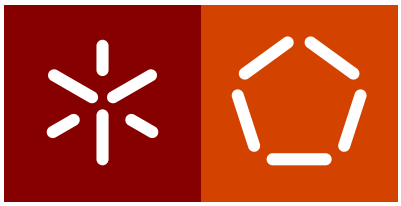
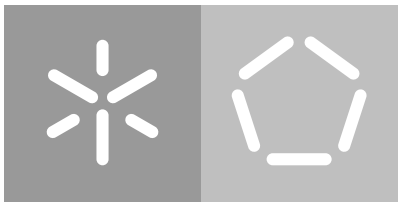


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Vítor Emanuel Gonçalves Fernandes

Integration of time in a quantum process algebra

December 2019



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Vítor Emanuel Gonçalves Fernandes

Integration of time in a quantum process algebra

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
Luís Barbosa
Renato Neves

December 2019

ACKNOWLEDGEMENTS

I want to thanks my parents and my brothers for the support given during this time.

A special thanks to my co-supervisor Renato Neves, for having the patience to teach me and guide me throughout this MSc dissertation. Also to my supervisor Luis Barbosa, that helped me in critical moments.

At last, thanks to all my colleagues and friends.

This MSc project was funded by the SmartEGOV project (NORTE-01-0145-FEDER-000037) supported by Norte Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (EEDF). Part of the results were framed within the KLEE project (POCI-01-0145-FEDER-030947 - PTDC/CCI-COM/30947/2017), funded by ERDF through the Operational Programme for Competitiveness and Internationalisation, COMPETE2020 Programme and by National Funds through the Portuguese funding agency, FCT

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Process algebras are mathematical structures used in Computer Science to study, model, and verify concurrent systems. Essentially, a process algebra consists of a language (used to specify a system that one wishes to study), a semantic domain to interpret the language (which allows the interpretation and the study of the system) and a set of axioms related to the language operators (which facilitates the derivation of properties of the system being studied). These basic ingredients make process algebras powerful tools, with many applications in the development of concurrent systems and many successful stories in industry, Bunte et al. (2019); Groote and Mousavi (2014). The three classical examples of a process algebra are: *CCS* introduced by Robin Milner, *ACP* introduced by Jan Bergstra and Jan Willem Klop, and *CSP* introduced by Tony Hoare. Of the three, the first stands out because of its main goal to isolate and study the elementary principles of communication and concurrency.

The development of Quantum Mechanics supports the design of computational systems ruled by quantum laws, which, in the context of certain problems, perform significantly better than any classical computational system. This is exemplified with Shor and Grover algorithms, respectively used in the factorization of integers and in unstructured searching. Moreover, Quantum computing has applications in the communications area, having as main examples the quantum teleportation protocol and the BB84 communication protocol. However, due to their high sensitivity to noise, quantum computers have a very limited memory space, and therefore they usually integrate a QRAM architecture: essentially, a network of classical computers that process and manage a general task list, invoking quantum computers only when high-cost computational tasks arise. This highlights the importance of extending the theory of process algebras to the quantum domain. In fact, some quantum process algebras were already proposed in last years: examples include *qCCS*, developed by Mingsheng Ying *et al* based on *CCS*, and the *CQP* algebra, introduced by Simon Gay and Rajagopal Nagarajan. Related to *qCCS*, a typing system was developed, where the typable processes are exactly the valid *qCCS* processes.

Current quantum process algebras assume the existence of an ideal quantum system, *i.e.* a quantum system immune to noise. In contrast, the aim of this dissertation is to study and develop a quantum process algebra in which this assumption is discarded. More specifically, we do not assume that a quantum state can be stored indefinitely, it may become corrupted over time, or in other words, have a limited time of coherence. For that goal, 1) we developed a new quantum process algebra that merges the strengths of *qCCS* and *CQP*,

in particular, recursion, memory allocation, and a typing system, so that we can study complex quantum systems that integrate the QRAM architecture; 2) we extended the new process algebra with a notion of time so that we could study its effects on quantum states; and 3) we developed a number of case studies, via the mentioned extension, in which the coherence time of quantum systems has a central role. This includes, for example, a simplified version of the IBM Cloud, which provides access to a quantum computer via *web*.

Keywords: Concurrency, process algebra, quantum, time, CCS, qCCS, TCCS, TqCCS

RESUMO

Álgebras de processos são estruturas matemáticas usadas em Ciências da Computação para o estudo, modelação, e verificação de sistemas concorrentes. Essencialmente, uma álgebra de processos é composta por uma linguagem (usada para especificar o sistema que se deseja estudar), um domínio semântico para interpretação dessa linguagem (o que permite interpretar e estudar o sistema) e um conjunto de axiomas relativos aos operadores da linguagem (o que facilita a derivação de propriedades do sistema em estudo). Estes ingredientes básicos tornam as álgebras de processos ferramentas poderosas, com várias aplicações no desenvolvimento de sistemas concorrentes e várias histórias de sucesso na indústria, Bunte et al. (2019); Groote and Mousavi (2014). Os três exemplos clássicos de álgebras de processos são: *CCS* introduzida por Robin Milner, *ACP* introduzida por Jan Bergstra e Jan Willem Klop, e *CSP* introduzida por Tony Hoare. Das três, destaca-se a primeira como sendo aquela cujo objectivo principal é isolar e estudar os princípios elementares da comunicação e da concurrência.

O desenvolvimento da Mecânica Quântica, permitiu conceber sistemas de computação regidos pelas leis quânticas, que, no contexto de certos problemas, têm um desempenho significativamente melhor que qualquer sistema computacional clássico. Isto é exemplificado com os algoritmos de Shor e de Grover, usados respectivamente na factorização de inteiros e em procuras não estruturadas. Para além disso, a computação quântica tem aplicações na área da comunicação, tendo como exemplos principais o protocolo de teletransporte quântico e o protocolo de comunicação BB84. No entanto devido à sua elevada sensibilidade ao ruído, os computadores quânticos têm um espaço de memória bastante limitado, e portanto costumam integrar uma arquitectura QRAM, em que uma rede de computadores clássicos processam e gerem uma lista geral de tarefas, invocando o computador quântico apenas quando surgem certas tarefas de elevado custo computacional. Isto sublinha a importância de estender a teoria das álgebras de processos ao domínio quântico. De facto, algumas álgebras de processos quânticos já foram propostas nos últimos anos: exemplos incluem *qCCS*, desenvolvida por Mingsheng Ying *et al* e que tem por base *CCS*, e a álgebra *CQP*, introduzida por Simon Gay e Rajagopal Nagarajan. Relacionado com *qCCS*, um sistema de tipos foi desenvolvido, onde os processos tipados são exatamente os processos válidos em *qCCS*.

As actuais álgebras de processos quânticos assumem a existência de um sistema quântico ideal, *i.e.* um sistema quântico insensível ao ruído. Em contraste, o objectivo deste trabalho é o estudo e desenvolvimento de uma álgebra de processos quânticos em que esta assunção

seja descartada. Mais especificamente, não assumimos que um estado quântico possa ser guardado indefinidamente, mas que possa tornar-se corrompido ao longo do tempo, ou por outras palavras, que tenha um tempo limitado de coerência. Para tal, 1) desenvolvemos uma nova álgebra de processos quânticos que reúne diversos pontos fortes de *qCCS* e *CQP*, em particular recursividade, alocação de memória, e sistemas de tipagem, para que possamos estudar sistemas quânticos complexos e que integrem uma arquitectura QRAM; 2) estendemos a nova álgebra de processos com uma noção de tempo para podermos estudar o efeito deste sobre os estados quânticos; e 3) desenvolvemos uma série de casos de estudo, através da extensão referida, em que o tempo de coerência dos estados quânticos tem um papel central. Isto inclui, por exemplo, uma versão simplificada da IBM Cloud, que providencia acesso a um computador quântico via *web*.

Palavras-chave: Concurrência, álgebra de processos, quântica, tempo, CCS, qCCS, TCCS, TqCCS

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

1.1 MOTIVATION

Process algebras are used to study, model and verify concurrent systems, and have successful case-studies in industry, Bunte et al. (2019); Groote and Mousavi (2014). There are three classical examples of a process algebra: *CCS* developed by Robin Milner, *ACP* developed by Jan Bergstra and Jan Willem Klop, and *CSP* developed by Tony Hoare. Of the three, we highlight the first as the one that aims to isolate and study the elementary principles of communication and concurrency.

It is well known that quantum technologies can have, in a very near future, a great impact in our society even if Quantum Mechanics, as an area, only appeared in the last century. As examples of this, we have the algorithms of Shor and Grove: the first provides an exponential speedup in integer factorization; the second performs rapid searches on non-structured data. Process algebras for quantum computational systems were introduced in the past: *qCCS*, Ying et al. (2009), and *CQP*, Gay and Nagarajan (2006), are the two main examples. The first was developed by Mingsheng Ying *et al* and it is based on *CCS*; the second was developed by Simon Gay and Rajagopal Nagarajan.

The introduction of temporal notions into a quantum process algebra is important, because nowadays the quantum computers that exist are faulty, *i.e.* the qubits are not immune to noise and they have a certain amount of time in which the operations occur without the introduction of noise.

This dissertation has the goal to describe the introduction of a notion of time into a new quantum process algebra that joins the positive aspects of *qCCS* and *CQP*. We also developed a number of case studies to see the theory developed in action.

1.2 CONTRIBUTIONS

To develop our theory we studied a quantum process algebra, more concretely a quantum version of *CCS* that only takes into account quantum variables, known as *qCCS*, Ying et al. (2009). For this quantum process algebra, we developed a *typing system*, which allows

another tool to assure that the no-cloning theorem is respected and that new properties can be found, like the weakening property. Moreover, we proved that the typing system developed has the weakening property. We also proved that the formulas of our typing system are similar to *qCCS* processes.

Next, we present another version of *qCCS*, Feng et al. (2012). This version takes into consideration classical and quantum variables, which makes possible a QRAM architecture. Here, we have proved that there still exists a relation between *qCCS* processes and the typing system. We also proved that classical and quantum weakening is implicit in our typing system.

Through examples developed by us, we present limitations of this new version of *qCCS*. Moreover, we introduce a new action, named *new*, and change the semantic rule of the measurement to overcome the limitations.

Then, we extended *qCCS* with a temporal dimension and presented its syntax and semantics. Furthermore, we developed a typing system. Finally, we adapted the previous examples, so that we can see our theory in action.

Summing up, we have contributed to the evolution of quantum process algebras by incorporating suitable notions of time. With that, we are capable to control the lifetime of qubits and see if all the operations are made within the coherence time of qubits.

1.3 DOCUMENT STRUCTURE

The current dissertation is divided into six chapters. Chapter 1 motivates our work and summarises its mains contributions. Chapter 2 provides the necessary background, namely basic notions of concurrency, and a general view of process algebras. We give a particular focus to one of the classical process algebras: *Calculus of Communicating Systems* (*CCS*), Milner (1989), due to its minimal nature relative to other classic process algebras. Our overview will be mostly based on Aceto et al. (2005). Chapter 3 provides some general concepts relative to the introduction of time in process algebras. In particular, it presents a temporal extension of *CCS* called *Timed CCS*. Then, Chapter 4 presents basic notions of quantum computation and surveys quantum process algebras. Analogously to the previous chapter, we will give a particular focus to a quantum extension of *CCS* known as *quantum CCS* (in short, *qCCS*). As a first contribution of this dissertation, it was developed a typing system for *qCCS*.

Chapter 5 then details the remaining contributions of our work: as discussed before, it amounts to the introduction of a quantum process algebra with timing constraints and associated notions. Finally, Chapter 6 discusses future work and concludes.

2

A BRIEF INTRODUCTION TO PROCESS ALGEBRA

2.1 BASIC NOTIONS OF CONCURRENCY

The notion of concurrency was popularised in the ninetieth century by new technological systems, more specifically railroads and telegraphic communications; in the former as a way to increase the number of trains sharing the same railroad without collisions; in the latter to efficiently handle multiple transmissions over a given set of wires. Not much time after the appearance of the first computational devices, concurrency was also popularised within computer science, and is by now a central concept in the area. The study of concurrent algorithms started in the sixties with Dijkstra's pioneer work, Dijkstra (1965).

So, what is concurrency? Concurrency is the ability to perform different tasks at the same time, providing a way to speed up processes and naturally requiring notions of communication. A standard view of a concurrent system emphasises the division of the workload into simple, small tasks, independent of each other, which are then executed in an interleaved way or simultaneously. Such systems propound thorny challenges: three important cases are the notion of deadlock (happens when tasks arrive at a given state and cannot proceed), mutual exclusion (pertains to the *exclusive* access of an entity to a memory region at a given time), and starvation (happens when a task, in order to perform an operation, waits *ad infinitum* for necessary resources).

Two important features are usually supported by a computational, concurrent system: one is the ability to interact with the environment and the other is that it must exhibit infinite behaviour, *i.e.* it should not stop.

The following simple pseudo-code illustrates these features in tandem:

```
while(true){
   sensor()
   analysis(data)
   execute()
}
```

Let us analyse this pseudo-code in detail: first, the *while loop* mimics a system that never stops. Then, the *sensor()* interacts with the environment to collect data to be analysed by

different tasks, *analysis(data)*. Finally, the tasks communicate with each other to perform an action, *execute()*.

2.2 A BIRD'S EYE VIEW OF PROCESS ALGEBRA

In the beginning of the seventies, concurrency theory was limited to the thesis of Petri, Petri (1962). In the same period of time, there were three central styles of formal reasoning about computer programs by means of program semantics:

1. Operational semantics, McCarthy (1963)
2. Denotational semantics, Scott and Strachey (1971)
3. Axiomatic semantics, Hoare (1969)

These styles of semantics focused primarily on *sequential programming languages*. Later on the scientific community tried to extend them to accommodate a notion of parallelism, giving rise to three important solutions (currently, known as process algebras): *CCS*, Milner (1989), *CSP*, Hoare (1978), and *ACP*, Bergstra and Klop (1984). More information about this topic is available in Baeten (2005).

Briefly, a *process algebra* is a tool to model concurrency and, as its name suggests, is an algebra of processes. By analysing these two words individually, process and algebra, we are able to give a notion of what it is a process algebra. The term *process* refers to the computational processing of a program, and in essence is regarded as a (possibly infinite) sequence of *actions*. The orderly execution of this possibly infinite sequence of actions is known as the *behaviour* of a process. The term *algebra* is related with the composition of certain objects according a number of axioms and rules. Therefore, a process algebra is a set of processes that can be described by a set of algebraic rules.

Recall from the previous section that a concurrent system is composed by small tasks. These small tasks are processes, and they originate others via composition operators: *e.g.* the parallel composition operator ($||$), the action prefix ($.$), and the choice operator ($+$). Having this in mind, process algebras are used to describe the behaviour of a concurrent system and allows to reason about it via algebraic concepts.

In concurrency, one key aspect is the communication between processes. Attention on the next example.

Example 2.2.1. Consider two people negotiating a price for a certain item. The person number one suggests a price and the person number two hears it and decides if he agrees or not. In case of agreement, the negotiation is over. Otherwise, this procedure repeats itself until an agreement is achieved. This interaction between them can be represented by

two processes composed in parallel – a line over an action represents the action of talking and the lack of it represents the action of hearing:

$$\mathtt{P}_1 = \overline{\mathtt{price}}.(\mathtt{agree}.\mathtt{Deal} + \mathtt{disagree}.\mathtt{P}_1)$$
$$\mathtt{P}_2 = \mathtt{price}.(\overline{\mathtt{agree}}.\mathtt{Deal} + \overline{\mathtt{disagree}}.\mathtt{P}_2)$$
$$\mathtt{Negotiation} = \mathtt{P}_1 \mid\mid \mathtt{P}_2$$

The intuitions about process algebra given earlier mentioned the description of processes' behaviour via algebraic concepts. This is a very important mechanism because it allows a precise, mathematical comparison of the behaviour of two different processes. Baeten et al. (2010) says that a process can be viewed as an automaton, and because of that, the comparison of the behaviour of two different processes can also resort to automata theory. Briefly, an automaton is composed of a number of states (there are one or more initial states and one or more final states), a number of transitions and a number of actions that allow to transit from one state to another state. The behaviour of an automaton is an execution path of actions that lead from the initial state to a final state, Baeten et al. (2010).

Example 2.2.2. Recall the previous example and the process $\mathtt{P}_1$. When an action of $\mathtt{P}_1$ is performed, another process is obtained. As for example, after performing the action $\overline{\mathtt{price}}$, a new process ($\mathtt{P}_{11}$) is created.

Next we are going to represent the automaton of $\mathtt{P}_1$, where the states of the automaton are the processes and the transitions between states are the actions.

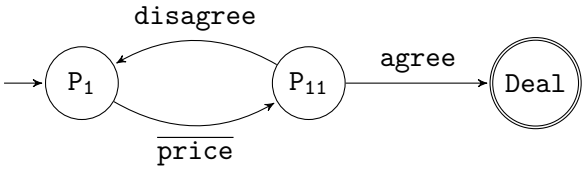


Figure 1: Automaton of $\mathtt{P}_1$

Now it is necessary to have a tool in order to compare the behaviour of two automata, expressing in this way a notion of equivalence, Hopcroft et al. (2006). With this notion, two automata that have the same behaviour will be classified as equivalent. Since processes can be represented by automata, two processes are equivalent if the behaviour of their automata is the same. However, this notion of equivalence is not complete, because it lacks the notion of interaction, that is, it does not take into account the possibility that processes can interact with one another (for example, when in a parallel composition context). Furthermore, when we take into consideration the non-deterministic operator it is not possible to distinguish between $a.(\mathtt{P} + \mathtt{Q})$ and $a.\mathtt{P} + a.\mathtt{Q}$. This will be made clearer in Subsection 2.3.4. To surpass these faults, another notion of equivalence was suggested: *bissimilarity*, Blackburn et al.

(2002). Here, two automata are said to be equivalent if the behaviour of both can be imitated in every state that they may reach.

In this work we will focus on the process algebra proposed by Robin Milner, Milner (1989). Although *ACP* and *CSP* are hand to hand with *CCS* the three classical theories of process algebra, the choice of the last one is due to its simplicity and restriction to the basic elements of concurrency. Having said that, in the next section we will outline *CCS*.

2.3 THE CALCULUS OF COMMUNICATING SYSTEMS

2.3.1 *Basic features and key concepts*

The Calculus of Communicating Systems, *CCS*, was introduced by Robin Milner in the eighties to describe a theory of processes that communicate and interact concurrently with each other.

A process in *CCS* can be intuitively seen as a black box with ports, called *channels*, for communicating with the external environment. The channels that receive information are called *input* channels and the ones that transmit information are called *output* channels. The output channels are decorated by a bar over their name to indicate that they are of type output (e.g: $\overline{\texttt{out}}$). Figure 2 presents an example of a *CCS* process named `Black Box` where the channels `in` and $\overline{\texttt{out}}$ interact with the external environment.

