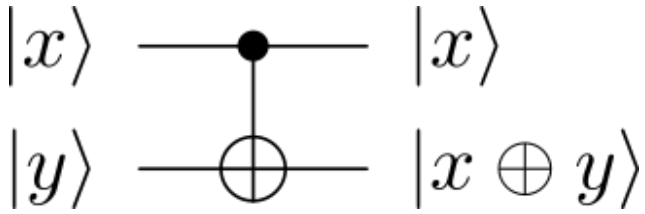

Figure 4: The CNOT gate

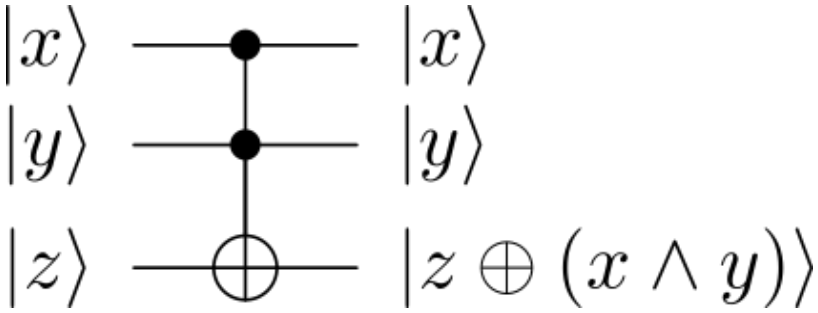

Figure 5: The CCNOT gate also named Toffoli gate

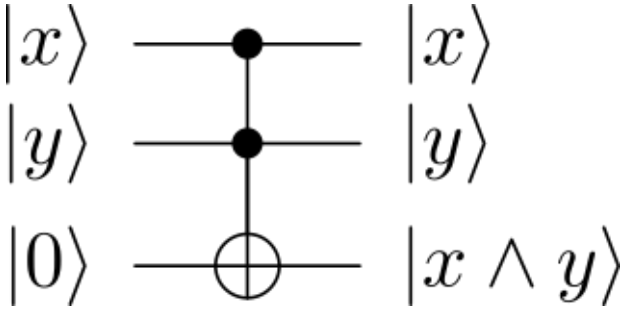

Figure 6: AND gate using the Toffoli gate

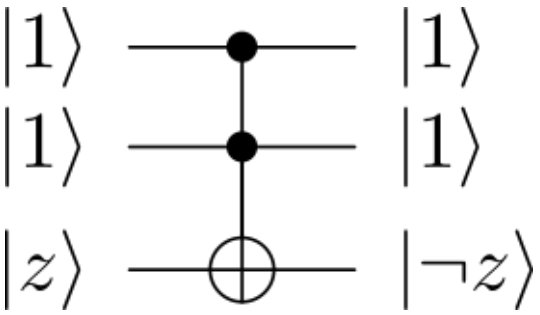

Figure 7: NOT gate using the Toffoli gate

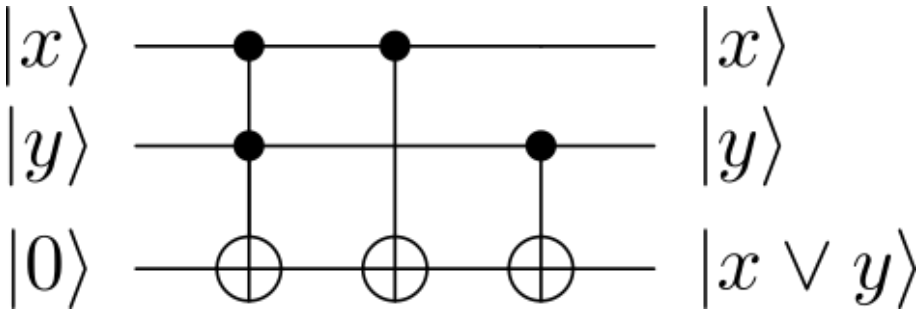

Figure 8: OR gate using the Toffoli and the CNOT gates

With these ideas in mind, we can simulate in a quantum manner, the quantum circuits that are equivalent to the classical logical operators.

For example, Figure 9 illustrates the quantum circuit of the quantum-AND. We have included two extra Hadamard gates to simulate the behaviour of this circuit with all possible input vectors. The idea is to obtain a homogeneous distribution of 0s and 1s in the input that permits to infer the Truth Tables of the classical logic operators {and, or}.

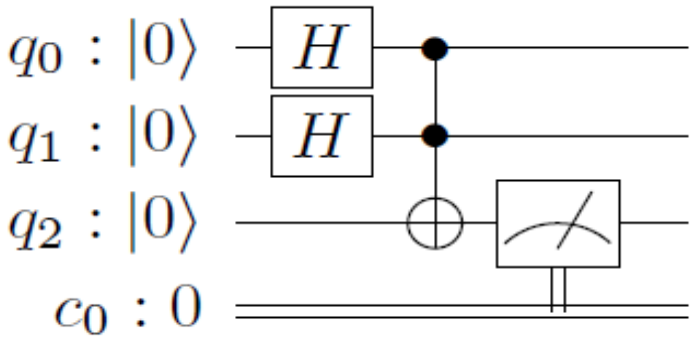

Figure 9: The Quantum-AND circuit

Table 2 illustrates the results of a simulation performed with both, classical-AND and quantum-AND, using the IBM Quantum Experience simulator (1048 shots.)

Input Vector	Input Truth Table	Output Truth Table	Measured Percentage	Estimated Percentage	Precision
000	00	0	25.0	25.0	1.000
010	01	0	24.8	25.0	0.992
100	10	0	24.9	25.0	0.996
111	11	1	25.3	25.0	0.988

Table 2: Classic-AND versus Quantum-AND

From the results shown in Table 2, it can be easily verified that after a few shots, the outputs are:

1. For output 0 = 25.0+24.8+24.9 which is very close to 75%
2. For output 1 = 25.3 which is close to 25 %

It appears to be clear that ANY classical inferential circuit can be represented as a quantum inferential circuit. More specifically, and in this case, the quantum representation of the circuit of Figure 3 would be that of Figure 10.