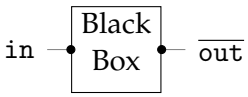


Figure 2: Notion of a *CCS* process

The terms `in` and $\overline{\texttt{out}}$ are *labels*, which are used to give names to channels and from now on will be called *actions*.

Next, the syntax and the semantics of *CCS* will be presented.

2.3.2 *Syntax*

The sentences of *CCS* are called processes and are formed via different constructors. In *CCS* there is a special process, called `nil` and represented by `0`, that mimics the behaviour of a deadlock. Next, the constructors will be outlined and an example, for each constructor, provided for a better understanding. The examples provided here are inspired by Aceto et al. (2005).

The first constructor to be presented is the *action prefixing*, which is denoted by a point (.). Look at the next example which mimics the behaviour of a coffee addict.

Example 2.3.1. Consider a process CA, which represents a coffee addict with the channels $\overline{\texttt{coin}}$ and coffee to communicate with the environment. The process is represented by

$$\texttt{CA} \stackrel{def}{=} \overline{\texttt{coin}}.\texttt{coffee}.\texttt{CA}$$

and can be intuitively seen as in Figure 3:

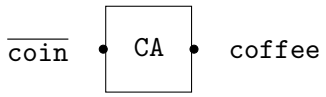


Figure 3: Coffee Addict process

In an initial state the coffee addict outputs a coin transiting to a state in which the action to input coffee can be performed. After this action, the process returns to the initial state where every step, previously outlined, can be realized again. Through this example, it is noticeable that the sequence in which the actions took place is from left to right and this is how processes in *CCS* evolve.

The next constructor is the *sum composition* operator, denoted by a plus signal (+), and which represents a non-deterministic choice. This feature is better understood by analysing the next example.

Example 2.3.2. Consider now a process, CTM, describing a machine that sells coffee and tea with channels coin, $\overline{\texttt{coffee}}$ and $\overline{\texttt{tea}}$ to communicate with the environment. The process is defined by

$$\texttt{CTM} \stackrel{def}{=} \texttt{coin}.(\overline{\texttt{coffee}}.\texttt{CTM} + \overline{\texttt{tea}}.\texttt{CTM})$$

and using the analogy of the black box can be seen as:

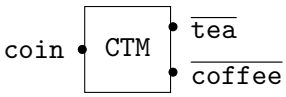


Figure 4: Coffee or Tea Machine process

Recall from the previous example that a *CCS* process performs its actions from left to right. Therefore, the first action to be executed is the input of a coin into the machine allowing a transition to a state where there are two possible outcomes, $\overline{\texttt{coffee}}$ or $\overline{\texttt{tea}}$. This state is composed by the sum operator and it is non-deterministic, *i.e.*, for the same input (coin) there are two outcomes (coffee and tea). This means that the output might differ for the same input. Independently of the choice, the process transits to its initial state where, once again, every step can be repeated.

The *parallel composition* operator, denoted by a vertical bar (|), is an important constructor in *CCS* because it allows the communication between processes and to concurrently execute them. Recall from the beginning of this section that there are two types of channel, input and output, and that each channel is labelled. In order for a communication to occur, the two channels used in the communication must be complementary, that is, the input and the output channels must have the same label. This is represented by an invisible action denoted by τ, which has no complement. See the following example that represents the interaction between a computer scientist and a coffee machine.

Example 2.3.3. Consider two different processes, CS and CM, where the first represents a computer scientist with channels $\overline{\texttt{pub}}$, $\overline{\texttt{coin}}$ and coffee and the second a coffee machine with channels coin and $\overline{\texttt{coffee}}$. Both processes are defined as

$$\texttt{CS} \stackrel{def}{=} \overline{\texttt{pub}}.\overline{\texttt{coin}}.\texttt{coffee}.\texttt{CS}$$
$$\texttt{CM} \stackrel{def}{=} \texttt{coin}.\overline{\texttt{coffee}}.\texttt{CM}$$

and can be represented individually as in Figure 5:

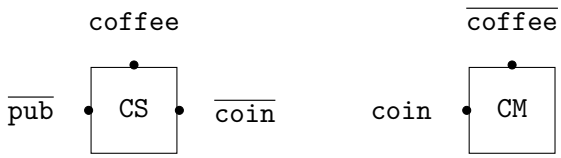


Figure 5: Computer Scientist and Coffee Machine processes

On the left hand side, the computer scientist after producing a publication is going to spend a coin to obtain a coffee so that he is able to produce another publication. On the right hand side, the coffee machine requires a coin to output the coffee. When one puts these processes in parallel, a new one is created:

$$\texttt{CM} \mid \texttt{CS} = \texttt{coin}.\overline{\texttt{coffee}}.\texttt{CM} \mid \overline{\texttt{pub}}.\overline{\texttt{coin}}.\texttt{coffee}.\texttt{CS}$$

and can be pictorially seen in Figure 6, where the complementary channels are connected:

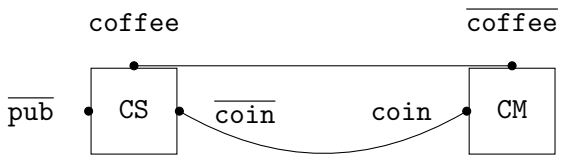


Figure 6: Computer Scientist and Coffee Machine processes in parallel

In this new process the first action to be executed is the output of a publication, which shows that processes that are composed through the parallel operator can evolve independently of each other.

$$\texttt{coin}.\overline{\texttt{coffee}}.\texttt{CM} \mid \overline{\texttt{coin}}.\texttt{coffee}.\texttt{CS}$$

The parallel process is now into a state where the computer scientist will give a coin and the coffee machine will receive it. In this case, the state transition is performed by the action τ. Next we have the output of coffee by the coffee machine and the input of coffee by the computer scientist. This situation is similar to the previous one and the transition to the initial state is performed, again, through a τ action.

The next constructor is the *restriction operator* which is denoted by $\backslash$ and has the purpose of hiding complementary channels of processes in parallel to the exterior environment. See the following example for a better understanding.

Example 2.3.4. Consider the processes CS and CM previously defined and a new computer scientist process CS′. By using the parallel operator a new process is defined

$$\texttt{CS} \mid \texttt{CM} \mid \texttt{CS}'$$

and its representation is in Figure 7:

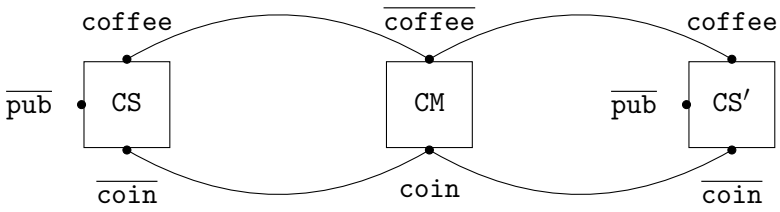


Figure 7: Computer Scientists and Coffee Machine parallel process

Through Figure 7 one can see that the complementary ports are all connected to each other, meaning that the two computer scientists have access to the same coffee machine. Now consider that the coffee machine is only available to the computer scientist on the left. To represent this condition using a *CCS* process one needs to use the restriction operator. The new process is defined as

$$(\texttt{CS} \mid \texttt{CM})\backslash\{\texttt{coin},\texttt{coffee}\} \mid \texttt{CS}'$$

and the figure below shows the intuitive representation of this new process:

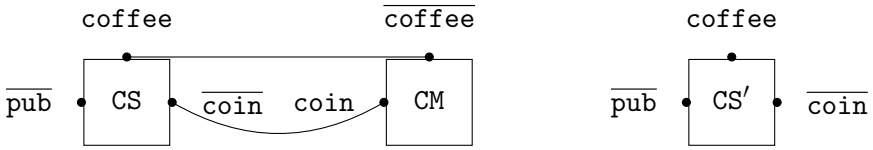


Figure 8: Computer Scientists and Coffee Machine restricted parallel process

As it can be seen in Figure 8 the complementary channels of the coffee machine and the computer scientist on the left are connected. On the other hand, the computer scientist on the right does not have any connection to the coffee machine, despite the existence of complementary channels. This shows that the channels of the coffee machine were hidden from the computer scientist on the right.

At last, the *renaming operator*, is a relabelling function. Let us illustrate it via the following example.

Example 2.3.5. Consider a vending machine that after the input of a coin outputs an item.

$$\mathtt{VM} \overset{def}{=} \mathtt{coin}.\overline{\mathtt{item}}.\mathtt{VM}$$

Consider now a vending machine that outputs a cookie. Instead of creating a new process, one can instead use a renaming operator that changes $\overline{\mathtt{item}}$ to $\overline{\mathtt{cookie}}$.

$$\mathtt{VM}[\mathtt{cookie}/\mathtt{item}] \overset{def}{=} \mathtt{coin}.\overline{\mathtt{cookie}}.\mathtt{VM}$$

With this constructor it is possible to create processes were actions have a general meaning. The symbol "/" is used as substitution in this example to mimic a relabelling function f, presented in Definition 2.3.6.

In the examples above, we saw that processes were identified by names. With this, it is possible to create recursive processes which enable expressing infinite behaviour.

The presentation of a grammar that describes the syntax of *CCS* is a good way to summarize all that was said in this subsection; but first let us present the sets used by this grammar. The set of labels, $\mathbf{L}$, is composed by an infinite set of channel names, $\mathbf{A}$, and of their complements, $\bar{\mathbf{A}}$.

$$\mathbf{L} = \mathbf{A} \cup \bar{\mathbf{A}}$$

The set of actions, **Act**, is composed by the set of labels and by the invisible action τ.

$$\mathbf{Act} = \mathbf{L} \cup \{\tau\}$$

Now, we are able to present the syntax of *CCS*.

Definition 2.3.6. The collection $\mathtt{P}$ and $\mathtt{Q}$ of *CCS* expressions is given by the following grammar:

$\mathtt{P}, \mathtt{Q} ::= \mathtt{K} \mid \alpha.\mathtt{P} \mid \sum_{i\in I}\mathtt{P}_i \mid \mathtt{P}|\mathtt{Q} \mid \mathtt{P}\backslash\mathbf{L} \mid \mathtt{P}[\mathtt{f}]$

where

1. $\mathtt{K}$ is a process name in a countably infinite collection $\mathbf{K}$ of process names

2. α is an action in **Act**
3. I is an index set
4. **L** is a set of labels
5. $\mathtt{f} : \mathbf{Act} \to \mathbf{Act}$ is a relabelling function satisfying the following constraints:

$$\mathtt{f}(\tau) = \tau$$
$$\mathtt{f}(\overline{\mathtt{a}}) = \overline{\mathtt{f}(\mathtt{a})} \text{ for each label } \mathtt{a}$$

Although not explicit in the grammar above, *CCS* supports *value passing* and *if-then-else clauses*. Until here, the processes considered did not exchange data between them; albeit transmission of data being important for processes that communicate. So, in order to allow processes to transmit data, *CCS* was extended with value passing. An example of value passing is the one-place buffer, sketched in Aceto et al. (2005).

Example 2.3.7. Here is a process that receives a value, increments it, and outputs the result.

$$\mathtt{P} \overset{def}{=} \mathtt{in}(\mathtt{x}).\mathtt{P}(\mathtt{x})$$
$$\mathtt{P}(\mathtt{x}) \overset{def}{=} \overline{\mathtt{out}}(\mathtt{x}+1).\mathtt{P}$$

Through the action $\mathtt{in}(\mathtt{x})$ the process $\mathtt{P}$ will receive a value $\mathtt{x}$ that is going to be passed to $\mathtt{P}(\mathtt{x})$. In the latter, $\mathtt{x}$ will be increased by one and outputted, $\overline{\mathtt{out}}(\mathtt{x}+1)$. After this, the process $\mathtt{P}$ is able to receive another value and repeat the same procedure.

Before presenting the if-then-else clauses, we will present the *if-clauses*, because we aim to write the former through the latter. Having said that, the if-clauses are written as follows:

if c **then** P

This means that if the condition c is obeyed then the process $\mathtt{P}$ executes.
The if-then-else clauses are represented by the following structure.

if c **then** $\mathtt{P}$ **else** $\mathtt{Q}$

This means that if the condition c is obeyed then the process $\mathtt{P}$ executes otherwise $\mathtt{Q}$ will execute. With this, we can write an if-then-else clause by means of the non-deterministic operator.

(**if** c **then** $\mathtt{P}$) + (**if** not c **then** $\mathtt{Q}$)

2.3.3 *Semantics*

The semantics of *CCS* is given in terms of a Structural Operational Semantics, Milner (1989), in short *SOS*. *SOS* is a set of rules that dictate which are the available transition between processes.
In this subsection the semantics of *CCS* will be presented as well as the corresponding expansion law. We will also formally show that a *CCS* process can be seen as an automaton, as stated in the Section 2.2.
In the previous subsection, through the examples provided, we saw that processes in *CCS* are composed by actions. We also saw that it is possible to define processes from other processes, enabling this way the definition of recursive processes. In either way, when an action of a process is performed a transition to another process happens. This transition is the same that occur in automata. So, one can infer that *CCS* processes can be seen as automata. To clarify this idea consider the following example.

Example 2.3.8. Recall the *CCS* process of a coffee addict.

$$\texttt{CA} \stackrel{def}{=} \overline{\texttt{coin}}.\texttt{coffee}.\texttt{CA}$$

Through the execution of the action $\overline{\texttt{coin}}$ the process will transit to a state defined by another process.

$$\texttt{CA}' \stackrel{def}{=} \texttt{coffee}.\texttt{CA}$$

After performing the action `coffee` the process will return to the initial state.

With this is possible to verify that state transitions are caused by the execution of actions and that states are definitions of processes. The following automaton mimics the transitions of the *CCS* process `CA`.

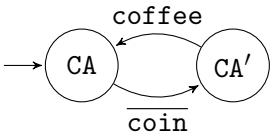


Figure 9: `CA` automaton

In this automaton, `CA` is the initial state which transits to $\texttt{CA}'$ through $\overline{\texttt{coin}}$. In $\texttt{CA}'$ the transition to the initial state is made through the action `coffee` imitating in this way the transitions that are made by the *CCS* process representing the coffee addict.

It is important to recall the definition of behaviour and what can be done with it. The behaviour of a process describes the actions that allows the transitions between states and the order in which they are made. With this, it is possible to give a notion of equivalence

between processes and see if they are equal. Since the behaviour of a *CCS* process is described by the actions performed and the states that were reached, it may be specified by automata, in particular, by Labelled Transition Systems, in short *LTS*.

The definition of *LTS* and *SOS* will be given in the next subsection.

LTS and SOS

As mentioned in the previous subsection, a *CCS* process can be represented by a *LTS*. The latter is defined as follows.

Definition 2.3.9. A labelled transition system (*LTS*) is a triple (**Proc**,**Act**,$\{\xrightarrow{a!} \mid a \in \mathbf{Act}\}$), where:

1. **Proc** is a set of states
2. **Act** is a set of actions
3. $\xrightarrow{a} \subseteq \mathbf{Proc} \times \mathbf{Proc}$ is a family of transition relations indexed by the actions in $a \in \mathbf{Act}$;

To represent the transition relations that were described in the last point of the Definition 2.3.9, we will use the notation $s \xrightarrow{a} s'$.

Previously, *SOS* was described as a set of rules that dictates how a process evolves along its execution. We will now present the rules used by *CCS*.

Action: $\dfrac{}{\alpha.\mathtt{P} \xrightarrow{\alpha} \mathtt{P}}$	**Con:** $\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P}'}{\mathtt{K} \xrightarrow{\alpha} \mathtt{P}'}\ \mathtt{K} \overset{\mathtt{def}}{=} \mathtt{P}$
Sum$_j$**:** $\dfrac{\mathtt{P}_j \xrightarrow{\alpha} \mathtt{P}'_j}{\sum_{i\in I} \mathtt{P}_i \xrightarrow{\alpha} \mathtt{P}'_j}\ j \in I$	**Com**$_1$**:** $\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P}'}{\mathtt{P} \mid \mathtt{Q} \xrightarrow{\alpha} \mathtt{P}' \mid \mathtt{Q}}$
Com$_2$**:** $\dfrac{\mathtt{Q} \xrightarrow{\alpha} \mathtt{Q}'}{\mathtt{P} \mid \mathtt{Q} \xrightarrow{\alpha} \mathtt{P} \mid \mathtt{Q}'}$	**Com**$_3$**:** $\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P}' \qquad \mathtt{Q} \xrightarrow{\overline{\alpha}} \mathtt{Q}'}{\mathtt{P} \mid \mathtt{Q} \xrightarrow{\tau} \mathtt{P}' \mid \mathtt{Q}'}$
Rel: $\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P}'}{\mathtt{P}[\mathtt{f}] \xrightarrow{f(\alpha)} \mathtt{P}'[\mathtt{f}]}$	**Res:** $\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P}'}{\mathtt{P}\backslash\mathtt{L} \xrightarrow{\alpha} \mathtt{P}'\backslash\mathtt{L}}\ \alpha, \overline{\alpha} \notin \mathtt{L}$

Table 1: *SOS* rules of *CCS*

CCS processes can be seen as automata, in particular, as *LTS*s. With this, a system can have information about all the transitions that are made. By using the *SOS* rules, we verify if the transitions are possible or not. Moreover, by making use of the *LTS* and the *SOS* rules, we can reason about processes. This is why *LTS* and *SOS* are important tools in *CCS*.

2.3.4 *Behavioural Equivalence*

The notion of equivalence between processes requires that we recall the notion of an equivalence relation.

Definition 2.3.10. An equivalence relation $\mathcal{R}$ over a set X is a subset of $X \times X$ (set of pairs of elements of X) that satisfies the following constraints:

1. $\mathcal{R}$ is reflexive : $x\mathcal{R}x$ for each $x \in X$
2. $\mathcal{R}$ is symmetric : $x\mathcal{R}y \Rightarrow y\mathcal{R}x$ for all $x, y \in X$
3. $\mathcal{R}$ is transitive : $x\mathcal{R}y \wedge y\mathcal{R}z \Rightarrow x\mathcal{R}z$ for all $x, y, z \in X$

Next we will present the notions of Trace Equivalence and Bisimulation (both strong and weak), which provide different notions of process equivalence.

Trace Equivalence

Before entering in detail some concepts must be reminded.