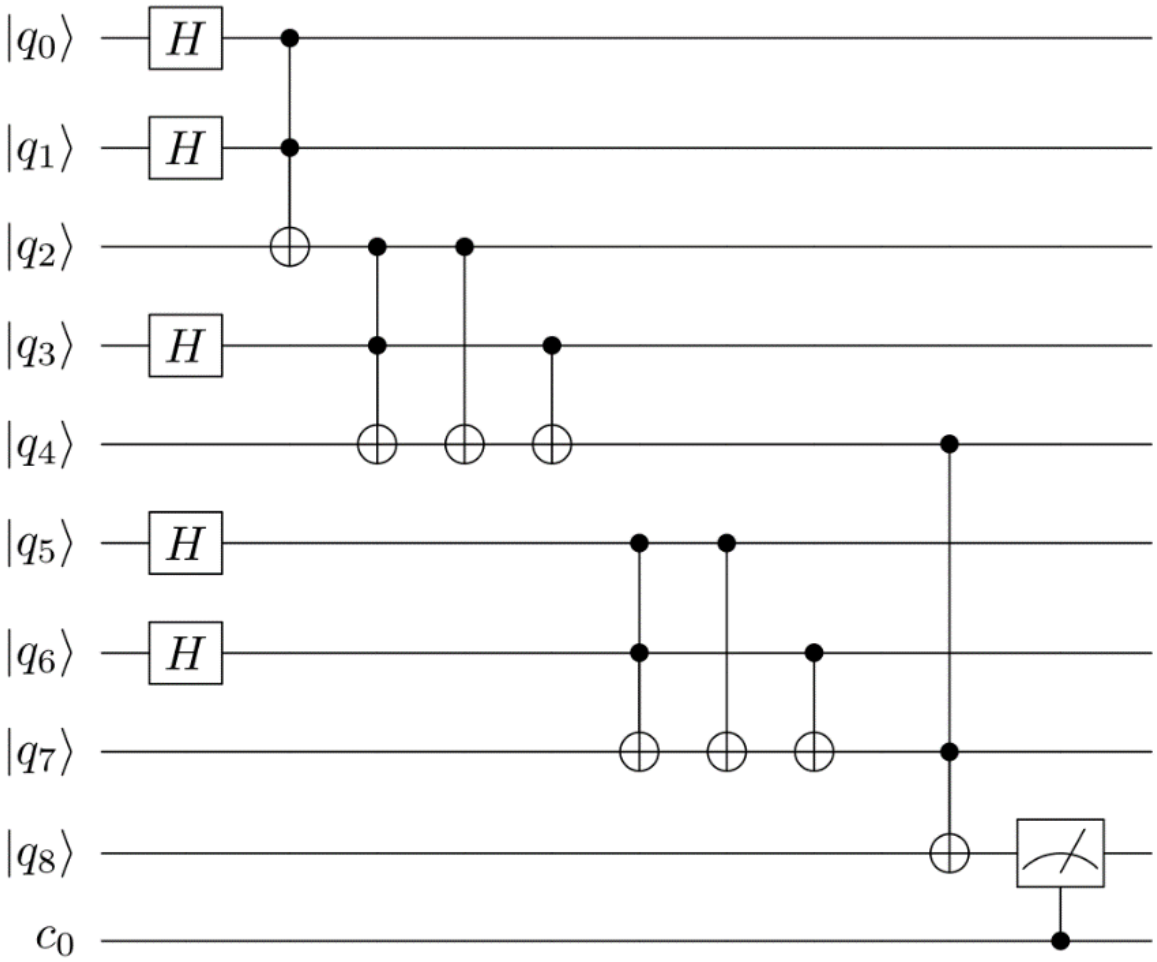

Figure 10: Quantum circuit representation of rules R1, R2 and R3

We will also focus on developing methods to present rules, their relations and inferential circuits in classical rule-based systems (RBS) using a set of reversible unitary quantum gates and quantum

circuits. The task will also be to develop a general and scalable methodology by using the concept of uncertainty in RBS which can be represented with a single unitary matrix in the QRBS.

5.2. Uncertainty in QRBS

Uncertainty is one of the fundamental problems of artificial intelligence (AI). In particular, it is one of the essential problems of the so-called Rule-Based Systems (RBS), and at the same time one of the most complex to deal with. Broadly speaking, we can consider that the origin of the uncertainty is related to one or several of the following causes (Lindley, 2014):

- It may happen that the information available is incomplete. In many cases the information available is not sufficient to make a categorical decision.
- Sometimes the available information we handle is wrong. Not always the information we manage is completely true.
- The information we use is usually imprecise. In many domains there are data that are difficult to quantify.
- Normally the real world is non-deterministic. Intelligent systems are not always governed by deterministic laws, so that general laws are not always applicable. Many times the same causes produce different effects without there being any apparent explanation.
- Our model is often incomplete. There are many phenomena whose cause is unknown. In addition, the lack of agreement between experts in the same field is frequent. Both circumstances make it difficult to include knowledge in a RBS.
- It may happen that, even if our model is complete, it contains inaccurate information.

Despite the large number of procedures that exist, any model that tries to quantify the uncertainty needs to include a large number of parameters. An example is the case of Bayesian networks (Pearl, 1986), in which we need to specify all a priori and conditional probabilities. However, a large part of this information is not usually available, so it must be estimated subjectively. In fact, subjective labels can be treated from different points of view. Consider the following example… fever is a symptom that is relevant when T > 37ºC. Now, say that we have a patient presenting T = 38ºC. It is obvious that the patient has fever. But the physician will pay different attention if the fever of the patient is moderate, high, or very high… question; is 38ºC very high fever, or moderate, or -even- significant? It depends on the context. This leads to a very interesting problem: The Symbolic Processing of Numeric Variables.

In summary, the treatment of inaccuracy in the inputs is, together with the representation of knowledge and learning, one of the fundamental problems of artificial intelligence (AI). In this context, and in a recent work, a quantum method to represent the uncertainty that may appear in the so-called QRBS is described (Moret-Bonillo, Emerging technologies in artificial intelligence: quantum rule-based systems, 2018).

5.2.1. Starting point

In quantum computing, the Bloch sphere is a geometric representation of the pure state space of a two-level quantum system. By extension, the set of pure states of an arbitrary finite number of levels is also usually called the Bloch sphere. In this case the Bloch sphere is no longer a sphere, but it has a geometric structure known as a symmetric space (Yanofsky & Mannucci, 2008). Consider Figure 11, which represents a Bloch sphere.