1. *LTS* are automata whose transitions are labelled by actions
2. a trace of a process P is a sequence of actions $(\alpha_1,\ \alpha_2\ \ldots\ \alpha_k)$ such that $\mathtt{P} = \mathtt{P}_0 \xrightarrow{\alpha_1} \mathtt{P}_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_k} \mathtt{P}_k$
   Traces(P) define the collection of traces of P

We already know that a *CCS* process corresponds to an automaton. Thus, a possible notion of process equivalence is trace equality: two processes are equivalent if they have the same set of traces. Unfortunately, there is one key element missing in this notion, namely interaction. Let us prove that Trace Equivalence is not a suitable notion of process equivalence via a concrete scenario. Consider three *CCS* processes defined as follows:

$$\mathtt{P} \overset{def}{=} \mathtt{a}.(\overline{\mathtt{b}}.\mathtt{P} + \overline{\mathtt{c}}.\mathtt{P})$$
$$\mathtt{Q} \overset{def}{=} \mathtt{a}.\overline{\mathtt{b}}.\mathtt{Q} + \mathtt{a}.\overline{\mathtt{c}}.\mathtt{Q}$$
$$\mathtt{A} \overset{def}{=} \overline{\mathtt{a}}.\mathtt{c}.\mathtt{A}$$

where the processes P and Q have the same trace with different *CCS* expressions. Through the usage of the parallel composition operator $\mathtt{A} \mid \mathtt{P}$ and $\mathtt{A} \mid \mathtt{Q}$ are created. When one calculates the set of traces of both processes it is obvious that they have a transition by τ first, where $\mathtt{A}|\mathtt{P}$ evolves to $\mathtt{c}.\mathtt{A}|(\overline{\mathtt{b}}.\mathtt{P} + \overline{\mathtt{c}}.\mathtt{P})$ but $\mathtt{A}|\mathtt{Q}$ could transit to $\mathtt{c}.\mathtt{A}|\overline{\mathtt{b}}.\mathtt{Q}$ or $\mathtt{c}.\mathtt{A}|\overline{\mathtt{c}}.\mathtt{Q}$. Imagine that the transition is made to $\mathtt{c}.\mathtt{A}|\overline{\mathtt{b}}.\mathtt{Q}$. In this state, the parallel process has entered in deadlock, as the channels are not complementary. A deadlock state will also happens to the parallel

process A | P if the no-deterministic operator, in P, chose to perform the action $\overline{b}$. When one compares the traces of P and Q, we see that they have equivalent traces. Although, when they are put in parallel with another process, A, the path to deadlock is different. This shows that this attempt is incomplete. The automata of A|P, Figure 10, and A|Q, Figure 11, will be represented for a better understanding of the case presented above.

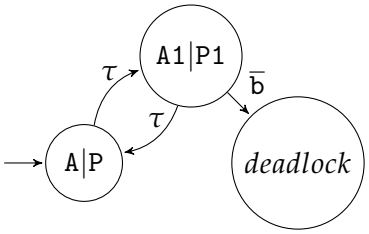


Figure 10: A|P automaton

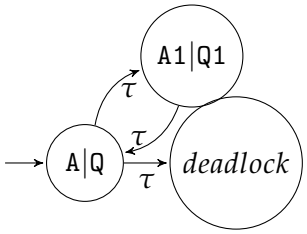


Figure 11: A|Q automaton

In Figure 10 we have an initial state, A | P, that can only transit to the state A_1 | P_1 through τ. This new state has two possibilities, one is to return to A | P, by performing the action τ, and the other is to enter in a deadlock state through the execution of the action $\overline{b}$. In Figure 11 the initial state, A | Q, can transit to the state A_1 | Q_1, by performing the action τ, or to a deadlock state. If the transition is made to A_1 | Q_1, then we can only return to the initial state through τ.

With this, we can take another conclusion.

$$\alpha.(P+Q) \neq \alpha.P + \alpha.Q$$

On the left side, the action α is performed and then the choice for the process P or Q is made. On the right side when the action α is performed one of the processes, P or Q, have already been chosen.

Strong Bisimulation

The concept of bisimulation first appeared on automata, reason why we are going to consider automata to explain strong and weak bisimulation.

Strong bisimulation relates states (of automata) that through the execution of the same set of actions achieve equivalent states. That is, the processes should have the same trace and the states reached by them must be equivalent. In this way, we take into consideration the interaction between processes. This intuition is formally described as follows.

Definition 2.3.11. A binary relation $\mathcal{R}$ over the set of states of an *LTS* is a *bisimulation* iff whenever $s_1 \mathcal{R} s_2$ and α is an action:

1. if $s_1 \xrightarrow{\alpha} s_1'$, then there is a transition $s_2 \xrightarrow{\alpha} s_2'$ such that $s_1' \mathcal{R} s_2'$
2. if $s_2 \xrightarrow{\alpha} s_2'$, then there is a transition $s_1 \xrightarrow{\alpha} s_1'$ such that $s_1' \mathcal{R} s_2'$

When two processes are related by a strong bisimulation they are denoted by $\mathtt{P} \sim \mathtt{Q}$. The symbol $\sim$ denotes the biggest strong bisimulation; its formal definition according to Milner (1989) is as follows:

Definition 2.3.12. $\mathtt{P}$ and $\mathtt{Q}$ are *strongly equivalent* or *strongly bisimilar*, written $\mathtt{P} \sim \mathtt{Q}$, if $(\mathtt{P},\mathtt{Q}) \in \mathcal{R}$ for some strong bisimulation $\mathcal{R}$. This may be equivalently expressed as follows:

$$\sim = \bigcup \{ \mathcal{R} \ : \ \mathcal{R} \text{ is a strong bisimulation } \}$$

Theorem 5.1 in Aceto et al. (2005) presents the properties of $\sim$ and proves them. Here, we only present the properties:

Theorem 2.3.13. For all *LTS*s, the relation $\sim$ is

1. an equivalence relation
2. the largest strong bisimulation

In the fourth chapter of Milner (1989), Robin Milner presents in Proposition 7 the *monoid laws* and in Proposition 8 the *static laws*. Although both laws show relations between *CCS* processes and strong bisimulation, the former focuses on the non-deterministic operator while the latter focuses on the parallel operator, as well as relabelling and restriction.

One interesting fact is that the relation $\sim$ is also a congruence, as proved in the fourth chapter of Milner (1989). A congruence is a property that preserves the relation of two processes when they are put in a "hole". Intuitively, the concept of "hole" can be seen as an "incomplete" operator that is waiting to receive a process $\mathtt{P}$ to become complete. When the process $\mathtt{P}$ is put in the hole [], denoted by $[\mathtt{P}]$, the process $\mathtt{P}$ becomes a "subprocess" of $[\mathtt{P}]$.

Using the concept of a hole, we are able to define congruence in a formal way:

Definition 2.3.14. $\mathcal{R}$ is a congruence if and only if $[\mathtt{P}]\mathcal{R}[\mathtt{Q}]$ holds for all processes $\mathtt{P}$ and $\mathtt{Q}$, such that $\mathtt{P} \sim \mathtt{Q}$.

Here, we enumerate what needs to be proved for $\sim$ to be a congruence. For that, assume that $\mathtt{P} \sim \mathtt{Q}$. Then we need to show that,

1. $\alpha.\mathtt{P} \sim \alpha.\mathtt{Q}$
2. $\mathtt{P} + \mathtt{R} \sim \mathtt{Q} + \mathtt{R}$
3. $\mathtt{P} \mid \mathtt{R} \sim \mathtt{Q} \mid \mathtt{R}$
4. $\mathtt{P}[\mathtt{f}] \sim \mathtt{Q}[\mathtt{f}]$
5. $\mathtt{P} \backslash \mathtt{L} \sim \mathtt{Q} \backslash \mathtt{L}$

Now, we are going to present a simplified version of the expansion law presented by Robin Milner in Milner (1989). In a very short way, the expansion law distinguishes the execution of actions of processes in two: asynchronous and synchronous. The first ones are actions that processes perform individually whilst the latter represents actions that are performed by two processes at the same time, which is the case of communication. To present the expansion law, we are going to consider two processes, as follows:

Proposition 2.3.15. Let $P \equiv P_1 \mid P_2$. Then

$$P \sim \sum \left\{ \alpha.(P'_1 \mid P_2) : P_1 \xrightarrow{\alpha} P'_1 \right\} + \sum \left\{ \alpha.(P_1 \mid P'_2) : P_2 \xrightarrow{\alpha} P'_2 \right\} +$$
$$\sum \left\{ \tau.(P'_1 \mid P'_2) : P_1 \xrightarrow{\alpha} P'_1, P_2 \xrightarrow{\overline{\alpha}} P'_2 \text{ or } P_1 \xrightarrow{\overline{\alpha}} P'_1, P_2 \xrightarrow{\alpha} P'_2 \right\}$$

where the first and the second sums are the asynchronous actions and the third sum the synchronous.

In Milner (1989) there is a step-by-step guide that explains how the expansion law was obtained.

To give some intuitions on how to apply this notion of bisimulation, we will outline an exercise presented in Aceto et al. (2005).

Consider the processes P and Q defined as

$$P \overset{def}{=} a.P_1 \qquad Q \overset{def}{=} a.Q_1$$
$$P_1 \overset{def}{=} b.P + c.P \qquad Q_1 \overset{def}{=} b.Q_2 + c.Q$$
$$Q_2 \overset{def}{=} a.Q_3$$
$$Q_3 \overset{def}{=} b.Q + c.Q_2$$

and also their respective automata.

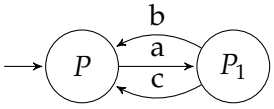


Figure 12: Automaton of P

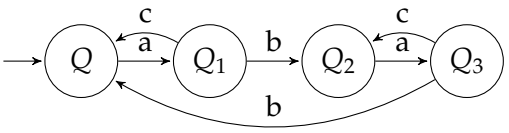


Figure 13: Automaton of Q

To verify that $P \sim Q$ is true we need to recall the definition of strong bisimulation and, for visual aid, use the automata above as support. Let us begin the verification. Through the action a the process P transits to P_1 and the process Q transits to Q_1. Therefore, P and Q are equivalent if P_1 and Q_1 are equivalent. Next, through the action b, P_1 transits to P and Q_1 transits to Q_2. So, now we need to check if P and Q_2 are equivalent, in order to guarantee that the pairs P and Q, and P_1 and Q_1 are equivalent. If one continues this procedure, we will reach a point where we return to the first step. When this happens we are able to conclude

if the processes are strong bisimilar or not. Next, we present all the transitions performed by the processes $\mathtt{P}$ and $\mathtt{Q}$. The symbol $\wedge$ is read as a simple 'and'.

$$\mathtt{P} \xrightarrow{\mathtt{a}} \mathtt{P}_1 \wedge \mathtt{Q} \xrightarrow{\mathtt{a}} \mathtt{Q}_1$$
$$\mathtt{P}_1 \xrightarrow{\mathtt{b}} \mathtt{P} \wedge \mathtt{Q}_1 \xrightarrow{\mathtt{b}} \mathtt{Q}_2$$
$$\mathtt{P} \xrightarrow{\mathtt{a}} \mathtt{P}_1 \wedge \mathtt{Q}_2 \xrightarrow{\mathtt{a}} \mathtt{Q}_3$$
$$\mathtt{P}_1 \xrightarrow{\mathtt{b}} \mathtt{P} \wedge \mathtt{Q}_3 \xrightarrow{\mathtt{b}} \mathtt{Q}$$
$$\mathtt{P}_1 \xrightarrow{\mathtt{c}} \mathtt{P} \wedge \mathtt{Q}_3 \xrightarrow{\mathtt{c}} \mathtt{Q}_2$$
$$\mathtt{P}_1 \xrightarrow{\mathtt{c}} \mathtt{P} \wedge \mathtt{Q}_1 \xrightarrow{\mathtt{c}} \mathtt{Q}$$

After analysing all the states of the two processes we can construct a bisimulation relation $\mathcal{R}$

$$\mathcal{R} = \{(\mathtt{P}, \mathtt{Q}); (\mathtt{P}_1, \mathtt{Q}_1); (\mathtt{P}, \mathtt{Q}_2); (\mathtt{P}_1, \mathtt{Q}_3)\}$$

such that $\mathtt{P} \sim \mathtt{Q}$.

Through this example one can infer that the notion of strong bisimulation, unlike trace equivalence, takes into account the branching structure between processes.

Although strong bisimulation is a better tool to verify the behaviour of processes than trace equivalence, it is also very rigid in the sense that the comparison of behaviours is performed via actions that can be seen. This arises the question "what if processes have in their composition invisible actions?". To answer this question weak bisimulation was developed.

Weak Bisimulation

Weak bisimulation, just like strong bisimulation, is another tool for behavioural equivalence. The difference between both is that weak bisimulation abstracts from invisible actions. The formal definition of weak bisimulation is as follows:

Definition 2.3.16. A binary relation $\mathcal{R}$ over the set of states of an *LTS* is a *weak bisimulation* iff whenever $s_1 \mathcal{R} s_2$ and α is an action:

1. if $s_1 \xrightarrow{\alpha} s_1'$, then there is a transition $s_2 \stackrel{\alpha}{\Rightarrow} s_2'$ such that $s_1' \mathcal{R} s_2'$
2. if $s_2 \xrightarrow{\alpha} s_2'$, then there is a transition $s_1 \stackrel{\alpha}{\Rightarrow} s_1'$ such that $s_1' \mathcal{R} s_2'$

The symbol $\stackrel{\alpha}{\Rightarrow}$ can be seen as a transition $\xrightarrow{\alpha}$ enveloped by a possible empty sequence of τ-transitions, represented as follows:

$$\stackrel{\alpha}{\Rightarrow} = (\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^*$$

where $(\xrightarrow{\tau})^*$ is a possible empty sequence of τ-transitions. When two processes are related by a weak bisimulation we denote them by $\mathtt{P} \approx \mathtt{Q}$. Formally,

Definition 2.3.17. $\mathtt{P}$ and $\mathtt{Q}$ are *observation-equivalent* or (*weakly*) *bisimilar*, written $\mathtt{P} \approx \mathtt{Q}$, if $(\mathtt{P},\mathtt{Q}) \in \mathcal{R}$ for some (weak) bisimulation $\mathcal{R}$. That is,

$$\approx = \bigcup\{\mathcal{R} : \mathcal{R} \text{ is a bisimulation}\}$$

Through Theorem 5.2, Aceto et al. (2005) presents the properties of $\approx$ and indicates how to prove them. Here, we only present the theorem's statement:

Theorem 2.3.18. For all *LTS*s, the relation $\approx$ is:

1. an equivalence relation
2. the largest weakest bisimulation

Unlike strong bisimulation, weak bisimulation is not a congruence, because of the non-deterministic operator. In Aceto et al. (2005), is presented an example of why congruence is not respected on the non-deterministic operator. Now, we will outline an exercise that was inpired on Aceto et al. (2005).

Consider the processes $\mathtt{A}$ and $\mathtt{P}$ defined as:

$$\mathtt{A} \overset{def}{=} \mathtt{a}.0 + \tau.\mathtt{A}_1$$
$$\mathtt{A}_1 \overset{def}{=} \mathtt{b}.0 + \tau.\mathtt{A}$$
$$\mathtt{P} \overset{def}{=} \mathtt{a}.0 + \mathtt{b}.0$$

The automata of both processes are:

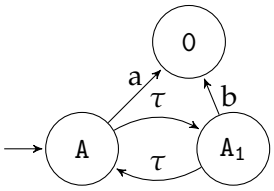


Figure 14: Automaton of $\mathtt{A}$

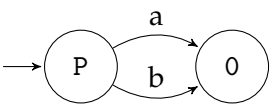


Figure 15: Automaton of $\mathtt{P}$

Now we have to see if the definition stands for these two processes. We follow the same procedure as in strong bisimulation example.

$$\mathtt{P} \xrightarrow{a} 0 \ \wedge \ \mathtt{A} \overset{a}{\Rightarrow} 0 \ (\mathtt{A} \xrightarrow{\tau} \mathtt{A}_1 \xrightarrow{\tau} \mathtt{A} \xrightarrow{a} 0)$$
$$\mathtt{P} \xrightarrow{b} 0 \ \wedge \ \mathtt{A}_1 \overset{b}{\Rightarrow} 0 \ (\mathtt{A}_1 \xrightarrow{\tau} \mathtt{A} \xrightarrow{\tau} \mathtt{A}_1 \xrightarrow{b} 0)$$

In the brackets are the transitions that are made through τ.
The relation $\mathcal{R}$ is:

$$\mathcal{R} = \{(\mathtt{P}, \mathtt{A}); (\mathtt{P}, \mathtt{A}_1); (\mathtt{0}, \mathtt{0})\}$$

With this, we proved that $\mathtt{A} \approx \mathtt{P}$.

3

TIMED PROCESS ALGEBRAS

3.1 MOTIVATION AND CONTEXT

The notion of time is essential to concurrent systems because is the basis of several synchronization mechanisms, specifically it allows to reason about real-timed systems and communication protocols (*e.g. TCP*).

The notion of time that exists in the first process algebras is qualitative, *i.e.* we know that an action is going to be performed after another but we do not know the amount of time it takes to execute the second action. This difference between a qualitative and a quantitative temporal notion is important when one considers, for example, a protocol communication, like *TCP*, in which actions are triggered after some amount of time. Therefore, the introduction of a quantitative notion of time is necessary.

In 1988, G.M. Reed and A.W. Roscoe introduced a quantitative notion of time in *CSP* and developed *TCSP*, Roscoe and Reed (1988). According to Baeten (2005) this was the first work where a quantitative notion of time was introduced in a process algebra.

In 1990, Wang Yi developed *TCCS*, Yi (1990), a timed version of *CCS*, where he presented a wrong expansion theorem. However, a year later, the same author introduced timed variables and corrected the expansion theorem, Yi (1991).

In 1991, J.C.M Baeten and J.A. Bergstra developed ACP_ρ, Baeten and Bergstra (1991), which is a timed version of *ACP*.

Other temporal extensions of *CSP*, *CCS*, and *ACP* were made, but here we only refer to the first works.

3.2 BASIC CONCEPTS

Interestingly, there are several aspects that must be taken into account when extending a process algebra with a notion of time. This includes:

1. Time domain

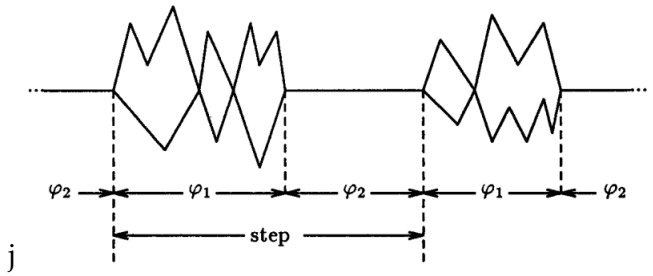
   The time domain can be either discrete (e.g: $\mathbb{N}$) or dense (e.g: $\mathbb{Q}^+$, $\mathbb{R}^+$). The former is used to describe a discrete evolution of time, and is mainly used to specify discrete systems such as computers. The latter, on the other hand, is used to describe real-timed systems, such as the measurement of physical properties over time like radiation, speed, etc.

2. Measurement of time

   The measurement of time can be either *absolute* or *relative*. In the former there is a global clock for all processes while in the latter the passage of time is relative to the previous action executed.

3. Execution sequence

   The execution sequence is normally described by the *two-phase scheme*: there is one phase where processes performs asynchronous or synchronous actions and another where processes stop for time to progress. The scheme of the two-phase step from Nicollin and Sifakis (1992) is depicted in Figure 16.



j

Figure 16: Two-phase scheme

   In Figure 16 we have two phases, denoted by φ_1 and φ_2. In the first one, processes execute synchronous and asynchronous actions. In the second one, no action is performed so that time can progress.

4. Introduction of the notion of time at the syntactic and semantic levels

   Syntactically, there are some alternatives to implement the passage of time, as shown in Section 3.5 of Nicollin and Sifakis (1992). One of them is the delay action. The interested reader can find other cases in Nicollin and Sifakis (1992).

   Semantically, one needs to decide if the execution of atomic actions is instantaneous or has a duration.

5. Model properties

   We need to choose which properties we want the temporal notion to comply. There are several well-known properties, however we are only going to present the most usual ones: namely, time determinism, time additivity, maximal progress and persistency. We will show them by using the mathematical description presented in Nicollin and Sifakis (1992). Intuitively, the expression $\mathtt{P} \xrightarrow{d} \mathtt{P}'$ reads as "$\mathtt{P}$ transits into $\mathtt{P}'$ in d time units".

   - Time Determinism:

     $$\forall\ \mathtt{P},\ \mathtt{P}',\ \mathtt{P}'', d : \mathtt{P} \xrightarrow{d} \mathtt{P}' \ \wedge\ \mathtt{P} \xrightarrow{d} \mathtt{P}'' \Rightarrow \mathtt{P}' = \mathtt{P}''$$

     where $=$ is the syntactic equality, and $\wedge$ means "and".

   - Time Additivity:

     $$\forall\ \mathtt{P},\ \mathtt{P}', d,\ d' : \left(\exists \mathtt{P}'' : \mathtt{P} \xrightarrow{d} \mathtt{P}'' \ \wedge\ \mathtt{P}'' \xrightarrow{d'} \mathtt{P}'\right) \Leftrightarrow \mathtt{P} \xrightarrow{d+d'} \mathtt{P}'$$

   - Maximal Progress:

     $$\forall\ \mathtt{P},\ \mathtt{P}',\ \mathtt{Q}, d : \mathtt{P} \xrightarrow{\tau} \mathtt{P}' \Rightarrow \mathtt{P} \not\xrightarrow{d} \mathtt{Q}$$

   - Persistency

     $$\forall\ \mathtt{P},\ \mathtt{P}',\ \mathtt{Q}, d,\ a : \mathtt{P} \xrightarrow{a} \mathtt{P}' \ \wedge\ \mathtt{P} \xrightarrow{d} \mathtt{Q} \Rightarrow \exists \mathtt{P}'' : \mathtt{Q} \xrightarrow{a} \mathtt{P}''$$

     where a is an action

Next, we present a temporal extension of *CCS*.

3.3 TIMED CALCULUS OF COMMUNICATING SYSTEMS

As already mentioned, process algebras allow to describe and verify concurrent protocols. Sometimes, the latter also involve time, a notion which unfortunately is not integrated in *CCS* or any other classical process algebra. To overcome this, in 1988 an extended version of *CSP* with a temporal notion was presented in Roscoe and Reed (1988). The core feature of this extension is that it allows to describe processes with a quantitative notion of time. In what concerns *CCS*, a first attempt to introduce time was made by Yi (1990). A year later, the same author extended his previous work with *time variables* Yi (1991). *Timed CCS* supports numerical time domains, such as the set of natural numbers or the set of non-negative reals. When we chose as time domain the singleton $\{0\}$, we obtain the classical theory of *CCS*.

The introduction of the passage of time on the syntax is made by time variables and delay actions. The first ones were introduced in order to have a correct expansion theorem, allowing the division of a parallel process into its synchronous and asynchronous part. The interpretation given to time variables is equal to the one of a λ-term. In the latter, $\lambda x.f$

is a term that is expecting an input v. When v is received, $f[v/x]$, we substitute every occurrence of x by v and output the result. Having this in mind, we present three situations that can happen when using time variables.

- $\alpha x@t.\mathtt{P}$ means that a process P is able to receive a message, x, through the channel α after t units of time. When the message arrives, the process behaves like $\mathtt{P}[v/x, d/t]$, where v is the content of the message received and d is the delay before the message is available.
- $\overline{\alpha} v@t.\mathtt{P}$ means that a process P is able to send a message, v, through the channel α after t units of time. When the message is sent, the process behaves like $\mathtt{P}[d/t]$, where d is the delay before the message is sent.
- $\alpha@t.\mathtt{P}$ means that an invisible action is going to happen after t units of time. After that, the process behaves as $\mathtt{P}[d/t]$, where d represents the delay before the execution of the invisible action.

Relatively to the delay actions, they are represented by ϵ and have the purpose of idling for a finite amount of time. For example, a process defined as $\epsilon(d).\mathtt{P}$ behaves like P after idling d time units.

In terms of semantics, the atomic actions are performed instantaneously.

Regarding model properties, *Timed CCS* adopts time determinism, time additivity, maximal progress and persistency. Let us now analyse in detail the syntax, semantics and notions of behavioural equivalence for *Timed CCS*.

3.3.1 *Syntax*

The time domain, $\mathcal{T}$, is the set of non-negative real numbers and the set of temporal actions is $\delta_{\mathcal{T}} = \{\epsilon(d) \mid d \in \mathcal{T} - \{0\}\}$. The set of non-negative reals is ranged by c and d and the set of time variables is ranged by t, u, x, y. Yi (1991) uses time expressions such as $t + d$ and $c \dot{-} d$, where $\dot{-}$ is defined by $c \dot{-} d = 0$ if $d \geq c$, otherwise $c - d$. Recall from *CCS* that the set of labels is composed by the set of names and co-names, $\mathbf{L}$=$\mathbf{A} \cup \bar{\mathbf{A}}$. We now add to the set of actions the set $\delta_{\mathcal{T}}$, yielding $\mathbf{Act}$=$\mathbf{L} \cup \{\tau\} \cup \delta_{\mathcal{T}}$. The actions τ and $\varepsilon(t)$ do not have a complementary name. The syntax of *Timed CCS* is presented in the following grammar:

$$\mathtt{E}, \mathtt{F} ::= \mathtt{nil} \mid \mathtt{X} \mid \epsilon(e).\mathtt{E} \mid \mu@t.\mathtt{E} \mid \mathtt{E} + \mathtt{F} \mid \mathtt{E} \mid \mathtt{F} \mid \mathtt{E}\backslash\mathtt{L} \mid \mathtt{E}[\mathtt{S}]$$

where:

1. nil: represents a deadlock state;
2. X: is a set of agent variables;

3. $\epsilon(e).\mathtt{E}$: is a delay action as described before;
4. $\mu@t.\mathtt{E}$: is a time variable as described before, where μ is a communication channel;
5. $\mathtt{E} + \mathtt{F}$: is the choice composition;
6. $\mathtt{E} \mid \mathtt{F}$: is the parallel composition;
7. $\mathtt{E}\backslash\mathtt{L}$: hides all channels names in L;
8. $\mathtt{E}[\mathtt{S}]$: corresponds to relabelling of functions;

As one can see via this grammar, the main difference between the syntax of *CCS* and *Timed CCS* is the presence of temporal actions. Another difference, is related with the choice operator: here, the choice operator, is binary, while in *CCS* is defined by a sum indexed by a set of values.

Having in mind λ-calculus, the substitution in *Timed CCS* is α-conversion, *i.e.* the bound variables can be replaced with another variable. Furthermore, Yi (1991) states that he does not distinguish $\mu@t.\mathtt{E}$ from $\mu@u.\mathtt{E}[u/t]$.

To familiarize the reader with the notation of *TCCS*, let us made an example.

Example 3.3.1. Consider a coffee addict. In order to drink a coffee, he needs to insert a coin in the machine, wait for the machine to prepare the coffee, and then wait some time for the coffee be at an adequate temperature for drinking. This behaviour can be represented by the following process in *TCCS*.

$$\mathtt{CA} \stackrel{def}{=} \overline{coin}@0.\epsilon(2).coffee@3.drink@0.\mathtt{CA}$$

Through this process, the coffee addict does not wait to input a coin in the machine, $\overline{coin}@0$. However, he needs to wait two units of time to have coffee, $\epsilon(2)$. At last, he waits three more units of time, $coffee@3$, and then he drinks the coffee, $drink@0$.

3.3.2 *Semantics*

Before presenting the semantics for *Timed CCS*, we need to introduce the concept of sort that is necessary for the parallel composition, Yi (1991). Informally, we can see this concept as the set of visible actions that a process P can perform within d units of time. Formally:

Definition 3.3.2. Let E be an agent expression containing no free time variable. Given an assignment function ρ assigning each free agent variable a subset of **L**, we define $Sort_0(E)\rho = \emptyset$ and $Sort_d(E)\rho$ to be the least set satisfying:

$Sort_d(\mathtt{X})\rho = \rho(\mathtt{X})$

$Sort_d(\mathtt{nil})\rho = \emptyset$

$Sort_d(\alpha@t.\mathtt{E})\rho = \{\alpha\}$

$Sort_d(\tau@.\mathtt{E})\rho = \emptyset$

$Sort_d(\epsilon(e).\mathtt{E})\rho = Sort_{d \dot{-} e}(\mathtt{E})\rho$

$Sort_d(\mathtt{E} + \mathtt{F})\rho = Sort_d(\mathtt{E})\rho \cup Sort_d(\mathtt{F})\rho$

$Sort_d(\mathtt{E} \mid \mathtt{F})\rho = Sort_d(\mathtt{E})\rho \cup Sort_d(\mathtt{F})\rho$

$Sort_d(\mathtt{E}\backslash L)\rho = Sort_d(\mathtt{P})\rho - \{\mathtt{L} \cup \overline{\mathtt{L}}\}$

$Sort_d(\mathtt{E}[S])\rho = \{S(\alpha) | \alpha \in Sort_d(\mathtt{E})\rho\}$

To simplify the notation, the assignment function ρ will be omitted for an agent $\mathtt{P}$ and, as a consequence, we write $Sort_d(\mathtt{P})$.

After these definitions, we are prepared to show the semantics of *Timed CCS*.

As in *CCS*, the semantics is given in terms of *SOS*. Therefore, we present in Table 2 the transition rules for *Timed CCS*.

Inaction	$\dfrac{}{\mathtt{nil} \xrightarrow{\epsilon(d)} \mathtt{nil}}$
Prefix	$\dfrac{}{\mu@t.\mathtt{P} \xrightarrow{\mu} \mathtt{P}[0/t]}$ $\quad$ $\dfrac{}{\alpha@t.\mathtt{P} \xrightarrow{\epsilon(d)} \alpha@t.\mathtt{P}[t+d/t]}$ $\dfrac{}{\epsilon(d+e).\mathtt{P} \xrightarrow{\epsilon(d)} \epsilon(e).\mathtt{P}}$ $\quad$ $\dfrac{\mathtt{P} \xrightarrow{\epsilon(d)} \mathtt{P}'}{\epsilon(e).\mathtt{P} \xrightarrow{\epsilon(d+e)} \mathtt{P}'}$ $\dfrac{\mathtt{P} \xrightarrow{\sigma} \mathtt{P}'}{\epsilon(0).\mathtt{P} \xrightarrow{\sigma} \mathtt{P}'}$
Summation	$\dfrac{\mathtt{P} \xrightarrow{\mu} \mathtt{P}'}{\mathtt{P} + \mathtt{Q} \xrightarrow{\mu} \mathtt{P}'}$ $\quad$ $\dfrac{\mathtt{P} \xrightarrow{\epsilon(d)} \mathtt{P}' \quad \mathtt{Q} \xrightarrow{\epsilon(d)} \mathtt{Q}'}{\mathtt{P} + \mathtt{Q} \xrightarrow{\epsilon(d)} \mathtt{P}' + \mathtt{Q}'}$
Composition	$\dfrac{\mathtt{P} \xrightarrow{\mu} \mathtt{P}'}{\mathtt{P} \mid \mathtt{Q} \xrightarrow{\mu} \mathtt{P}'}$ $\quad$ $\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P}' \quad \mathtt{Q} \xrightarrow{\overline{\alpha}} \mathtt{Q}'}{\mathtt{P} \mid \mathtt{Q} \xrightarrow{\tau} \mathtt{P}' \mid \mathtt{Q}'}$ $\dfrac{\mathtt{P} \xrightarrow{\epsilon(d)} \mathtt{P}' \quad \mathtt{Q} \xrightarrow{\epsilon(d)} \mathtt{Q}'}{\mathtt{P} \mid \mathtt{Q} \xrightarrow{\epsilon(d)} \mathtt{P}' \mid \mathtt{Q}'}$ $Sort_d(\mathtt{P}) \cap \overline{Sort_d(\mathtt{Q})} = \emptyset$

Restriction	$\dfrac{\mathtt{P} \xrightarrow{\sigma} \mathtt{P}'}{\mathtt{P}\backslash\mathtt{L} \xrightarrow{\sigma} \mathtt{P}'\backslash\mathtt{L}}\ \sigma, \overline{\sigma} \notin L$
Relabelling	$\dfrac{\mathtt{P} \xrightarrow{\sigma} \mathtt{P}'}{\mathtt{P}[S] \xrightarrow{S(\sigma)} \mathtt{P}'[S]}$

Table 2: *SOS* for *Timed CCS*

where $\alpha \in \mathbf{L}$, $\mu \in \mathbf{L} \cup \{\tau\}$ and $\sigma \in \mathbf{Act}$.

The side-condition $Sort_d(\mathtt{P}) \cap \overline{Sort_d(\mathtt{Q})} = \emptyset$, in the **Composition** rule of the Table 2, is used so that the maximal progression property is satisfied. For a better comprehension consider the following example shown in Yi (1990).

Example 3.3.3. Consider a parallel process, $\mathtt{X} \mid \mathtt{Y}$, where $\mathtt{X} \overset{def}{=} \alpha.\mathtt{A}$ and $\mathtt{Y} \overset{def}{=} \overline{\alpha}.\mathtt{B}$.

The parallel process $\mathtt{X} \mid \mathtt{Y}$ transits to $\mathtt{A} \mid \mathtt{B}$ through a τ-transition, $\mathtt{X} \mid \mathtt{Y} \xrightarrow{\tau} \mathtt{A} \mid \mathtt{B}$ since $\mathtt{X} \xrightarrow{\alpha} \mathtt{A}$ and $\mathtt{Y} \xrightarrow{\overline{\alpha}} \mathtt{B}$. However, $\mathtt{X} \mid \mathtt{Y}$ is not allowed to idle, $\mathtt{X} \mid \mathtt{Y} \xrightarrow{\epsilon(t)} \mathtt{X}' \mid \mathtt{Y}'$, for no t because the side-condition is not obeyed, *i.e.* $Sort_t(X) \cap \overline{Sort_t(Y)} = \{\alpha\}$.

Related to relabelling, its semantics is the same as in *CCS*.

3.3.3 *Behavioural Equivalence*

Strong Bisimulation

The definition of strong bisimulation in *Timed CCS* is the same than the one presented in *CCS*. Because of that, we are only going to explore the expansion theorem and the model properties. The expansion theorem is defined as follows:

Proposition 3.3.4. Let $\mathtt{E} \equiv \sum_{i\in I} \epsilon(c_i).\mu_i@x_i.\mathtt{E}_i$ and $\mathtt{F} \equiv \sum_{j\in J} \epsilon(d_j).\nu_j@y_j.\mathtt{F}_j$. Then

$$\begin{aligned}\mathtt{E} \mid \mathtt{F} \sim & \sum_{\mu_i=\nu_j\in\mathbf{L}} \epsilon(max(c_i,d_j)).\tau.(\mathtt{E}_i[d_j \dot{-} c_i/x_i] \mid \mathtt{F}_j[c_i \dot{-} d_j/y_j]) \\ & + \sum_{i\in I} \epsilon(c_i).\mu_i@x_i.(\mathtt{E}_i \mid \mathtt{F}') + \sum_{j\in J} \epsilon(d_j).\nu_j@y_j.(\mathtt{E}' \mid \mathtt{F}_j)\end{aligned}$$

where

- $\mathtt{F}' \equiv \sum\limits_{j\in J} \epsilon(d_j \dot{-} c_i \dot{-} x_i).\nu_j@y_j.\{\mathtt{F}_j[y_j + ((c_i + x_i) \dot{-} d_j)/y_j]\}$
- $\mathtt{E}' \equiv \sum\limits_{i\in I} \epsilon(c_i \dot{-} d_j \dot{-} y_j).\mu_i@x_i.\{\mathtt{E}_i[x_i + ((d_j + y_j) \dot{-} c_i)/x_i]\}$

In the expression above, the first sum is related to the synchronous part and the remaining sums are related to the asynchronous part. In the synchronous part we have the

condition $max(c_i, d_j)$ and truncated subtractions. The former is a condition to assure that a communication can occur. The latter is a consequence of the former because we need to adjust the process delays. This is necessary for the following reason: since we chose the biggest delay we need to truncate the subtraction to avoid negative delays.

For a better understanding of the condition $max(c_i, d_j)$, consider the following process,

$$\epsilon(2).\alpha@t.\mathtt{P} \mid \epsilon(1).\overline{\alpha}@t.\mathtt{Q}$$

Now, imagine that one unit of time elapsed. The process transits to a state

$$\epsilon(1).\alpha@t.\mathtt{P}[1 + t/t] \mid \overline{\alpha}@t.\mathtt{Q}[1 + t/t]$$

In this state, the process cannot communicate because the action $\alpha@t$ is not available since it is blocked by $\epsilon(1)$. Therefore, we need to wait another unit of time to unblock the action $\alpha@t$ and, consequently, be able to perform a communication. By analysing this simple example, we see that we need to choose the greatest delay to guarantee that the actions that can communicate do not have any temporal constraint.

Real Time Laws

Recall that *Timed CCS* has as model properties maximal progression, time determinacy, time additive and persistency. In Yi (1991) the model properties are written as lemmas and as strong bisimulation relations. In the latter they are known as Real Time Laws. First we present the lemmas and later the Real Time Laws.

Lemma 3.3.5. (Maximal Progress)
If $\mathtt{P} \xrightarrow{\tau} \mathtt{P}'$ for some $\mathtt{P}'$, then $\mathtt{P} \xrightarrow{\epsilon(d)} \mathtt{P}''$ for no d and $\mathtt{P}''$.