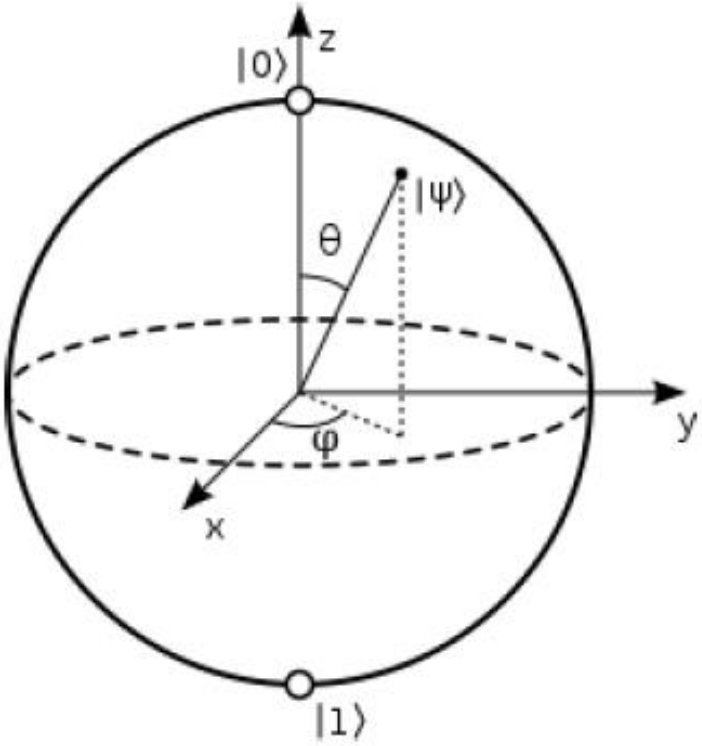

Figure 11: Schematic representation of a Bloch sphere

This sphere is important because it allows us to visualize the state of a QuBit. Note that it is a three-dimensional space in which the state of a QuBit $|\psi\rangle$ is represented by:

- The module, which is always 1. The meaning of this is that the QuBit is an energetically closed system.
- The angle θ, which represents the displacement of the QuBit along the Z axis, from the north pole to the south pole of the sphere.
- The angle ϕ, which represents the phase of the system that is inherent in complex vector spaces.

Given that the Bloch sphere locates $|0\rangle$ at the north pole, and since the direction of rotation on the Z axis is from top to bottom, we assume that a TRUE statement is represented by a $|0\rangle$, and a FALSE statement is represented by means of a $|1\rangle$.

5.2.2. Representing the Uncertainty in QRBS

The question we are now asking is the following: assumed a given Quantum Rule-Based System, *is there any quantum property that allows to represent the imprecision associated with the facts and the uncertainty associated with the rules?*

To try to answer the previous question let's look at the angle θ of the Bloch sphere.

- When $\theta = 0$ radians → $|0\rangle$ → The associated statement is true.
- When $\theta = \pi$ radians → $|1\rangle$ → The associated declaration is false.
- When $0 < \theta < \pi$ → Both statements are in superposition, and the associated statement is neither true nor false, or - in an equivalent way - it is true and it is false simultaneously.

The election of the angle is due to the fact that considering the Bloch sphere, the location of |0> and the location of |1>, the imprecision could be considered as a projection of the input state |X> on the Z-axis.

For practical reasons we will call “Credibility” to our confidence in a given fact. In such way that:

- Credibility = 100 → The fact is true
- Credibility = 0 → The fact is false

We also relate the concept of credibility with the concept of “Degree of Disbelief” associated to a given fact. The relation between these two concepts is as follows:

- Credibility = 100 – Disbelief

For the reasons just explained, the quantification of Z displacements in a Bloch sphere could be used to quantify our credibility associated to a given fact.

To define a general procedure capable of representing any degree of uncertainty (or certainty) it would be convenient to have a single quantum gate that, of course respecting all the restrictions imposed by quantum mechanics, brings us closer to analog world. In this context there are already several universal gates, but none of them explicitly works with imprecise information in the domain of artificial intelligence. In this regard, and taking into account what has been described so far, our proposal is as follows:

Let DELTA (δ) be the degree of subjective disbelief that we can associate with a fact in a rule-based system. It is trivial that the parameter δ can be converted into an ALPHA angle (α) that satisfies the restrictions of Z displacements. Now suppose that our subjective disbelief δ is defined in the closed interval [0, 100]. Obviously:

- If δ = 0 → Our credibility in the fact is total → The fact is true
- If δ = 100 → Our credibility in the negation of the fact is total → The fact is false
- If 0 < δ < 100 → There is subjective disbelief in the fact under consideration

The following equation establishes the correspondence between δ and α, so that δ is compatible with the concept of subjective disbelief, and α is compatible with the restrictions imposed by the Bloch sphere:

- $\alpha = \frac{\pi \times \delta}{100}$ radians

Now let us define THETA (θ) = (π − α) / 2 as the angle of rotation, or displacement, in Z. Table 3 illustrates the values of ALPHA (α) - in degrees and in radians - as a function of the values of DELTA (δ) -defined in the interval [0, 100], and the corresponding values of THETA (θ) - expressed in radians.