Lemma 3.3.6. (Time Determinacy)
Whenever $\mathtt{P} \xrightarrow{\epsilon(d)} \mathtt{P}_1$ and $\mathtt{P} \xrightarrow{\epsilon(d)} \mathtt{P}_2$ then $\mathtt{P}_1 = \mathtt{P}_2$, where "=" is the syntactical equality.

Lemma 3.3.7. (Time Additive)
$\mathtt{P} \xrightarrow{\epsilon(c+d)} \mathtt{P}_2$ iff there exists $\mathtt{P}_1$ such that $\mathtt{P} \xrightarrow{\epsilon(c)} \mathtt{P}_1$ and $\mathtt{P}_1 \xrightarrow{\epsilon(d)} \mathtt{P}_2$.

Lemma 3.3.8. (Persistency)
If $\mathtt{P} \xrightarrow{\epsilon(d)} \mathtt{P}'$ and $\mathtt{P} \xrightarrow{\alpha} \mathtt{Q}$ for some $\mathtt{Q}$, then $\mathtt{P}' \xrightarrow{\alpha} \mathtt{Q}'$ for some $\mathtt{Q}'$

Next we outline the Real Time Laws.

1. Maximal Progress

   $\tau@t.\mathtt{P} \sim \tau.\mathtt{P}[0/t]$

   $\tau.\mathtt{P} + \epsilon(d).\mathtt{Q} \sim \tau.\mathtt{P}$

2. Time Determinacy

$$\epsilon(d).(\mathtt{P}+\mathtt{Q}) \sim \epsilon(d).\mathtt{P} + \epsilon(d).\mathtt{Q}$$

$$\epsilon(d).(\mathtt{P} \mid \mathtt{Q}) \sim \epsilon(d).\mathtt{P} \mid \epsilon(d).\mathtt{Q}$$

3. Time Additive

$$\epsilon(c+d).\mathtt{P} \sim \epsilon(c).\epsilon(d).\mathtt{P}$$

$$\epsilon(0).\mathtt{P} \sim \mathtt{P}$$

$$\epsilon(d).\mathtt{nil} \sim \mathtt{nil}$$

4. Persistency

$$\alpha@t.\mathtt{P} + \epsilon(d).\alpha@t.\mathtt{P}[t+d/t] \sim \alpha@t.\mathtt{P}$$

In *Timed CCS*, just like in *CCS*, strong bisimulation is a congruence.
To justify the usage of α-conversion, Yi (1991) presents the following proposition:

Proposition 3.3.9 (α-conversion). $\mu@t.\mathtt{E} \sim \mu@u.\mathtt{E}[u/t]$

Weak Bisimulation

The notion of weak bisimulation relaxes the conditions on invisible actions for *CCS* and *Timed CCS*. In the latter, this notion also relaxes conditions concerning the delay actions. To see this, let us begin by defining the transitions on which the notion of weak bisimulation for *Timed CCS* is based. According to Yi (1991):

Definition 3.3.10.

1. $\mathtt{P} \stackrel{\alpha}{\Rightarrow} \mathtt{Q}$ if $\mathtt{P}(\stackrel{\tau}{\rightarrow})^* \stackrel{\alpha}{\rightarrow} (\stackrel{\tau}{\rightarrow})^*\mathtt{Q}$
2. $\mathtt{P} \stackrel{\epsilon(d)}{\Rightarrow} \mathtt{Q}$ if $\mathtt{P}(\stackrel{\tau}{\rightarrow})^* \stackrel{\epsilon(d_1)}{\longrightarrow} (\stackrel{\tau}{\rightarrow})^* \cdots (\stackrel{\tau}{\rightarrow})^* \stackrel{\epsilon(d_n)}{\longrightarrow} (\stackrel{\tau}{\rightarrow})^*\mathtt{Q}$

   where $d = \sum_{i \leq n} d_i$

Here $(\stackrel{\tau}{\rightarrow})^*$ means that we can have zero or more transitions through τ. With this definition, we can build a labelled transition system $(\mathcal{P}, \boldsymbol{\epsilon}, \Rightarrow)$, where $\boldsymbol{\epsilon}$ is the set of actions excluding invisible ones, Yi (1991), and $\Rightarrow$ according to the *SOS* rules. Then, we are able to define weak bisimulation as follows:

Definition 3.3.11. A binary relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ is a weak bisimulation if $(\mathtt{P}, \mathtt{Q}) \in \mathcal{R}$ implies that for all $\zeta \in \boldsymbol{\epsilon}$,

1. $\forall \mathtt{P}' : \mathtt{P} \stackrel{\zeta}{\Rightarrow} \mathtt{P}' \supset \exists \mathtt{Q}' : \mathtt{Q} \stackrel{\zeta}{\Rightarrow} \mathtt{Q}'$ & $(\mathtt{P}', \mathtt{Q}') \in \mathcal{R}$
2. $\forall \mathtt{Q}' : \mathtt{Q} \stackrel{\zeta}{\Rightarrow} \mathtt{Q}' \supset \exists \mathtt{P}' : \mathtt{P} \stackrel{\zeta}{\Rightarrow} \mathtt{P}'$ & $(\mathtt{P}', \mathtt{Q}') \in \mathcal{R}$

where $\supset$ is another notation for implication and "&" is read as "and".

With this definition we are able to define weak equivalence between two processes.

Definition 3.3.12. Two processes $\mathtt{P}$ and $\mathtt{Q}$ are weak equivalent, written $\mathtt{P}\approx\mathtt{Q}$ iff there exists a weak bisimulation $\mathcal{R}$ such that $(\mathtt{P},\mathtt{Q})\in\mathcal{R}$.

Just like in *CCS*, weak bisimulation is not a congruence, because of the non-deterministic operator.

4

WHEN QUANTUM COMPUTING ENTERS INTO THE SCENE

4.1 QUANTUM COMPUTING IN A NUTSHELL

Richard Feynman idealized that to simulate nature we would need a quantum computer. In 1982 he said that: "Nature isn't classical, dammit, and if you want to make a simulation of nature, you'd better make it quantum mechanical, and by golly it's a wonderful problem, because it doesn't look so easy." [Feynman (1982)]

Since then, the interest in quantum computers arose together with the idea of *quantum advantage*: i.e. the idea that quantum computers can solve certain tasks much faster than classical computers. In 1992, David Deutsch and Richard Jozsa developed a quantum algorithm, Deutsch and Jozsa (1992), that indeed showed that quantum computers permit exponential speedups in certain tasks. In a nutshell, the Deutsch-Jozsa algorithm can verify if a function is either *balanced* or *constant*. In the former, the image has the same number of zeros and ones, and in the latter, the image is made only of zeros or ones. If one considers a function from the two-point set to the same set, a classical computer will need to perform two tests while in a quantum computer one test suffices to have the answer. Although this algorithm has no practical utility, it showed the remarkable power of quantum computers for certain tasks.

In 1994 Peter Shor developed an algorithm to efficiently factorize prime numbers on quantum computers, Shor (1994), which is a very hard task for classical computers. Using Shor's algorithm, a factorization of prime numbers that would take years for classical computers, can be achieved in just a few hours on a quantum computer. Clearly, this algorithm puts in risk the current encryption mechanisms.

In 1996, Knill proposed conventions for quantum pseucode, Knill (1996), and described the QRAM architecture. The latter establishes a notion of *master-slave* model between classical and quantum computers. In this setting the classical computer is the *master*: it manages the set of given tasks, selects the ones computationally hard and feeds them to the quantum computer.

Also in 1996, Lov Grover developed an algorithm for unstructured search, Grover (1996). With this, it was proved once again that for same problems, the quantum computer had better performance than a classical computer.

Twenty-three years later, the fight for quantum advantage still continues. Companies like IBM, Microsoft and Google are heavily competing for the first working quantum computers. Moreover, IBM provides cloud services for communication with their quantum computers.

Next, we present some notions related to quantum mechanics. For readers who want to deepen their knowledge on quantum computation, a good source is Nielsen and Chuang (2000).

4.1.1 *Dirac's Notation*

Dirac, Dirac (1981), created a notation that is used in quantum mechanics: vectors are written as a *ket*, $|\cdot\rangle$, and the conjugate transpose, denoted as $\dagger$, of a vector is written as a *bra*, $\langle\cdot|$. Due to their names, this notation is also known as the *braket* notation.

A *ket* can be represented by a column vector

$$|\psi\rangle = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix}$$

and the corresponding *bra* by a line vector of its complex conjugate

$$\langle\psi| = |\psi\rangle^\dagger = [c_1^* \cdots c_n^*]$$

where c^* is the complex conjugate of c.

Using the *braket* notation, the inner product of two vectors $|\psi\rangle$ and $|\phi\rangle$ is expressed as

$$\langle\psi|\phi\rangle = [c_1^* \cdots c_n^*] \cdot \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix} = \sum_{i=1}^{n} c_i^* \cdot d_i$$

4.1.2 *Hilbert Space*

The semantic structure underlying quantum computing and quantum information is that of an *Hilbert space* over the complex numbers, $\mathbb{C}$, as discussed in von Neumann and Beyer (1955).

An Hilbert space, denoted as $\mathcal{H}$, is a vector space with an inner product. The inner product is a mapping from $\mathcal{H} \times \mathcal{H} \to \mathbb{C}$ satisfying the following properties, Feng et al. (2012):

1. $\langle\psi|\psi\rangle \geq 0$ for any $|\psi\rangle \in \mathcal{H}$, with equality iff $|\psi\rangle = 0$
2. $\langle\phi|\psi\rangle = \langle\psi|\phi\rangle^*$
3. $\langle\phi|\sum_i c_i \cdot \psi_i\rangle = \sum_i c_i \cdot \langle\phi|\psi_i\rangle$

where $c \in \mathbb{C}$, c^* denotes the complex conjugate of c and $|\psi\rangle, |\phi\rangle$ are vectors in $\mathcal{H}$. Any two vectors, $|\psi\rangle$ and $|\phi\rangle$, are *orthogonal* if their inner product equals zero:

$$\langle\psi|\phi\rangle = 0$$

The norm of a vector is:

$$|||\psi\rangle|| = \sqrt{\langle\psi|\psi\rangle}$$

When the norm is equal to one the vector is said to be *normalized*.

An *orthonormal basis* in an Hilbert space is a set of vectors $|i\rangle$ normalized and orthogonal between them. The dimension of a Hilbert space is the number of vectors that compose a basis. For example, a Hilbert space over the complex numbers with a basis formed by two vectors has dimension two. If the basis is formed by n vectors, then the dimension of the Hilbert space is n, where n is a positive integer.

4.1.3 *Tensor Product*

The *tensor product*, denoted by $\otimes$, is a mathematical operation that concatenates vector spaces. For example, if one considers a vector space V with dimension n and another vector space W with dimension m, the tensor product of these two vector spaces, $V \otimes W$, is another vector space whose dimension, $dim(V \otimes W)$, is equal to $n \times m$. In this new vector space, a vector is a linear combination of the tensor product of vectors from V and W, $|v\rangle \otimes |w\rangle = |v \otimes w\rangle = |vw\rangle$. Notably, if $|i\rangle$ is a basis in V and $|j\rangle$ is a basis in W, then $|i \otimes j\rangle$ is a basis for $V \otimes W$. The tensor product has the following properties, Nielsen and Chuang (2000):

1. For an arbitrary scalar z and elements $|v\rangle$ of V and $|w\rangle$ of W,
$$z(|v\rangle \otimes |w\rangle) = (z|v\rangle) \otimes |w\rangle = |v\rangle \otimes (z|w\rangle)$$
2. For arbitrary $|v_1\rangle$ and $|v_2\rangle$ in V and $|w\rangle$ in W,
$$(|v_1\rangle + |v_2\rangle) \otimes |w\rangle = |v_1\rangle \otimes |w\rangle + |v_2\rangle \otimes |w\rangle$$
3. For arbitrary $|v\rangle$ in V and $|w_1\rangle$ and $|w_2\rangle$ in W,
$$|v\rangle \otimes (|w_1\rangle + |w_2\rangle) = |v\rangle \otimes |w_1\rangle + |v\rangle \otimes |w_2\rangle$$

The tensor product can also be applied to matrices, where it is known as the *Kronecker product*. Consider two matrices, A and B, with dimensions $m \times n$ and $p \times q$, respectively. The Kronecker product of these two matrices is:

$$A \otimes B = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{bmatrix} =$$

$$= \begin{bmatrix} a_{11} \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{bmatrix} & \cdots & a_{1n} \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{bmatrix} \\ \vdots & \ddots & \vdots \\ a_{m1} \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{bmatrix} & \cdots & a_{mn} \begin{bmatrix} b_{11} & \cdots & b_{1q} \\ \vdots & \ddots & \vdots \\ b_{p1} & \cdots & b_{pq} \end{bmatrix} \end{bmatrix}$$

The dimension of the resulting matrix is $m \times p$ by $n \times q$.

4.1.4 *Qubits*

Every physical system that does not interact with the environment, i.e. an isolated system, is associated with a Hilbert space which represents its *state space*. A physical system of great interest to us is the *qubit* which is represented through a two dimensional Hilbert space, $\mathcal{H}^2$. The set of vectors $\{|0\rangle, |1\rangle\}$ form the so-called *computational basis*, where

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The possible linear combinations of these basis vectors completely describe the state of a qubit, which in the general case is in *superposition*:

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

where $\alpha, \beta \in \mathbb{C}$ are known as probability amplitudes and the equation $|\alpha|^2 + |\beta|^2 = 1$ holds. The values $|\alpha|^2, |\beta|^2$ are the probabilities of measuring $|0\rangle$ or $|1\rangle$, respectively. A

n-qubit system corresponds to the tensor product of each qubit (seen as an Hilbert space) presented in the system:

$$\mathcal{H}^{2^n} = \bigotimes^{n} \mathcal{H}^2$$

An example of a n-qubit system is the two-qubit system. Here the Hilbert space is $\mathcal{H}^{2\times 2}$ and one of the possible basis is the set $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. A general state is then given by:

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$$

where $\alpha, \beta, \gamma, \delta \in \mathbb{C}$, $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$, and $|\alpha|^2$, $|\beta|^2$, $|\gamma|^2$ and $|\delta|^2$, are the probabilities of measuring $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$, respectively.

The states represented by multiple qubits are denominated either as *separable states* or as *entangled states*. The former can be written as a linear combination of tensor products, like for example

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle)$$

while the latter cannot. An example of an entangled state for two-qubit systems is the Bell state.

$$|\beta_{00}\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

For two-qubit systems, there are three more Bell states, which are further analyzed in Nielsen and Chuang (2000).

4.1.5 *Unitary Operators*

Unitary operators respect the following equation:

$$U^\dagger U = UU^\dagger = I$$

where $U^\dagger$ is the conjugate transpose or *Hermitian conjugate* of the operator U, and I is the identity operator.

In closed quantum systems, i.e. systems that do not interact with the environment, the evolution of a state is usually described via the notion of an unitary operator. As an example, consider a final state $|\phi\rangle$ obtained from an initial state $|\psi\rangle$ and a unitary operator U by application $U(|\psi\rangle) = U|\psi\rangle = |\phi\rangle$.

Some common unitary operators are the *Pauli operators* (I, X, Y, Z) and the *Hadamard* (H) for single qubits and the *controlled-NOT* ($CNot$) for two-qubit systems. On a matrix representation in the computational basis, these operators are written as:

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad CNot = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Next we will show how the single qubit operators affect the general state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ of a qubit:

$$\begin{aligned} I|\psi\rangle &= \alpha|0\rangle + \beta|1\rangle \\ X|\psi\rangle &= \alpha|1\rangle + \beta|0\rangle \\ Y|\psi\rangle &= i\alpha|1\rangle - i\beta|0\rangle \\ Z|\psi\rangle &= \alpha|0\rangle - \beta|1\rangle \\ H|\psi\rangle &= \frac{1}{\sqrt{2}}\left[\alpha(|0\rangle + |1\rangle) + \beta(|0\rangle - |1\rangle)\right] \end{aligned}$$

Note that the Hadamard operator creates superposition between the states $|0\rangle$ and $|1\rangle$ when just acting on one of them, for example $H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. Remarkably, it can also destroy the superposition, since by unitarity $HH|0\rangle = |0\rangle$. ID Quantique, IDQ (2016), makes use of this gate to generate random numbers.

The $CNot$ is a two-qubit system whose behaviour is similar to a controller, therefore the C in $CNot$. When applying a $CNot$ to a two-qubit system, we consider that the first qubit is the control qubit and that the second qubit is going to be changed. Therefore, when the control qubit is zero, the second qubit is not changed. Otherwise, if the control qubit is one, the second qubit is changed. As an example, consider the effect that a $CNot$ produces on a general two-qubit state $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ is as follows:

$$CNot|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|11\rangle + \delta|10\rangle$$

Now consider that we have a two-qubit system, $|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, and we want to apply a X operator on the first qubit and a Z operator on the second. To do that we make use of the tensor product.

$$
\begin{aligned}
X \otimes Z|\psi\rangle &= (X \otimes Z)\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \\
&= (X \otimes Z)\frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle) \\
&= \frac{1}{\sqrt{2}}[(X \otimes Z)|0\rangle \otimes |0\rangle + (X \otimes Z)|1\rangle \otimes |1\rangle] \\
&= \frac{1}{\sqrt{2}}[X|0\rangle \otimes Z|0\rangle + X|1\rangle \otimes Z|1\rangle] \\
&= \frac{1}{\sqrt{2}}(|10\rangle - |01\rangle)
\end{aligned}
$$

To simplify this process, one can label the qubits, using letters or numbers, and then put the label of the qubit that is going to be manipulated as a subscript in the operator, as follows:

$$X_1|\psi\rangle = X \otimes I|\psi\rangle = \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle)$$

4.1.6 *Measurement*

Measurement is an action that allow us to see how an isolated system has evolved through the action of unitary operations. Recall that there is a probability amplitude associated to each state. When one performs a measurement, the probability of measuring a state in the basis of the Hilbert space is given by the modulus square of the probability amplitude associated to this state.

In detail, a measurement is composed by a collection of *measurement operators*, $\{M_m\}$, where m indicates the measurement outcomes that may occur. The measurement operators must obey the *completeness equation*:

$$\sum_m M_m^\dagger M_m = I$$

The probability of m being the measurement outcome of a state $|\psi\rangle$ is given by:

$$p(m) = \langle\psi|M_m^\dagger M_m|\psi\rangle$$

The state of the system after the measurement is:

$$\frac{M_m|\psi\rangle}{\sqrt{\langle\psi|M_m^\dagger M_m|\psi\rangle}}$$

For a better understanding of how measurement works, consider the following example.

Example 4.1.1. In the next figure we have a two-qubit state $|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$ and all the possible outcomes of measuring first the left qubit and then the right qubit.

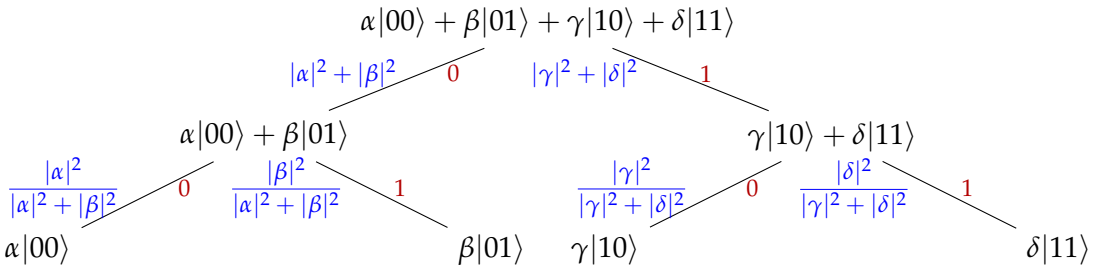


Figure 17: Measure scheme

In the Figure 17, in red we have the measurement outcome and in blue the probability of the measurement outcome. The states presented after the measurement are not normalized.

4.1.7 *Density Operators*

Another way to represent quantum states is by using *density operators* or *density matrices*.

While a pure state $|\psi\rangle$ is represented by the state vectors of a system, a density operator, denoted by ρ, can be written by an *ensemble of pure states*, $\{p_i, |\psi_i\rangle\}$, just as follows:

$$\rho = \sum_i p_i |\psi_i\rangle\langle\psi_i|$$

where p_i is the probability of being in the ensemble state $|\psi\rangle$.

A density operator thus can also represent a mixture of pure states. To distinguish a pure state from a mixed state, the following criteria must be used:

1. $tr(\rho^2) = 1 \rightarrow$ pure states
2. $tr(\rho^2) < 1 \rightarrow$ mixed states

where tr is the trace function that sums all the diagonal elements of a matrix A

$$tr(A) = \sum_{i=1}^{n} \langle i|A|i\rangle$$

in which i denotes a basis vector of A. Another fact about density operators is that one density operator can represent different state vectors, as shown in the following example.

Example 4.1.2. Consider two different state vectors $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|\phi\rangle = -|\psi\rangle$. Now let us construct the density operator of both state vectors. For $|\psi\rangle$ the density operator

is $\rho_1 = \frac{1}{2}(|0\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 0| + |1\rangle\langle 1|)$ and for $|\phi\rangle$ the density operator is $\rho_2 = \frac{1}{2}(|0\rangle\langle 0| + |0\rangle\langle 1| + |1\rangle\langle 0| + |1\rangle\langle 1|)$. Here, we see that $\rho_1 = \rho_2$, which means that two different state vectors can be represented by the same density operator.

Formally, a density operator is defined as follows, Nielsen and Chuang (2000):

Definition 4.1.3. An operator ρ is the density operator associated to some ensemble $\{p_i, |\psi_i\rangle\}$ iff it satisfies the conditions:

1. $tr(\rho) = 1$
2. ρ is a positive operator

where a positive operator is an operator with non-negative eigenvalues.

An unitary operator U acts on a density operator ρ as follows:

$$U\rho U^\dagger$$

Since a density operator can be written through an ensemble of pure states, we can represent the action of an unitary operator in terms of an ensemble of pure states

$$U\rho U^\dagger = U\sum_i p_i|\psi_i\rangle\langle\psi_i|U^\dagger = \sum_i p_i U|\psi_i\rangle\langle\psi_i|U^\dagger$$

If we simplify the notation, we obtain

$$U|\psi\rangle\langle\psi|U^\dagger$$

After a measurement, $M_m^\dagger \rho M_m$ the probability of m being the measurement outcome is given by Nielsen and Chuang (2000):

$$p(m) = tr(M_m^\dagger M_m \rho)$$

The state of the system after the measurement is:

$$\frac{M_m^\dagger \rho M_m}{tr(M_m^\dagger M_m \rho)}$$

A quantum system, described by a density operator, can be composed by others small systems, known as subsystems. In order to describe a subsystem, we need to perform the *partial trace* of the density operator, Nielsen and Chuang (2000).

Definition 4.1.4. Let ρ be a density operator of a system composed by two subsystems A and B, such that $\rho = \rho_A \otimes \rho_B$. The partial trace over the subsystem B is defined by:

$$tr_B(|a_1\rangle\langle a_2| \otimes |b_1\rangle\langle b_2|) = |a_1\rangle\langle a_2|tr(|b_1\rangle\langle b_2|)$$

where $|a_1\rangle$ and $|a_2\rangle$ are any two vectors in the state space of A, and $|b_1\rangle$ and $|b_2\rangle$ are any two vectors in the state space of B.

With this, the notion of *reduced density operator* will be introduced by an example.

Example 4.1.5. Consider a quantum system composed by two entangled subsystems where the first qubit represents subsystem A and the second qubit subsystem B:

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

The density operator of $|\Phi^+\rangle$ is:

$$\rho = \frac{1}{2}(|00\rangle\langle 00| + |00\rangle\langle 11| + |11\rangle\langle 00| + |11\rangle\langle 11|)$$

In order to obtain the reduced density matrix to the subsystem A we need to perform a partial trace operation with respect to subsystem B, as follows:

$$\begin{aligned}\rho^A &= tr_B(\rho) = \\ &= \frac{1}{2}(|0\rangle\langle 0|\langle 0|0\rangle + |0\rangle\langle 1|\langle 0|1\rangle + |1\rangle\langle 0|\langle 1|0\rangle + |1\rangle\langle 1|\langle 1|1\rangle) = \\ &= \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)\end{aligned}$$

With this example we wanted to show how the reduced density operators are calculated and that when we calculate the reduced density matrix of an entangled state, information is lost. On the other hand, if the above density operator represented a separable state, no information would have been lost.

4.1.8 *Super-operators*

On an isolated system, the evolution of a system can be described by unitary operators. But is this still valid for open systems? The answer is no. To describe the evolution of an open system we need to consider *super-operators*. According to Ying et al. (2009), a "super-operator on a Hilbert space $\mathcal{H}$ is a linear operator ε from the space of linear operators on $\mathcal{H}$ into itself" satisfying the following two conditions:

1. $tr[\varepsilon(\rho)] \leq tr(\rho)$ for each $\rho \in \mathcal{D}(\mathcal{H})$
2. Complete positivity: for any extra Hilbert space $\mathcal{H}_R$, $(\mathcal{I}_R \otimes \varepsilon)(A)$ is positive provided A is a positive operator on $\mathcal{H}_R \otimes \mathcal{H}$, where $\mathcal{I}_R$ is the identity operation on $\mathcal{H}_R$

where $\mathcal{D}(\mathcal{H})$ is the set of partial density operators in $\mathcal{H}$, with a partial density operator being a positive operator, ρ, with $tr(\rho) \leq 1$. When the first clause yields $tr[\varepsilon(\rho)] = tr(\rho)$, for all $\rho \in \mathcal{D}(\mathcal{H})$, then ε is trace-preserving. Intuitively, a super-operator can be understood as an operator that acts on density operators.

4.2 A survey of quantum process algebras

Quantum Mechanics is a recent field (it emerged in the beginning of the last century) that sees the world as a quantization (physical properties, such as energy, are studied via discrete methods). Concurrent Systems and Process Algebra also appeared in the last century but around the 60s (70s) and they are a field of Computer Science. The first studies systems that interact with the environment and communicate with other systems. The second studies concurrent systems in an algebraic way.

It is normal to perform/recreate in a quantum setting what was successful classically and because of that quantum process algebras were created in the quantum domain as well. These include *CQP* (Communicating Quantum Processes, Gay and Nagarajan (2005)), *QPAlg* (Quantum Process Algebra, Lalire and Jorrand (2004)) and *qCCS* (Quantum Calculus of Communicating Systems, Ying et al. (2009)). *CQP* gives a formal model to analyse quantum communicating protocols and most of these protocols involve the transmission of both quantum and classical data. *QPAlg* aims to study the coordination between classical and quantum computation. With this description it is possible to see that *CQP* and *QPAlg* handle classical information. Because of that, it might be better to call them Hybrid Quantum Process Algebras, where the word hybrid comes from the fact that they need to interact with both the quantum and classical worlds. The one that is missing, *qCCS*, is the one that is going to be thoroughly analysed in this dissertation. So why did we choose *qCCS*? *qCCS* is a process algebra that only cares about pure quantum processes, discarding everything that is classical. In addition, *qCCS* is a quantum version of *CCS* (the process algebra that was discussed in the previous chapters). These are the two reasons we chose *qCCS*.

4.3 Quantum calculus of communicating systems

Quantum Calculus of Communicating Systems, Feng et al. (2007), is a process algebra whose goal is to extend *CCS* with quantum input/output, quantum operators and measurements. This version takes into account classical and quantum variables. Later on, Mingsheng Ying *et al*, Ying et al. (2009), developed a purely quantum version of *qCCS*, in order to develop a tool to reason about computation and communication in quantum systems. Unlike the first version of *qCCS*, Feng et al. (2007), this one only takes into account

quantum variables. In Feng et al. (2012) the two previous versions of *qCCS* were combined to develop a version that harbours classical and quantum data.

In this section we will outline the syntax, operational semantics and a notion of behavioural equivalence of *qCCS*, Ying et al. (2009).

4.3.1 *Syntax*

From now on it will be assumed that *Chan* is the set of names for quantum channels, *Var* is the set of names for quantum variables, τ represents the silent action, $\mathcal{H}$ is a Hilbert space, $\mathcal{P}$ is the set of quantum processes and $fv(\mathtt{P})$ is the set of free quantum variables in a process $\mathtt{P} \in \mathcal{P}$.

The quantum variables in *Var* are used to represent quantum systems in the Hilbert space. For example, if $x \in Var$, then there is a state space for the x-system that is represented by a finite-dimensional Hilbert space, denoted as $\mathcal{H}_x$. Furthermore, we can have a big quantum system composed of all the x-systems, defined by:

$$\mathcal{H}_X = \bigotimes_{x \in X} \mathcal{H}_x$$

for any $X \subseteq Var$. Note that $\mathcal{H}_{Var}$ is a countably infinite-dimensional Hilbert space because of the tensor products between finite-dimensional Hilbert spaces. In what follows, typically when we write $\mathcal{H}$ we mean $\mathcal{H}_{Var}$.

In $\mathcal{P}$ there exists a set of process constant schemes. A process constant scheme has associated to it a non-negative arity. So, if $\mathtt{P}$ is a process constant and $\tilde{x} = x_1, \cdots, x_{ar(\mathtt{P})} \in Var$, then $\mathtt{P}(\tilde{x})$ is called a process constant.

Next, the formal syntax of *qCCS* will be presented. Note that, in Section 4.5 we make a comparison between *CCS* and *qCCS*.

Definition 4.3.1. Quantum processes are defined inductively by the following formation rules:

1. each process constant $\mathtt{A}(\tilde{x})$ is in $\mathcal{P}$ and $fv(\mathtt{A}(\tilde{x})) = \{\tilde{x}\}$
2. $\mathtt{nil} \in \mathcal{P}$ and $fv(\mathtt{nil}) = \emptyset$
3. if $\mathtt{P} \in \mathcal{P}$, then $\tau.\mathtt{P} \in \mathcal{P}$ and $fv(\tau.\mathtt{P}) = fv(\mathtt{P})$
4. if $\mathtt{P} \in \mathcal{P}$, X is a finite subset of *Var*, and ε is a super-operator on $\mathcal{H}_X$, then $\varepsilon[X].\mathtt{P} \in \mathcal{P}$ and $fv(\varepsilon[X].\mathtt{P}) = fv(\mathtt{P}) \cup X$
5. if $\mathtt{P} \in \mathcal{P}$, then $c?x.\mathtt{P} \in \mathcal{P}$, and $fv(c?x.\mathtt{P}) = fv(\mathtt{P}) - \{x\}$
6. if $\mathtt{P} \in \mathcal{P}$ and $x \notin fv(\mathtt{P})$, then $c!x.\mathtt{P} \in \mathcal{P}$, and $fv(c!x.\mathtt{P}) = fv(\mathtt{P}) \cup \{x\}$

7. if $\mathtt{P}, \mathtt{Q} \in \mathcal{P}$, then $\mathtt{P} + \mathtt{Q} \in \mathcal{P}$ and $fv(\mathtt{P} + \mathtt{Q}) = fv(\mathtt{P}) \cup fv(\mathtt{Q})$
8. if $\mathtt{P}, \mathtt{Q} \in \mathcal{P}$ and $fv(\mathtt{P}) \cap fv(\mathtt{Q}) = \emptyset$, then $\mathtt{P} \mid\mid \mathtt{Q} \in \mathcal{P}$ and $fv(\mathtt{P} \mid\mid \mathtt{Q}) = fv(\mathtt{P}) \cup fv(\mathtt{Q})$
9. if $\mathtt{P} \in \mathcal{P}$ and $\mathtt{L} \subseteq \mathit{Chan}$, then $\mathtt{P}\backslash\mathtt{L} \in \mathcal{P}$ and $fv(\mathtt{P}\backslash\mathtt{L}) = fv(\mathtt{P})$

Note that, every clause in Definition 4.3.1 includes a calculation of quantum free variables. This is done in order to control the quantum variables that are presented in processes and in particular to forbid the violation of the no-cloning theorem. This theorem is a restriction imposed by quantum mechanics that says that it is not possible to clone a state. The majority of the clauses are mostly identical to the ones in *CCS* syntax; the only differences are in clause 4 and in the conditions in clauses 6 ($x \notin fv(\mathtt{P})$) and 8 ($fv(\mathtt{P}) \cap fv(\mathtt{Q}) = \emptyset$). The first one is new: it allows to perform quantum operations on systems and the others are related with the no-cloning theorem.

Let us focus on the clauses 5 and 6 concerning the reception and emission of data on quantum channels, respectively. When receiving information ($c?x$), the quantum variable x is eliminated from the set of quantum free variables and becomes bound. On the other hand, when a quantum variable is sent ($c!x$), it becomes a free variable because the variable will be bound by the channel that is going to receive it.

We assume that each process constant has a defining equation:

$$\mathtt{A}(\tilde{x}) \stackrel{def}{=} \mathtt{P}$$

where $\mathtt{P}$ is a process with $fv(\mathtt{P}) \subseteq \{\tilde{x}\}$.

Another feature presented in *qCCS* is the substitution of variables names. To perform this action, caution is needed due the impossibility of cloning states in quantum mechanics. Mingsheng Ying *et al*, Ying et al. (2009), define substitution of quantum variables and give an explanation about it. For those interested, we recommend consulting the *op. cit.* In this document, we will only present the final result.

Definition 4.3.2. For any $\mathtt{P} \in \mathcal{P}$ and substitution f, $\mathtt{P}f$ is defined recursively as follows:

1. if $\mathtt{P}$ is a process constant $\mathtt{A}(x_1, \cdots, x_n)$ then
$$\mathtt{P}f = \mathtt{A}(f(x_1), \cdots, f(x_n))$$
2. if $\mathtt{P} = \mathtt{nil}$ then $\mathtt{P}f = \mathtt{nil}$
3. if $\mathtt{P} = \tau.\mathtt{P}'$ then $\mathtt{P}f = \tau.\mathtt{P}'(f)$
4. if $\mathtt{P} = \varepsilon[X].\mathtt{P}'$ then $\mathtt{P}f = (\varepsilon f)[f(X)].\mathtt{P}'(f)$
5. if $\mathtt{P} = c?x.\mathtt{P}'$ then $\mathtt{P}f = c?y.\mathtt{P}'\{y/x\}(f_y)$, where $y \notin fv(c?x.\mathtt{P}') \cup fv(\mathtt{P}'f)$, and f_y is the substitution with $f_y(y) = y$, $f_y(f^{-1}(y)) = f(y)$ and $f_y(z) = f(z)$ for all $z \neq y$ and $z \neq f^{-1}(y)$

6. if $\mathtt{P} = c!x.\mathtt{P}'$ then $\mathtt{P}f = c!f(x).\mathtt{P}'(f)$
7. if $\mathtt{P} = \mathtt{P}_1 + \mathtt{P}_2$ then $\mathtt{P}f = \mathtt{P}_1(f) + \mathtt{P}_2(f)$
8. if $\mathtt{P} = \mathtt{P}_1 \mid\mid \mathtt{P}_2$ then $\mathtt{P}f = \mathtt{P}_1(f) \mid\mid \mathtt{P}_2(f)$
9. if $\mathtt{P} = \mathtt{P}'\backslash \mathtt{L}$ then $\mathtt{P}f = \mathtt{P}'(f)\backslash \mathtt{L}$

The clause 5 is the most difficult to understand. Recall that it is not possible to clone, and through the syntax rules of *qCCS*, when a channel receives a quantum variable the latter becomes bounded. Now consider the processes in clause 5. When the substitution is applied to $\mathtt{P}$, all the occurrences of x in $\mathtt{P}$ must be changed to y, in order to respect the no-cloning theorem. This explains the fact that $c?x$ became $c?y$ and the fact that f_y does not include x. With all this, the new bounded variable became y, and the *qCCS* syntax is respected.

4.3.2 *Semantics*

The operational semantics of *qCCS* is composed by configurations and transitions between them labelled by actions. A configuration is a pair $\langle \mathtt{P}, \rho \rangle$, where $\mathtt{P} \in \mathcal{P}$ is a process and $\rho \in D(\mathcal{H})$ is a partial density operator, which specifies the actual state of the environment. Here, a partial density operator is a positive operator, ρ, with $tr(\rho) \leq 1$. $D(\mathcal{H})$ represents the set of partial density operators on $\mathcal{H}$. The set of configurations is represented by *Con*, and *Act* is the set of actions composed of

$$Act = \{\tau\} \cup Act_{op} \cup Act_{com}$$

where the set of quantum operations is

$$Act_{op} = \{\varepsilon[X] : \ X \text{ is a finite subset of } Var \text{ and } \varepsilon \text{ is a super-operator on } \mathcal{H}_X\}$$

and the set of communication actions is

$$Act_{com} = \{c?x, c!x : c \in Chan \text{ and } x \in Var\}$$

We will also need the following notation:

- for each $\alpha \in Act$, we use $cn(\alpha)$ to stand for the channel name in action α; thus $cn(c?x) = cn(c!x) = c$, and $cn(\tau) = \emptyset$ and $cn(\varepsilon[X]) = \emptyset$;
- $fv(\alpha)$ is the set of free variables in α; thus, $fv(c!x) = \{x\}$, $fv(\varepsilon[X]) = X$ and $fv(\tau) = fv(c?x) = \emptyset$;

- $bv(\alpha)$ is the set of bounded variable in α; thus, $bv(c?x) = x$ and $bv(\tau) = \emptyset$, $bv(\varepsilon[X]) = \emptyset$ and $bv(c!x) = \emptyset$.