DELTA (Subjective Disbelief)	ALPHA (Degrees)	ALPHA (Radians)	THETA (Radians)
0	0	0	π/2
25	45	π/4	3π/8
50	90	π/2	π/4
75	135	3π/4	π/8
100	180	π	0

Table 3: Correspondence between the values of the parameters DELTA, ALPHA and THETA

We will now define, based on the angle θ, the following Matrix:

$$M(\theta) = \begin{pmatrix} \sin(\theta) & \cos(\theta) \\ \cos(\theta) & -\sin(\theta) \end{pmatrix}$$

This matrix verifies that:

$$\begin{pmatrix} \sin(\theta) & \cos(\theta) \\ \cos(\theta) & -\sin(\theta) \end{pmatrix} \times \begin{pmatrix} \sin(\theta) & \cos(\theta) \\ \cos(\theta) & -\sin(\theta) \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

In this context:

- If α = 0 → M (θ) |0⟩ = M (π / 2) |0⟩ = |0⟩
- If α = π / 4 → M (θ) |0⟩ = M (3π / 8) |0⟩ = 0.924 |0⟩ + 0.383 |1⟩
- If α = π / 2 → M (θ) |0⟩ = M (π / 4) |0⟩ = 0.707 |0⟩ + 0.707 |1⟩
- If α = 3π / 4 → M (θ) |0⟩ = M (π / 8) |0⟩ = 0.383 |0⟩ + 0.924 |1⟩
- If α = π → M (θ) |0⟩ = M (0) |0⟩ = |1⟩

Obviously:

- If α = 0 → there is no disbelief in the fact, the credibility is total and the fact is true
- If α = π → there is no disbelief in the negation of the fact, the credibility is none and the fact is false

Uncertainty appears when ALPHA is between 0 and π.

In Appendix A we can see how to implement the quantum AND and OR gates and the M gate using myQLM. We also implemented the inferential circuit of section 5.1 using that gates and executed a simulation using the PyLinalg QPU (Atos, 2016-2020).

6. Formal Specification Approach

Formal Specification Approach Critical Systems Development usually involves a plan-based software process. Both the system requirements and the system design are expressed in detail and carefully analysed and checked before implementation begins.

If a formal specification of the software is developed, this usually comes after the system requirements have been specified but before the detailed system design. There is a tight feedback loop between the detailed requirements specification and the formal specification. One of the main benefits of formal specification is its ability to uncover problems and ambiguities in the system requirements. Figure 12 shows specification and design activities may be carried out in parallel streams.

There is a two-way relationship between each stage in the process. Information is fed from the specification to the design process and vice versa.

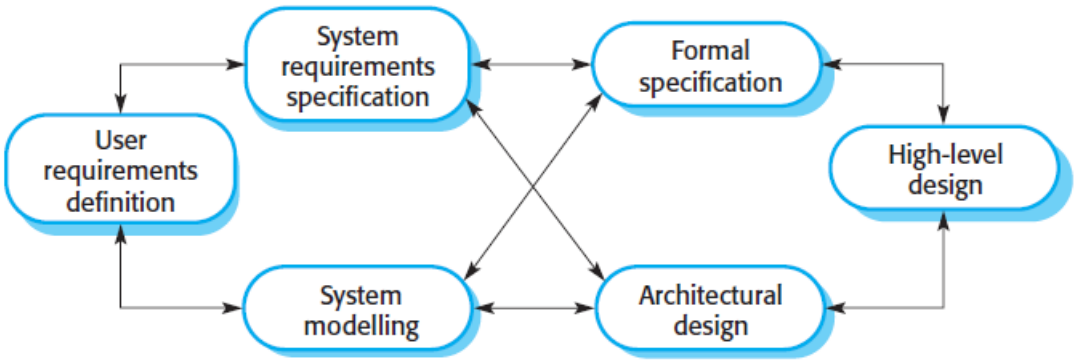

Figure 12: Formal specification in the software process

Two fundamental approaches to formal specification can be used to write detailed specifications for software systems. These are:

- Algebraic approaches where the system is described in terms of operations and their relationships.
- Model-based approaches where a model of the system is built using mathematical constructs such as sets and sequences and the system operations are defined by how they modify the system state.

We will focus on the algebraic approaches. The algebraic method of formal specification defines the abstract data type in terms of the relationships between the type operations. Figure 13 illustrates the structure of an algebraic specification.

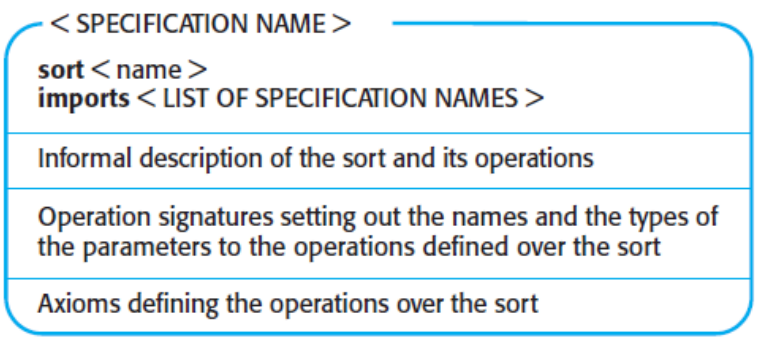

Figure 13: Structure of an algebraic specification

The body of the specification has four components:

- The introduction that declares the sort (the type name) of the entity being specified. A sort is the name of a set of objects with common characteristics. It is similar to a type in a programming language. The introduction may also include an "imports" declaration, where the names of specifications defining other sorts are declared. Importing a specification makes these sorts available for use.
- The description part, where the operations are described informally. This makes the formal specification easier to understand. The formal specification complements this description by providing an unambiguous syntax and semantics for the type operations.
- The signature part defines the syntax of the interface to the object class or abstract data type. The names of the operations that are defined, the number and sorts of their parameters, and the sort of operation results are described in the signature.
- The axioms part defines the semantics of the operations by defining a set of axioms that characterize the behaviour of the abstract data type. These axioms relate the operations used to construct entities of the defined sort with operations used to inspect its values.

We will follow these specifications but with minor changes to improve readability, that is, we will put the informal description of each operation and the axioms next to the operation signatures. And the axioms will be mainly focused in specifying pre and post conditions of each operation.

6.1.1. Formal specification of facts and the working memory

As said, the basic element of a RBS is a *fact*. It can be defined as the smallest unit of information that can be separately added to or removed from the working memory of a rule-based system.

The simplest way to define a fact is to use attribute-value pairs, that is, we define an attribute, and we define the value that this attribute takes. The attribute will be the identifier of the fact while the value will be a literal belonging to some existing data type. The simplest facts are Boolean facts, that is, an attribute and a boolean value indicating if the attribute is true or false: *(rain true)*. This can be extended to other data types (strings, numeric values, enumerated values, etc.) to obtain different types of facts: *(name "Bob Smith") (age 34) (gender Male).*

In complex systems, fact can be grouped to form complex structures, e.g., we can use the previous facts to form the person fact like this "*(person (name "Bob Smith") (age 34) (gender Male))*" but in this document we will try to keep things simple, so will assume that a fact is simply an attribute-value pair.

In Table 4 we can see the formal specification of a fact. For each element that we will include in this formal specification a description of the element is also added (showing the name and an informal description) and the relevant operations. For each operation we will consider the following elements. If any of them is not necessary (a function without inputs, for example) then it won't be included in the table:

- Signature. With the inputs and outputs separated by an arrow "→". If the operation has no return value then simply the output is not represented (nor the arrow).
- Objective. Informal description of the objective of the operation.
- Input values.
- Output values.
- Preconditions. Conditions that must be true in order for the operation to be executed.
- Postconditions. Conditions that will be true upon completion of the operation.

The operations depicted here are the minimum operations needed. A given implementation probably would include more operations, in order to make it easier to work with the different elements.

Element		
Element	**fact**	
Description	Represents a working memory element composed by a pair attribute-value.	
Operations		
Signature	**fact(attribute, value) → fact**	
Objective	Creates a new fact based on an attribute and a value.	
Inputs	attribute	Represented by a valid identifier.
	value	Represented as a literal value of one of the accepted data types: boolean, integer, string, enumeration, etc.
Outputs	A fact element.	
Preconditions	The identifier must be valid. The literal value must be a valid value of the corresponding data type.	
Postconditions	A new fact is created as an attribute-value pair.	

Table 4: Formal specification of a fact.

Since facts are defined as working memory elements, it would now be appropriate to define how the working memory works. To do so, we will create a new element, called working memory, which is responsible for asserting and retracting facts. Its formal specification is included in Table 5.

Element		
Element	**working_memory**	
Description	Element that contains all the pieces of information the rule-based system is working with.	
Operations		
Signature	**working_memory() → working_memory**	
Objective	Creates a new *working_memory.*	
Outputs	A working_memory element.	
Postconditions	A new empty working_memory is created.	
Signature	**assert(fact)**	
Objective	Adds facts to the working memory.	
Inputs	fact	A fact element.
Preconditions	The fact is valid.	
Postconditions	The given fact is added to the working_memory. If a fact with the same attribute was already present in the working memory that fact is retracted before this new fact is inserted.	
Signature	**retract(fact)**	
Objective	Removes facts from the working memory.	
Inputs	fact	A fact element.
Preconditions	The fact is valid.	
Postconditions	The given fact is removed from the working_memory.	
Signature	**reset()**	
Objective	Initializes the working memory removing all the facts from it.	
Postconditions	The working_memory is empty.	

Table 5: Formal specification of a working memory element.

6.1.2. Formal specification of rules and the inference engine

After defining facts, we must define rules in more detail. Rules are like if-then statements in procedural languages, but they are not used in a procedural way. Whereas if-then statements are executed at a specific time and in a specific order, according to how the programmer writes them, a rule's "then" part can be executed whenever the "if" part is satisfied. This makes rules less obviously deterministic than a typical procedural program because the inference engine decides the order in which they are fired.

Rules are composed by a left-hand side (LHS), that is the premise, and a right-hand side (RHS), that is the conclusion. Before defining a rule, we must define in detail those parts.

The left-hand side of a rule (the "if" part) consists of patterns that match facts; they are not function calls. Therefore, a LHS is composed of three parts, a fact, an operator, and a value or literal. The simplest LHS is that formed by an attribute, and equality operator (*is*) and a boolean value. For example *(rain is true)*. With other data types we can uses other operators, for example, with numeric values we can have the following: less than (*lt*), less or equal than (*le*), greater than (*gt*), greater or equal than (*ge*), etc.

Also in the LHS we have to take into account the connective constraints. Quite often, matching with a literal value or a variable is not enough. Therefore, it is usual to add logic connective constraints to the LHS part of a rule, that is, the "and", "or" and "not" clauses. This way we can construct a LHS part of a rule that is a composite of LHS clauses connected by logic operators: "*(rain is true) and (outside is true)*".

The formal specification of a LHS element is included in Table 6. It includes three generative operations to create a LHS element. There is a primitive operation that allows to create a LHS element from a fact, an operator and a value; and there are also included two other operations to allow the creation of LHS elements that are a result of applying the logical clauses to already existing LHS elements.

Element		
Element	**lhs**	
Description	Represents the left-hand side (LHS) part of a rule.	
Operations		
Signature	**lhs(fact, operator, value) → lhs**	
Objective	Creates a new lhs element.	
Inputs	fact	A fact element.
	operator	An operator of a given data type.
	value	A value of a given data type.
Outputs	A lhs element.	
Preconditions	The fact is valid. The operator is defined in the value data type.	
Postconditions	A new lhs elements is created.	
Signature	**lhs(lhs, logical_operator, lhs) → lhs**	
Objective	Creates a new lhs element combining two previously created lhs elements with a binary logical operator.	
Inputs	Lhs	A lhs element.
	logical_operator	A binary logical operator to perform a logical operation.
	Lhs	A lhs element.
Outputs	A lhs element.	
Preconditions	The lhs elements are valid. The binary logical operator is "*or*" or "*and*".	
Postconditions	A new lhs elements is created that is the result of a logical combination of two previous lhs elements.	

Signature	**lhs(logical_operator, lhs) → lhs**	
Objective	Creates a new lhs element combining applying an unary logical operator to a previously created lhs element.	
Inputs	logical_operator	An unary logical operator.
	Lhs	A lhs element.
Outputs	A lhs element.	
Preconditions	The unary logical operator is “not”. The lhs element is valid.	
Postconditions	A new lhs elements is created that is the result of applying an unary logical operator to a previous lhs element.	

Table 6: Formal specification of a LHS element.

The right-hand side of a rule (RHS), on the other hand, is made up of function calls and their parameters. The simplest function call is an assignment function (set) that has two arguments, a fact, and a value. The function “set” assigns a value to a fact. If the fact is already in the working memory its value changes according with the RHS, if not, it is asserted in the working memory with the value specified. That is, “set” is calling the assert operation in the working memory. For example, a RHS can be (set wet_floor true) and therefore we have the following rule: if (rain is true) => (set wet_floor true). Table 7 shows the formal specification of a RHS element.