We will next present the operational semantics of *qCCS* as a transition system (*Con*, *Act*, $\rightarrow$), where the transition relation $\rightarrow$ is defined by the following rules:

$$\textbf{Tau:}\quad \frac{}{\langle\, \tau.\mathtt{P}, \rho \,\rangle \xrightarrow{\tau} \langle\, \mathtt{P}, \rho \,\rangle}$$

$$\textbf{Oper:}\quad \frac{}{\langle\, \varepsilon[X].\mathtt{P}, \rho \,\rangle \xrightarrow{\varepsilon[X]} \langle\, \mathtt{P}, \varepsilon_X(\rho) \,\rangle}$$

$$\textbf{Out:}\quad \frac{}{\langle\, c!x.\mathtt{P}, \rho \,\rangle \xrightarrow{c!x} \langle\, \mathtt{P}, \rho \,\rangle}$$

$$\textbf{In:}\quad \frac{}{\langle\, c?x.\mathtt{P}, \rho \,\rangle \xrightarrow{c?y} \langle\, \mathtt{P}\{y/x\}\,, \rho\rangle}\ y \notin fv(c?x.\mathtt{P})$$

$$\textbf{Sum:}\quad \frac{\langle\, \mathtt{P}, \rho \,\rangle \xrightarrow{\alpha} \langle\, \mathtt{P}', \rho' \,\rangle}{\langle\, \mathtt{P} + \mathtt{Q}, \rho \,\rangle \xrightarrow{\alpha} \langle\, \mathtt{P}', \rho' \,\rangle}$$

$$\textbf{Intl1:}\quad \frac{\langle\, \mathtt{P}, \rho \,\rangle \xrightarrow{c?x} \langle\, \mathtt{P}', \rho' \,\rangle}{\langle\, \mathtt{P} \,||\, \mathtt{Q}, \rho \,\rangle \xrightarrow{c?x} \langle\, \mathtt{P}' \,||\, \mathtt{Q}, \rho' \,\rangle}\ x \notin fv(\mathtt{Q})$$

$$\textbf{Intl2:}\quad \frac{\langle\, \mathtt{P}, \rho \,\rangle \xrightarrow{\alpha} \langle\, \mathtt{P}', \rho' \,\rangle}{\langle\, \mathtt{P} \,||\, \mathtt{Q}, \rho \,\rangle \xrightarrow{\alpha} \langle\, \mathtt{P}' \,||\, \mathtt{Q}, \rho' \,\rangle}\ \alpha \text{ not input}$$

$$\textbf{Res:}\quad \frac{\langle\, \mathtt{P}, \rho \,\rangle \xrightarrow{\alpha} \langle\, \mathtt{P}', \rho' \,\rangle}{\langle\, \mathtt{P}\backslash\mathtt{L}, \rho \,\rangle \xrightarrow{\alpha} \langle\, \mathtt{P}'\backslash\mathtt{L}, \rho' \,\rangle}\ cn(\alpha) \notin \mathtt{L}$$

$$\textbf{Def:}\quad \frac{\langle\, \mathtt{P}\{\tilde{y}/\tilde{x}\}, \rho \,\rangle \xrightarrow{\alpha} \langle\, \mathtt{P}', \rho' \,\rangle}{\langle\, \mathtt{A}(\tilde{y}), \rho \,\rangle \xrightarrow{\alpha} \langle\, \mathtt{P}', \rho' \,\rangle}\ \mathtt{A}(\tilde{x}) \overset{def}{=} \mathtt{P}$$

$$\textbf{Comm:}\quad \frac{\langle\, \mathtt{P}, \rho \,\rangle \xrightarrow{c?x} \langle\, \mathtt{P}', \rho \,\rangle \quad \langle\, \mathtt{Q}, \rho \,\rangle \xrightarrow{c!x} \langle\, \mathtt{Q}', \rho \,\rangle}{\langle\, \mathtt{P} \,||\, \mathtt{Q}, \rho \,\rangle \xrightarrow{\tau} \langle\, \mathtt{P}' \,||\, \mathtt{Q}', \rho \,\rangle}$$

Table 3: *qCCS SOS* rules

In Table 3, the symmetric forms of **Sum**, **Intl1**, **Intl2** and **Comm** are omitted. Through the rules **Out**, **In** and **Comm** one can conclude that the density operator information is exchanged. In the **Out** rule, unlike the **In** rule, the x-system is sent through the channel c. In

the latter, the x variable refers to the place where the received system will go. Furthermore, in the **In** rule, the side condition serves to avoid conflict of names.

The justification for the side conditions in **Intl1** and **Intl2**, and the fact that the **Comm** does not have a side condition is presented in Lemma 3.2 of Ying et al. (2009). Moreover, Section 3.4 of Ying et al. (2009) presents the properties of the transitions defined on Table 3.

4.3.3 *Behavioural Equivalence*

In *CCS*, the behavioural equivalence only considers processes. But, the semantics of *qCCS* uses pairs made of processes and states, which are denoted as configurations. Therefore a suitable notion of equivalence must pay attention not merely to processes but also to configurations. Having that in mind, the notion of strong bisimulation in *qCCS*, Ying et al. (2009), is presented next.

Definition 4.3.3. A symmetric relation $\mathcal{R} \subseteq \text{Con} \times \text{Con}$ is called a strong bisimulation if for any $\langle \mathtt{P},\rho\rangle$, $\langle \mathtt{Q},\sigma\rangle \in Con$, $\langle \mathtt{P},\rho\rangle \mathcal{R} \langle \mathtt{Q},\sigma\rangle$ implies,

1. whenever $\langle \mathtt{P},\rho\rangle \xrightarrow{\alpha} \langle \mathtt{P}',\rho'\rangle$ and α is not an input, then for some $\mathtt{Q}'$ and σ', $\langle \mathtt{Q},\sigma\rangle \xrightarrow{\alpha} \langle \mathtt{Q}',\sigma'\rangle$ and $\langle \mathtt{P}',\rho'\rangle \mathcal{R} \langle \mathtt{Q}',\sigma'\rangle$
2. whenever $\langle \mathtt{P},\rho\rangle \xrightarrow{c?x} \langle \mathtt{P}',\rho\rangle$ and $x \notin fv(\mathtt{P}) \cup fv(\mathtt{Q})$, then for some $\mathtt{Q}'$, $\langle \mathtt{Q},\sigma\rangle \xrightarrow{c?x} \langle \mathtt{Q}',\sigma\rangle$ and for all $y \notin fv(\mathtt{P}') \cup fv(\mathtt{Q}') - \{x\}$, $\langle \mathtt{P}'\{y/x\},\rho\rangle \mathcal{R} \langle \mathtt{Q}'\{y/x\},\sigma\rangle$
3. whenever $\langle \mathtt{Q},\sigma\rangle \xrightarrow{\alpha} \langle \mathtt{Q}',\sigma'\rangle$ and α is not an input, then for some $\mathtt{P}'$ and σ', $\langle \mathtt{P},\rho\rangle \xrightarrow{\alpha} \langle \mathtt{P}',\rho'\rangle$ and $\langle \mathtt{P}',\rho'\rangle \mathcal{R} \langle \mathtt{Q}',\sigma'\rangle$
4. whenever $\langle \mathtt{Q},\sigma\rangle \xrightarrow{c?x} \langle \mathtt{Q}',\sigma\rangle$ and $x \notin fv(\mathtt{P}) \cup fv(\mathtt{Q})$, then for some $\mathtt{P}'$, $\langle \mathtt{P},\rho\rangle \xrightarrow{c?x} \langle \mathtt{P}',\rho\rangle$ and for all $y \notin fv(\mathtt{P}') \cup fv(\mathtt{Q}') - \{x\}$, $\langle \mathtt{P}'\{y/x\},\rho\rangle \mathcal{R} \langle \mathtt{Q}'\{y/x\},\sigma\rangle$

In the second and fourth clauses, the condition $y \notin fv(\mathtt{P}') \cup fv(\mathtt{Q}') - \{x\}$ is necessary to prevent that two different quantum systems become the same.

Definition 4.3.4. For any $\langle \mathtt{P},\rho\rangle, \langle \mathtt{Q},\sigma\rangle \in Con$, we say that $\langle \mathtt{P},\rho\rangle$ and $\langle \mathtt{Q},\sigma\rangle$ are strongly bisimilar, written $\langle \mathtt{P},\rho\rangle \sim \langle \mathtt{Q},\sigma\rangle$, if $\langle \mathtt{P},\rho\rangle \mathcal{R} \langle \mathtt{Q},\sigma\rangle$ for some strong bisimulation $\mathcal{R}$; that is, strong bisimilarity on *Con* is the greatest strong bisimulation:

$$\sim = \bigcup \{\mathcal{R} : \mathcal{R} \textit{ is a strong bisimulation}\}$$

When two processes are in the same environment, we can compare the processes.

Definition 4.3.5. For any quantum processes $\mathtt{P}$, $\mathtt{Q} \in \mathcal{P}$, we say that $\mathtt{P}$ and $\mathtt{Q}$ are strongly bisimilar, written $\mathtt{P} \sim \mathtt{Q}$, if $\langle \mathtt{P}, \rho\rangle \sim \langle \mathtt{Q}, \rho\rangle$ for all $\rho \in \mathcal{D}(\mathcal{H})$.

The proof of this result, as well as the ones that immediatly follow are in Ying et al. (2009).

For a recursive characterization of strong bisimilarity between configurations, the following lemma is presented.

Lemma 4.3.6. For any $\langle \mathtt{P}, \rho\rangle$,$\langle \mathtt{Q}, \sigma\rangle \in Con$, $\langle \mathtt{P}, \rho\rangle \sim \langle \mathtt{Q}, \sigma\rangle$ if and only if:

1. whenever $\langle \mathtt{P}, \rho\rangle \xrightarrow{\alpha} \langle \mathtt{P}', \rho'\rangle$ and α is not an input, then for some $\mathtt{Q}'$ and σ', $\langle \mathtt{Q}, \sigma\rangle \xrightarrow{\alpha} \langle \mathtt{Q}', \sigma'\rangle$ and $\langle \mathtt{P}', \rho'\rangle \sim \langle \mathtt{Q}', \sigma'\rangle$
2. whenever $\langle \mathtt{P}, \rho\rangle \xrightarrow{c?x} \langle \mathtt{P}', \rho\rangle$ and $x \notin fv(\mathtt{P}) \cup fv(\mathtt{Q})$, then for some $\mathtt{Q}'$, $\langle \mathtt{Q}, \sigma\rangle \xrightarrow{c?x} \langle \mathtt{Q}', \sigma\rangle$ and for all $y \notin fv(\mathtt{P}') \cup fv(\mathtt{Q}') - \{x\}$, $\langle \mathtt{P}'\{y/x\}, \rho\rangle \sim \langle \mathtt{Q}'\{y/x\}, \sigma\rangle$

and the symmetric forms of 1 and 2

Recalling Definition 4.3.5, we present below the generalization to *qCCS* of the monoid and static laws of *CCS*.

Proposition 4.3.7. For any $\mathtt{P}$, $\mathtt{Q}$, $\mathtt{R} \in \mathcal{P}$, and $\mathtt{K}$, $\mathtt{L} \subseteq Chan$:

1. $\mathtt{P} + \mathtt{Q} \sim \mathtt{Q} + \mathtt{P}$
2. $\mathtt{P} + (\mathtt{Q} + \mathtt{R}) \sim (\mathtt{P} + \mathtt{Q}) + \mathtt{R}$
3. $\mathtt{P} + \mathtt{P} \sim \mathtt{P}$
4. $\mathtt{P} + \mathtt{nil} \sim \mathtt{P}$
5. $\mathtt{P} \mid\mid \mathtt{Q} \sim \mathtt{Q} \mid\mid \mathtt{P}$
6. $\mathtt{P} \mid\mid (\mathtt{Q} \mid\mid \mathtt{R}) \sim (\mathtt{P} \mid\mid \mathtt{Q}) \mid\mid \mathtt{R}$
7. $\mathtt{P} \mid\mid \mathtt{nil} \sim \mathtt{P}$
8. $\mathtt{P}\backslash\mathtt{L} \sim \mathtt{P}$ if $cn(\mathtt{P}) \cap \mathtt{L} = \emptyset$, where $cn(\mathtt{P})$ is the set of free channel names in $\mathtt{P}$
9. $(\mathtt{P}\backslash\mathtt{K})\backslash\mathtt{L} \sim \mathtt{P}\backslash(\mathtt{K} \cup \mathtt{L})$

The expansion law, presented in *CCS*, can also be generalized here. Recall that this law tells that we can separate the actions performed synchronously form the ones executed asynchronously.

Proposition 4.3.8. (Expansion law) For any $\mathtt{P}$, $\mathtt{Q} \in \mathcal{P}$:

$$\begin{aligned}(\mathtt{P} \mid\mid \mathtt{Q})\backslash\mathtt{L} \sim & \sum\{\alpha.(\mathtt{P}' \mid\mid \mathtt{Q})\backslash\mathtt{L} : \mathtt{P} \xrightarrow{\alpha} \mathtt{P}' \text{ and } cn(\alpha) \notin \mathtt{L}\} \\ & + \sum\{\alpha.(\mathtt{P} \mid\mid \mathtt{Q}')\backslash\mathtt{L} : \mathtt{Q} \xrightarrow{\alpha} \mathtt{Q}' \text{ and } cn(\alpha) \notin \mathtt{L}\} \\ & \sum\{\tau.(\mathtt{P}' \mid\mid \mathtt{Q}')\backslash\mathtt{L} : \mathtt{P} \xrightarrow{c?x} \mathtt{P}' \text{ and } \mathtt{Q} \xrightarrow{c!x} \mathtt{Q}', \\ & \text{or } \mathtt{P} \xrightarrow{c!x} \mathtt{P}' \text{ and } \mathtt{Q} \xrightarrow{c?x} \mathtt{Q}'\}\end{aligned}$$

Next, we see that strong bisimulation is a congruence.

Theorem 4.3.9. 1. If $\mathtt{A} \overset{def}{=} \mathtt{P}$ then $\mathtt{A} \sim \mathtt{P}$

2. If $\mathtt{P} \sim \mathtt{Q}$, then:

   (a) $\tau.\mathtt{P} \sim \tau.\mathtt{Q}$

   (b) $\varepsilon[X].\mathtt{P} \sim \varepsilon[X].\mathtt{Q}$

   (c) $c!x.\mathtt{P} \sim c!x.\mathtt{Q}$

   (d) $c?x.\mathtt{P} \sim c?x.\mathtt{Q}$

   (e) $\mathtt{P} + \mathtt{R} \sim \mathtt{Q} + \mathtt{R}$

   (f) $\mathtt{P} \,||\, \mathtt{R} \sim \mathtt{Q} \,||\, \mathtt{R}$

   (g) $\mathtt{P}\backslash\mathtt{L} \sim \mathtt{Q}\backslash\mathtt{L}$

Next, we illustrate via an example that if two configurations are related through strong bisimilarity then the underlying processes have exactly the same transitions.

Example 4.3.10. Consider the following configurations:

$$\langle \mathtt{P}, \rho \rangle; \langle \mathtt{Q}, \rho \rangle; \langle \mathtt{R}, \rho \rangle$$

where $\mathtt{P} = c?x.c!x.\mathtt{nil}$, $\mathtt{Q} = \mathtt{P} + \mathtt{nil}$ and $\mathtt{R} = c?x.c!x.\tau.\mathtt{nil}$.

It is easy to see that $\mathtt{Q} \sim \mathtt{P}$, since $\mathtt{Q} = \mathtt{P} + \mathtt{nil}$ and by the item four in Proposition 4.3.7 $\mathtt{P} + \mathtt{nil} \sim \mathtt{P}$. Therefore $\mathtt{Q} \sim \mathtt{P}$. However, it is not possible to say that $\mathtt{P} \sim \mathtt{R}$ or $\mathtt{Q} \sim \mathtt{R}$, because the process $\mathtt{R}$ has a transition by τ that it is not presented in $\mathtt{P}$ nor $\mathtt{Q}$.

Just as in *CCS*, the notion of strong bisimulation is too strong and, in particular, it does not treat invisible action in a suitable way. In order to solve this problem a weaker notion of equivalence is shown in Ying et al. (2009).

4.3.4 *Recursion*

Recursion is a very important feature in process algebras because it allows to specify iterative behaviour, such as while-loops. In classical computation recursion is a familiar notion, but in the quantum context the no-cloning theorem raises precautions. In the following example, inspired from Ying et al. (2009), we aim to present a problem that can arise when dealing with recursion in quantum.

Example 4.3.11. Consider the following defining equation that describes a process constant scheme $\mathtt{A}(x)$.

$$\mathtt{A}(x) \overset{def}{=} c!x.\mathtt{A}(x)$$

To see if $\mathtt{A}(x)$ is valid let us calculate the free variables.

$$fv(\mathtt{A}(x)) = \{x\}$$
$$fv(c!x.\mathtt{A}(x)) \rightarrow \text{does not respect the } qCCS \text{ syntax}$$

Recalling the clause four of Definition 4.3.1 it is possible to see that the precondition $x \notin fv(\mathtt{A}(x))$ is violated and consequently $\mathtt{A}(x) \stackrel{def}{=} c!x.\mathtt{A}(x)$ is not valid. Therefore $\mathtt{A}(x) \notin \mathcal{P}$.

Consider now the following defining equation, very similar to the previous but only with a simple and very important change in the variables on the process constant scheme.

$$\mathtt{A}(y) \stackrel{def}{=} c?x.c!x.\mathtt{A}(y)$$

Calculating the free variables.

$$fv(\mathtt{A}(y)) = \{y\}$$
$$fv(c?x.c!x.\mathtt{A}(y)) = fv(c!x.\mathtt{A}(y)) - \{x\}$$
$$fv(c!x.\mathtt{A}(y)) = fv(\mathtt{A}(y)) \cup \{x\} = \{y\} \cup \{x\} = \{x, y\}$$
$$fv(c!x.\mathtt{A}(y)) - \{x\} = \{x, y\} - \{x\} = \{y\} \subseteq fv(\mathtt{A}(y)) = \{y\}$$

In the third step, the precondition on the fourth clause of Definition 4.3.1 is respected and the condition of a defining equation is obeyed, therefore the process is valid.

Is important to emphasize that recursion in *qCCS* is made through defining equations of processes constant schemes.

4.4 A TYPING SYSTEM FOR QCCS

In this section we present a typing system for *qCCS* developed by us, as the first original contribution of the disseration. The typing system was inspired by *CQP*, Gay and Nagarajan (2006), that is a quantum process algebra which has as a prominent feature a static type system.