Element		
Element	**rhs**	
Description	Represents the right-hand side (RHS) part of a rule.	
Operations		
Signature	**rhs(function, parameters, …) → rhs**	
Objective	Creates a new rhs element.	
Inputs	function	A function.
	parameters	A list of parameters for the function separated by commas. If the function has no parameters it can be empty.
Outputs	A rhs element.	
Preconditions	The function is valid and the parameters are valid for that function.	
Postconditions	A new rhs elements is created.	

Table 7: Formal specification of a RHS element.

Once we have defined a LHS and a RHS it is easy to define a rule as a combination of them (Table 8).

Element		
Element	**rule**	
Description	Represents a rule formed by a LHS and a RHS.	
Operations		
Signature	**rule(lhs, rhs) → rule**	
Objective	Creates a new rule element.	
Inputs	lhs	A lhs element.
	rhs	A rhs element.
Outputs	A rule element.	
Preconditions	The input elements (lhs and rhs) are well formed.	
Postconditions	A new rule elements is created.	

Table 8: Formal specification of a rule element.

Finally, once we have defined the rules, we must define the specification of the inference engine. This engine controls the whole process of applying the rules to the working memory in order to obtain the outputs of the system.

In previous sections we have seen that the inference engine has three phases of functioning: *match*, *conflict resolution* and *act*. Let us describe in more detail the elements of the inference engine that perform these phases to better comprehend how this engine works internally:

- Pattern matcher: The match phase is performed by the patter matcher. The purpose of the pattern matcher is to decide which rules apply, given the current contents of the working memory. This is the most expensive part of the process or a rule-based system the pattern since the matcher might need to search through millions of combinations of facts to find those combinations that satisfy rules. If we are dealing with inaccurate information we can consider the following example: Rule: IF A is true THEN B is possible. Fact: A is almost true. Matcher: A is present in the rule and it is also a fact with a given degree of belief. Inferential Generalized Modus Ponens: B could be possible.
- Agenda: The agenda is in charge of the conflict resolution phase. The agenda stores the list of rules that could potentially be fired (the conflict set) and is responsible for using the conflict strategy to decide which of the rules, out of all those that apply, have the highest priority, and should be fired first.
- Execution engine: Finally, the execution engine is in charge of the act phase. Once the agenda has decided what rule to fire, it has to execute that rule's action part. Therefore, the execution engine is the component of a rule engine that fires the rules.

These elements are the active part of the inference engine, who are in charge of making it work. There are also passive elements that store the elements needed by these active parts. These passive elements are:

- Working memory: stores the facts that the inference engine uses to work with.
- Rule base: list of rules that composes the knowledge adquired by the system.

Since we are going to describe the upper level of abstraction of a rule engine, we do not need to formally specify all these internal parts, but it is useful to know their existence for formalizing the rule engine itself. We only have described in detail the working memory because it is directly related with the facts. In Table 9 we can see the formal specification of a rule engine.

Element		
Element	**rule_engine**	
Description	Represents a rule engine element.	
Operations		
Signature	**rule_engine() → rule_engine**	
Objective	Creates a new rule_engine with all its internal elementes (pattern matcher, agenda, execution engine, rule base and working memory).	
Outputs	A rule_engine element.	
Postconditions	A new empty rule_engine is created.	
Signature	**add_rule(rule)**	
Objective	Adds a new rule to the rule base.	
Inputs	rule	A rule element.
Preconditions	The rule is valid.	
Postconditions	The rule is added to the rule base.	
Signature	**delete_rule(rule)**	
Objective	Deletes a given rule from the rule base.	
Inputs	rule	A rule element.
Preconditions	The rule is valid.	
Postconditions	Removes the given rule form the rule_base.	

Signature	**pattern_match()**
Objective	Performs a pattern matching between the facts of the working memory and the rules of the rule base. The result is a list of activated rules that go into the agenda.
Preconditions	The rule base has at least one rule.
Postconditions	The agenda of the inference engine is filled with the activated rules given the facts stored in the working memory.
Signature	**fire_rule()**
Objective	Fires the rule with the higher priority in the agenda, modifying the working memory if necessary.
Preconditions	The agenda has at least one rule activated.
Postconditions	The rule is fired executing the RHS action as specified.
Signature	**reset_engine()**
Objective	Resets the rule engine to its initial state emptying the agenda and working memory.
Postconditions	The agenda and the working memory are empty.

Table 9: Formal specification of a rule engine.

6.1.3. Functioning of the inference engine

In the previous sections we have described the formal specification of facts, rules, the working memory and the rule engine, as they are the basic elements of a inferential system based on rules. Nevertheless, we are not describing how these elements should be used, as that is a job for another element, the inferential reasoner, a much high level element that decides the inferential strategy that it is needed to follow.

Since the inferential reasoner is a high level element whose functioning strongly depends on the characteristics of the domain and of the problem, we will only briefly describe here how it would work. There are two main strategies for the inferential reasoner to follow: forward chaining and backward chaining:

- Forward chaining: is the main strategy followed by an inferential reasoner based on rules. The functioning is as described previously, the rule engine matches the LHS part of the rules with the working memory and fires one of the rules activated by the facts. It is a forward strategy from facts to conclusions with the aim to reach a final conclusion to our problem.
- Backward chaining: works in the opposite way as forward chaining, it is a backward strategy from conclusions to facts following a behavior called goal seeking. A backward-chaining reasoner starts with a final conclusion and actively tries to satisfy the LHS conditions of the rules pointing to that conclusion. If the reasoner determines that, for satisfying those LHS conditions, it needs to fire some other rules, then it focuses in trying to fullfill the LHS conditions of those new rules.

It is also possible to use a combination of the two chaining methods to solve the same problem, changing the strategy as needed during the execution of the rules.

In Figure 14 we can see an example of how it would be the functioning of a inferential reasoner using forward chaining and the formal specification described previously. We are working with the same example described in section 5.1.

```
# Creation of the working memory and the rule engine
wm = working_memory()
re = rule_engine()

# Definition of rules
r1 = rule(lhs(lhs(A, is, true), and, lhs(B, is, true)), rhs(set, X, true))
r2 = rule(lhs(lhs(X, is, true), or, lhs(C, is, true)), rhs(set, Y, true))
r3 = rule(lhs(lhs(Y, is, true), and, lhs(lhs(D, is, true), or, lhs(E, is, true))),
          rhs(set, R, true))

# Assert facts into the working memory
wm.assert(A, true)
wm.assert(B, false)
wm.assert(C, true)
wm.assert(D, true)
wm.assert(E, false)

# Follow a forward chaining strategy
do {
   re.pattern_match()
   re.fire_rule()
} while (agenda has rules)
```

Figure 14: forward chaining strategy of an inferential reasoner using the formal specification described.

6.1.4. Uncertainty management

Finally, we left out the uncertainty from the formal specification because we wanted to formally describe the basic functioning of a RBS. But uncertainty exists in the real world and it is something that we want to include in our system, and also take advantage of the probabilistic nature of QuBits for representing this uncertainty. For including uncertainty in our model we have to modify the fact specification to include a new parameter in his generative function, the degree of uncertainty over that fact, as we can see in Table 10.

Element		
Element	**Fact**	
Description	Represents a working memory element composed by a pair attribute-value affected by a degree of uncertainty.	
Operations		
Signature	**fact(attribute, value, uncertainty) → fact**	
Objective	Creates a new fact based on an attribute and a value.	
Inputs	attribute	Represented by a valid identifier.
	value	Represented as a literal value of one of the accepted data types: boolean, integer, string, enumeration, etc.
	uncertainty	Degree of uncertainty of that attribute having that value.
Outputs	A fact element.	
Preconditions	The identifier must be valid. The literal value must be a valid value of the corresponding data type.	
Postconditions	A new fact is created as an attribute-value pair with a degree of uncertainty.	

Table 10: Formal specification of a fact with uncertainty.

Also, we must include the uncertainty in the functioning of the rule engine, for that reason we must modify the pattern_match() and the fire_rule() functions as we can see in Table 11, not changing their interfaces but their internal working.

Element	
Element	**quantum_rule_engine**
Description	Represents a working memory element composed by a pair attribute-value.
Operations	
Signature	**pattern_match()**
Objective	Performs a pattern matching between the facts of the working memory and the rules of the rule base. The result is a list of activated rules that go into the agenda. The pattern match has to take into account that the fact can be affected of uncertainty and thus be partially true (or false).
Preconditions	The rule base has at least one rule.
Postconditions	The agenda of the inference engine is filled with the activated rules given the facts stored in the working memory.
Signature	**fire_rule()**
Objective	Fires the rule with the higher priority in the agenda, modifying the working memory if necessary.
Preconditions	The agenda has at least one rule activated.
Postconditions	The rule is fired executing the RHS action as specified. The uncertainty of the facts is taken into account in order to calculate the uncertainty of the conclusions.

Table 11: Formal specification of a rule engine with uncertainty.

As there may be different approaches on how to combine the uncertainty of the facts into the uncertainty of the conclusions, and even the rules themselves may be affected by uncertainty, it has been preferred in this formal specification not to go into such low-level details, as they are dependent on the particular implementation we are considering.

7. Acronyms and Abbreviations

Term	Definition
IDC	Invasive Ductal Carcinoma
QRBS	Quantum Rule-Based Systems
RBS	Rule-Based Systems

Table 12: Acronyms and Abbreviations

8. List of Figures

9. List of Tables

10. Bibliography

American Cancer Society. (2019). Breast Cancer Facts & Figures 2019-2020. *Atlanta: American Cancer Society, Inc.*

Atos. (2016-2020). *qat.pylinalg: Python Linear-algebra simulator*. Obtenido de https://myqlm.github.io/myqlm_specific/qat-pylinalg.html

Commission Européenne, 7. C. (10 de 2016). *Grant Agreement.* Obtenido de https://shirocommunity.bull.com/ext/fpop/clouddbappliance/shareddocuments/GRANT/Grant%20Agreement-732051-CloudDBAppliance.pdf

Kendal, S. L., & Creen, M. (2007). *An Introduction to Knowledge Engineering.* Springer.

Ledley, R. S., & Lusted, L. B. (1959). Reasoning Foundations of Medial Diagnosis. *Science*, 9-21.

Lindley, D. V. (2014). *Understanding Uncertainty, Revised Edition.* John Wiley & Sons Ltd.

Moret-Bonillo, V. (2000). *Fundamentos de la Inteligencia Artificial.* Universidade da Coruña.

Moret-Bonillo, V. (2018). Emerging technologies in artificial intelligence: quantum rule-based systems. *Progress in Artificial Intelligence 7*, 155-166.

Nalepa, G. J. (2008). *Methodologies and Technologies for Rule-Based Systems Design and Implementation. Towards Hybrid Knowledge Engineering.* Springer.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence, Volume 29, Issue 3*, 241-288.

Project, 7. C. (07 de 11 de 2016). *Consortium Agreement.* Obtenido de https://shirocommunity.bull.com/ext/fpop/clouddbappliance/shareddocuments/Consortium%20Agreement/DESCA%20CloudDBAppliance.Final.2016-11-07.pdf

Shaffer, G. (1976). *A Mathematical Theory of Evidence.* Princeton University Press.

Shortliffe, E. H., & Buchanan, B. G. (1975). A model of inexact reasoning in medicine. *Mathematical biosciences*, 351-379.

Yanofsky, N. S., & Mannucci, M. A. (2008). *Quantum Computing for Computer Scientists.* Cambridge University Press.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control, Volume 8, Issue 3*, 338-353.

Appendix A. Examples and simulations

The defined quantum gates, as well as the example circuit and the M gate, have been implemented and simulated using myQLM. The figures below only show the lines of code that define the circuits and their gates, and they all have been simulated using the PyLinalg QPU.

A.1 Simulation of the Q-AND gate in myQLM

```
# Q-AND Gate
prog = Program()
qbits = prog.qalloc(3)
prog.apply(H, qbits[0])
prog.apply(H, qbits[1])
prog.apply(CCNOT, qbits[0],qbits[1],qbits[2])
```

Figure 15: myQLM implementation of the Q-AND gate

```
State |000> probability=0.2499999999999999
State |010> probability=0.2499999999999999
State |100> probability=0.2499999999999999
State |111> probability=0.2499999999999999
```

Figure 16: Results of the simulation of the Q-AND gate

A.2 Simulation of the Q-OR gate in myQLM

```
# Q-OR Gate
prog = Program()
qbits = prog.qalloc(3)
prog.apply(H, qbits[0])
prog.apply(H, qbits[1])
prog.apply(CCNOT, qbits[0],qbits[1],qbits[2])
prog.apply(CNOT, qbits[0], qbits[2])
prog.apply(CNOT, qbits[1], qbits[2])
```

Figure 17: myQLM implementation of the Q-OR gate

```
State |000> probability=0.2499999999999999
State |011> probability=0.2499999999999999
State |101> probability=0.2499999999999999
State |111> probability=0.2499999999999999
```

Figure 18: Results of the simulation of the Q-OR gate

A.3 Simulation of the example circuit

```
# Quantum circuit
prog = Program()
qbits = prog.qalloc(9)

# A(q0) and B(q1) then X(q2)
prog.apply(H, qbits[0])
prog.apply(H, qbits[1])
prog.apply(CCNOT, qbits[0], qbits[1], qbits[2])

# X(q2) or C(q3) then Y(q4)
prog.apply(H, qbits[3])
prog.apply(CCNOT, qbits[2], qbits[3], qbits[4])
prog.apply(CNOT, qbits[2], qbits[4])
prog.apply(CNOT, qbits[3], qbits[4])

# Y(q4) and (D(q5) or E(q6)) then R(q8)
prog.apply(H, qbits[5])
prog.apply(H, qbits[6])
prog.apply(CCNOT, qbits[5], qbits[6], qbits[7])
prog.apply(CNOT, qbits[5], qbits[7])
prog.apply(CNOT, qbits[6], qbits[7])
prog.apply(CCNOT, qbits[4], qbits[7], qbits[8])
```

Figure 19: myQLM implementation of the example circuit

```
State |000000000> probability=0.031249999999999976
State |000000110> probability=0.031249999999999976
State |000001010> probability=0.031249999999999976
State |000001110> probability=0.031249999999999976
State |000110000> probability=0.031249999999999976
State |000110111> probability=0.031249999999999976
State |000111011> probability=0.031249999999999976
State |000111111> probability=0.031249999999999976
State |010000000> probability=0.031249999999999976
. . .
State |100111011> probability=0.031249999999999976
State |100111111> probability=0.031249999999999976
State |111010000> probability=0.031249999999999976
State |111010111> probability=0.031249999999999976
State |111011011> probability=0.031249999999999976
State |111011111> probability=0.031249999999999976
State |111110000> probability=0.031249999999999976
State |111110111> probability=0.031249999999999976
State |111111011> probability=0.031249999999999976
State |111111111> probability=0.031249999999999976
```

Figure 20: Results of the simulation of the example circuit (for the sake of brevity only some results are shown)

A.4 Definition of the M gate

```
def m_matrix(theta):
    return np.array(
        [
            [np.sin(theta), np.cos(theta)],
            [np.cos(theta), -np.sin(theta)]
        ]
    )
M = AbstractGate("M", [float], matrix_generator=m_matrix, arity=1)
```

Figure 21: myQLM implementation of the M gate

A.5 Simulation of circuits with the M gate

```
# Quantum circuit
prog = Program()
qbits = prog.qalloc(9)

# A(q0) and B(q1) tM(thetas[])en X(q2)
prog.apply(M(thetas[0]), qbits[0])
prog.apply(M(thetas[1]), qbits[1])
prog.apply(CCNOT, qbits[0], qbits[1], qbits[2])

# X(q2) or C(q3) tM(thetas[])en Y(q4)
prog.apply(M(thetas[2]), qbits[3])
prog.apply(CCNOT, qbits[2], qbits[3], qbits[4])
prog.apply(CNOT, qbits[2], qbits[4])
prog.apply(CNOT, qbits[3], qbits[4])

# Y(q4) and (D(q5) or E(q6)) tM(thetas[])en R(q8)
prog.apply(M(thetas[3]), qbits[5])
prog.apply(M(thetas[4]), qbits[6])
prog.apply(CCNOT, qbits[5], qbits[6], qbits[7])
prog.apply(CNOT, qbits[5], qbits[7])
prog.apply(CNOT, qbits[6], qbits[7])
prog.apply(CCNOT, qbits[4], qbits[7], qbits[8])
```

Figure 22: myQLM implementation of the example circuit with M gates

As we can appreciate on the figure above, the circuit is the exact same as the one with the Hadamard gates, but the M gates allow us to introduce the uncertainty into the circuit seamlessly. This code is part of a function that receives as an argument the uncertainty of the facts (the variable thetas, that is a list with the uncertainty of each fact) and returns the program to be executed in the QLM.

```
Results for case [0, 0, 20, 0, 0]
Prob. true: 1.0 | Prob. false: 0.0
Total prob.: 1.0

Results for case [20, 60, 0, 0, 20]
Prob. true: 0.994 | Prob. false: 0.006
Total prob.: 1.0

Results for case [40, 80, 0, 20, 20]
Prob. true: 0.943 | Prob. false: 0.057
Total prob.: 1.0

Results for case [60, 100, 20, 0, 0]
Prob. true: 1.0 | Prob. false: 0.0
Total prob.: 1.0

Results for case [80, 80, 20, 0, 20]
Prob. true: 0.92 | Prob. false: 0.08
Total prob.: 1.0

Results for case [100, 60, 20, 20, 20]
Prob. true: 0.875 | Prob. false: 0.125
Total prob.: 1.0

Results for case [0, 40, 0, 0, 0]
Prob. true: 1.0 | Prob. false: 0.0
Total prob.: 1.0

Results for case [20, 20, 0, 0, 20]
Prob. true: 0.999 | Prob. false: 0.001
Total prob.: 1.0

Results for case [40, 0, 0, 20, 0]
Prob. true: 1.0 | Prob. false: 0.0
Total prob.: 1.0

Results for case [60, 20, 0, 20, 20]
Prob. true: 0.989 | Prob. false: 0.011
Total prob.: 1.0

Results for case [100, 60, 20, 0, 20]
Prob. true: 0.934 | Prob. false: 0.066
Total prob.: 1.0
```

Figure 23: Results of the simulation of the example circuit with the M gate for some cases of uncertainty