With a typing system we have the capacity to evaluate if the no-cloning theorem is being respected and if the actions performed throughout the process are well-typed, this is, there are typing rules that allows to derive the execution of the actions. Furthermore, with a typing system we have a tool to study typical derivation properties such that the weakening property.

The typing system is composed by a set of rules. A rule is composed by premises and a conclusion. In order to obtain a conclusion, all the corresponding premises must hold. A rule is represented as follows, Plotkin (1981):

$$\frac{Premises}{Conclusion}$$

We will also use the following format, Gay and Nagarajan (2006):

$$\Gamma \vdash \mathtt{P}$$

which reads as follows: the process $\mathtt{P}$ is valid and well-typed in environment Γ. The environment Γ is a set of variables, and $\mathtt{P}$ is to be understood as an (unrestricted) expression over the grammar underlying the *qCCS* processes. In the typing system, (Γ, x) is interpreted as $(\Gamma \cup \{x\} \wedge x \notin \Gamma)$. Table 4 presents the typing rules.

$$\text{(NIL)} \quad \Gamma \vdash \mathtt{nil} \qquad \text{(CONST)} \quad \frac{\Delta \subseteq \Gamma}{\Gamma \vdash A(\Delta)}$$

$$\text{(INV)} \quad \frac{\Gamma \vdash \mathtt{P}}{\Gamma \vdash \tau.\mathtt{P}} \qquad \text{(OP)} \quad \frac{\Gamma \vdash \mathtt{P} \quad X \subseteq \Gamma}{\Gamma \vdash \varepsilon[X].\mathtt{P}}$$

$$\text{(IN)} \quad \frac{\Gamma, x \vdash \mathtt{P}}{\Gamma \backslash \{x\} \vdash c?x.\mathtt{P}} \qquad \text{(OUT)} \quad \frac{\Gamma \backslash \{x\} \vdash \mathtt{P}}{\Gamma, x \vdash c!x.\mathtt{P}}$$

$$\text{(RES)} \quad \frac{\Gamma \vdash \mathtt{P}}{\Gamma \vdash \mathtt{P} \backslash \mathtt{L}} \qquad \text{(SUM)} \quad \frac{\Gamma_1 \vdash \mathtt{P} \qquad \Gamma_2 \vdash \mathtt{Q}}{\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} + \mathtt{Q}}$$

$$\text{(COMM)} \quad \frac{\Gamma_1 \vdash \mathtt{P} \quad \Gamma_2 \vdash \mathtt{Q} \quad \Gamma_1 \cap \Gamma_2 = \emptyset}{\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} \mid\mid \mathtt{Q}}$$

Table 4: Typing rules for quantum processes

An important result about our typing system is that the weakening rule is admissible in it. Formally,

Theorem 4.4.1 (Weakening)**.** Assume that $\Gamma \vdash \mathtt{P}$. Then it also holds that $\Gamma, x \vdash \mathtt{P}$ for any variable x.

Let us prove that our typing system implicitly contains the weakening rule.

Proof. The proof follows by induction on the structure of quantum processes.

1. Starting with the process nil. We assume that $\Gamma \vdash \mathtt{nil}$. Then, it follows directly by an application of the rule (NIL) in Table 4 that $\Gamma, x \vdash \mathtt{nil}$.

2. Next, we consider the constant processes. Assume that $\Gamma \vdash \mathtt{A}(\Delta)$. By the rule (CONS) in Table 4 we have that $\Delta \subseteq \Gamma$. Moreover, $\Delta \subseteq \Gamma, x$. Therefore, through the application of the rule (CONS) in Table 4, $\Delta \subseteq \Gamma, x$ is true and then we obtain $\Gamma, x \vdash \mathtt{A}(\Delta)$.

3. We now consider the invisible actions. We assume that $\Gamma \vdash \tau.\mathtt{P}$, which by the application of the rule (INV) in Table 4 implies $\Gamma \vdash \mathtt{P}$. By induction we derive $\Gamma, x \vdash \mathtt{P}$. Furthermore, by an application of the rule (INV) in Table 4 we obtain $\Gamma, x \vdash \tau.\mathtt{P}$.

4. Let us now consider the super-operators actions. We assume that $\Gamma \vdash \varepsilon[X].\mathtt{P}$, which by the rule (OP) in Table 4 entails that $X \subseteq \Gamma$ and $\Gamma \vdash \mathtt{P}$. By induction we derive $\Gamma, x \vdash \mathtt{P}$, and as a consequence $X \subseteq \Gamma, x$. Moreover, by applying the rule (OP) in Table 4 we obtain $\Gamma, x \vdash \varepsilon[X].\mathtt{P}$.

5. Next, consider the case of the input actions. We assume that $\Gamma\backslash\{x\} \vdash c?x.\mathtt{P}$, which by the rule (IN) in Table 4 implies $\Gamma, x \vdash \mathtt{P}$. By induction we derive $\Gamma, x, y \vdash \mathtt{P}$. Furthermore, by applying the rule (IN) in Table 4 we obtain $\Gamma\backslash\{x\}, y \vdash c?x.\mathtt{P}$.

6. We now consider the case of the output actions. We assume that $\Gamma, x \vdash \mathtt{P}$. Furthermore, we want to prove that for **every** variable y, the condition $\Gamma, x, y \vdash c!x.\mathtt{P}$ holds. If $x = y$ we are done, because $\Gamma, x, y = \Gamma, x, x = \Gamma, x$. Otherwise, if $x \neq y$, we that $\Gamma, x \vdash \mathtt{P}$, which by the rule (OUT) in Table 4 entails $\Gamma\backslash \vdash \mathtt{P}$. By induction we derive $\Gamma\backslash\{x\}\, y \vdash \mathtt{P}$. Moreover, by applying the rule (OUT) we obtain $\Gamma, x, y \vdash c!x.\mathtt{P}$.

7. Let us now consider the restriction operator. We assume that $\Gamma \vdash \mathtt{P}\backslash\mathtt{L}$, which by the rule (RES) in Table 4 implies $\Gamma \vdash \mathtt{P}$. By induction we derive $\Gamma, x \vdash \mathtt{P}$. Moreover, by applying the rule (RES) in Table 4 we obtain $\Gamma, x \vdash \mathtt{P}\backslash\mathtt{L}$.

8. Next, we consider the sum operator. Assume that $\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} + \mathtt{Q}$, so there is $\Delta_1 \vdash \mathtt{P}$ and $\Delta_2 \vdash \mathtt{Q}$ such that $\Delta_1 \cup \Delta_2 \subseteq \Gamma_1 \cup \Gamma_2$. We have $\Delta_1 \cup \Delta_2 \vdash \mathtt{P} + \mathtt{Q}$, which implies $\Delta_1 \vdash \mathtt{P}$ and $\Delta_2 \vdash \mathtt{Q}$. By induction we derive $\Delta_1, x \vdash \mathtt{P}$ and $\Delta_2 \vdash \mathtt{Q}$ or $\Delta_1 \vdash \mathtt{P}$ and $\Delta_2, x \vdash \mathtt{Q}$. By applying the rule (SUM) in Table 4, in both cases, we obtain $\Delta_1 \cup \Delta_2, x \vdash \mathtt{P} + \mathtt{Q}$. Moreover, once $\Delta_1 \cup \Delta_2 \subseteq \Gamma_1 \cup \Gamma_2$ we have $\Gamma_1 \cup \Gamma_2, x \vdash \mathtt{P} + \mathtt{Q}$.

9. At last, we consider the case of the parallel operator. Assume that $\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} \,||\, \mathtt{Q}$, so there is $\Delta_1 \vdash \mathtt{P}$ and $\Delta_2 \vdash \mathtt{Q}$ such that $\Delta_1 \cup \Delta_2 \subseteq \Gamma_1 \cup \Gamma_2$. We have $\Delta_1 \cup \Delta_2 \vdash \mathtt{P} \,||\, \mathtt{Q}$, which entails $\Delta_1 \vdash \mathtt{P}$, $\Delta_2 \vdash \mathtt{Q}$ and $\Delta_1 \cap \Delta_2 = \emptyset$. By induction we derive $\Delta_1, x \vdash \mathtt{P}$ and $\Delta_2 \vdash \mathtt{Q}$ or $\Delta_1 \vdash \mathtt{P}$ and $\Delta_2, x \vdash \mathtt{Q}$. In both cases of induction the condition $\Delta_1 \cap \Delta_2 = \emptyset$ holds, since when $x \in \Delta_1$, $x \notin \Delta_2$ and when $x \in \Delta_2$, $x \notin \Delta_1$. By applying the rule (COMM) in Table 4, for both cases, we obtain $\Delta_1, x \cup \Delta_2 \vdash \mathtt{P} \,||\, \mathtt{Q}$ or $\Delta_1 \cup \Delta_2, x \vdash \mathtt{P} \,||\, \mathtt{Q}$, respectively. Moreover, once $\Delta_1 \cup \Delta_2 \subseteq \Gamma_1 \cup \Gamma_2$ we have $\Gamma_1, x \cup \Gamma_2 \vdash \mathtt{P} \,||\, \mathtt{Q}$ or $\Gamma_1 \cup \Gamma_2, x \vdash \mathtt{P} \,||\, \mathtt{Q}$.

□

By extending *qCCS* with a typing system, we hope to have a relation between *qCCS* processes and the formulas of the typing system. For that, we present the next theorem and prove it.

Theorem 4.4.2. $\mathtt{P}$ is a process in *qCCS* iff $\Gamma \vdash \mathtt{P}$ for some Γ.

Proof. We will begin by showing the left-to-right implication. To do that, we will need to strengthen our induction invariant to:

$$\mathtt{P} \text{ is a process in } qCCS \text{ implies that } \Gamma \vdash \mathtt{P} \text{ with } \Gamma \subseteq fv(\mathtt{P}).$$

1. We start with $\mathtt{P} = \mathtt{nil}$. We assume that $\mathtt{nil}$ is a process in *qCCS*. Applying the rule (NIL) in Table 4, we derive $\Gamma \vdash \mathtt{nil}$ for $\Gamma = \emptyset$. Since $\Gamma = \emptyset$, $\Gamma \subseteq fv(\mathtt{nil})$.

2. Next, we consider $\mathtt{P} = \mathtt{A}(\Delta)$. Assume that $\mathtt{A}(\Delta)$ is a process in *qCCS*. Applying the rule (CONS) in Table 4, we derive $\Delta \vdash \mathtt{A}(\Delta)$ with the condition $\Delta \subseteq \Delta$. Moreover, $\Delta \subseteq fv(\mathtt{A}(\Delta))$, since $fv(\mathtt{A}(\Delta)) = \Delta$.

3. Let us now consider the process $\tau.\mathtt{P}$. We assume that $\tau.\mathtt{P}$ is a process in *qCCS*. By the induction hypothesis, $\Gamma \vdash \mathtt{P}$ with $\Gamma \subseteq fv(\mathtt{P})$. Applying the rule (INV) in Table 4, we obtain $\Gamma \vdash \tau.\mathtt{P}$. Furthermore, $\Gamma \subseteq fv(\mathtt{P}) \subseteq fv(\tau.\mathtt{P})$.

4. Consider the process $\varepsilon[X].\mathtt{P}$. We assume that $\varepsilon[X].\mathtt{P}$ is a *qCCS* process. By the induction hypothesis, $\Gamma \vdash \mathtt{P}$ with $\Gamma \subseteq fv(\mathtt{P})$. Applying the rule (OP) in Table 4, we obtain $\Gamma \cup X \vdash \varepsilon[X].\mathtt{P}$ with the condition $X \subseteq \Gamma \cup X$. Since $fv(\varepsilon[X].\mathtt{P}) = fv(\mathtt{P}) \cup X$ we have that $\Gamma \cup X \subseteq fv(P) \cup X \subseteq fv(\varepsilon[X].\mathtt{P})$.

5. Consider the process $c?x.\mathtt{P}$. We assume that $c?x.\mathtt{P}$ is a *qCCS* process. By the induction hypothesis, $\Gamma \vdash \mathtt{P}$ with $\Gamma \subseteq fv(\mathtt{P})$. Since our typing system admits the weakening rule, we can weaken $\Gamma \vdash \mathtt{P}$ to $\Gamma, x \vdash \mathtt{P}$. Applying the rule (IN) in Table 4, we obtain $\Gamma \backslash \{x\} \vdash c?x.\mathtt{P}$. Moreover, $\Gamma \backslash \{x\} \subseteq fv(\mathtt{P}) \backslash \{x\} \subseteq fv(c?x.\mathtt{P})$.

6. We now consider the process $c!x.\mathtt{P}$. We assume that $c!x.\mathtt{P}$ is a *qCCS* process if $x \notin fv(\mathtt{P})$. By the induction hypothesis, $\Gamma \vdash \mathtt{P}$ with $\Gamma \subseteq fv(\mathtt{P})$, where $x \notin fv(\mathtt{P})$ and consequently $x \notin \Gamma$. Applying the rule (OUT) in Table 4, we obtain $\Gamma, x \vdash c!x.\mathtt{P}$. Moreover, $\Gamma, x \subseteq fv(\mathtt{P}) \cup \{x\} \subseteq fv(c!x.\mathtt{P})$.

7. Consider the process $\mathtt{P} \backslash \mathtt{L}$. We assume that $\mathtt{P} \backslash \mathtt{L}$ is a *qCCS* process. By the induction hypothesis, $\Gamma \vdash \mathtt{P}$ with $\Gamma \subseteq fv(\mathtt{P})$. Applying the rule (RES) in Table 4, we obtain $\Gamma \vdash \mathtt{P} \backslash \mathtt{L}$. Moreover $\Gamma \subseteq fv(\mathtt{P}) \subseteq fv(\mathtt{P} \backslash \mathtt{L})$.

8. Consider the process $\mathtt{P} + \mathtt{Q}$. We assume that $\mathtt{P} + \mathtt{Q}$ is a *qCCS* process. By the induction hypothesis, $\Gamma_1 \vdash \mathtt{P}$ with $\Gamma_1 \subseteq fv(\mathtt{P})$ and $\Gamma_2 \vdash \mathtt{Q}$ with $\Gamma_2 \subseteq fv(Q)$. Applying the rule (SUM) in Table 4 to the previous formulas, we obtain $\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} + \mathtt{Q}$. Moreover $\Gamma_1 \cup \Gamma_2 \subseteq fv(\mathtt{P}) \cup fv(\mathtt{Q}) \subseteq fv(\mathtt{P} + \mathtt{Q})$.

9. At last, let us consider the process $\mathtt{P} \mid\mid \mathtt{Q}$. We assume that $\mathtt{P} \mid\mid \mathtt{Q}$ is a *qCCS* process if $fv(\mathtt{P}) \cap fv(\mathtt{Q}) = \emptyset$. By induction we derive $\Gamma_1 \vdash \mathtt{P}$ with $\Gamma_1 \subseteq fv(\mathtt{P})$ and $\Gamma_2 \vdash \mathtt{Q}$ with $\Gamma_2 \subseteq fv(\mathtt{Q})$. Furthermore, since $fv(\mathtt{P}) \cap fv(\mathtt{Q}) = \emptyset$, $\Gamma_1 \subseteq fv(\mathtt{P})$ and $\Gamma_2 \subseteq fv(\mathtt{Q})$, we have that $\Gamma_1 \cap \Gamma_2 = \emptyset$. Applying the rule (COMM) in Table 4 to the previous formulas, we obtain $\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} \mid\mid \mathtt{Q}$. Moreover $\Gamma_1 \cup \Gamma_2 \subseteq fv(\mathtt{P}) \cup fv(\mathtt{Q}) \subseteq fv(\mathtt{P} \mid\mid \mathtt{Q})$.

Now we will show the right-to-left implication. To do that, we need another induction invariant.

$$\Gamma \vdash \mathtt{P} \text{ implies a } qCCS \text{ process } \mathtt{P} \text{ with } fv(\mathtt{P}) \subseteq \Gamma.$$

1. We start with the nil process. We assume that $\Gamma \vdash \mathtt{nil}$. By item 1 in Definition 4.3.1, $\mathtt{nil}$ is a *qCCS* process. Moreover $fv(\mathtt{nil}) = \emptyset \subseteq \Gamma$.

2. Next, we consider the constant processes. We assume that $\Gamma \vdash \mathtt{A}(\Delta)$ with $\Delta \subseteq \Gamma$. By item 2 in Definition 4.3.1, $\mathtt{A}(\Delta)$ is a *qCCS* process. Moreover, $fv(\mathtt{A}(\Delta)) = \Delta \subseteq \Gamma$.

3. Consider the invisible prefix. We assume that $\Gamma \vdash \tau.\mathtt{P}$. The rule (INV) in Table 4 implies $\Gamma \vdash \mathtt{P}$. By induction, we derive that $\mathtt{P}$ is a *qCCS* process with $fv(\mathtt{P}) \subseteq \Gamma$. By item 3 in Definition 4.3.1, we have that $\tau.\mathtt{P}$ is a process in *qCCS*. Moreover $fv(\tau.\mathtt{P}) = fv(\mathtt{P}) \subseteq \Gamma$.

4. Consider now the super-operator prefix. We assume that $\Gamma \vdash \varepsilon[X].\mathtt{P}$ with $X \subseteq \Gamma$. The rule (OP) in Table 4 implies $\Gamma \vdash \mathtt{P}$. By induction, we derive that $\mathtt{P}$ is a process in *qCCS* with $fv(\mathtt{P}) \subseteq \Gamma$. By item 4 in Definition 4.3.1 we have that $\varepsilon[X].\mathtt{P}$ is a *qCCS* process. Moreover, $fv(\varepsilon[X].\mathtt{P}) = fv(\mathtt{P}) \cup X \subseteq \Gamma \cup X \subseteq \Gamma$, since $X \subseteq \Gamma$.

5. Next, we consider the input prefix. We assume that $\Gamma \vdash c?x.\mathtt{P}$. By the rule (IN) in Table 4 we know that $x \notin \Gamma$. Furthermore, the same rule implies $\Gamma, x \vdash \mathtt{P}$. By induction $\mathtt{P}$ is a *qCCS* process with $fv(\mathtt{P}) \subseteq \Gamma \cup \{x\}$. By item 5 in Definition 4.3.1, $c?x.\mathtt{P}$ is a *qCCS* process. Moreover, $fv(c?x.\mathtt{P}) = fv(\mathtt{P}) \backslash \{x\} \subseteq (\Gamma \cup \{x\}) \backslash \{x\} \subseteq \Gamma$.

6. Consider the output prefix. We assume that $\Gamma \vdash c!x.\mathtt{P}$. By the rule (OUT) in Table 4 we know that $x \in \Gamma$. Furthermore, the same rule implies $\Gamma \backslash \{x\} \vdash \mathtt{P}$. By induction $\mathtt{P}$ is a *qCCS* process with $fv(\mathtt{P}) \subseteq \Gamma$ and $x \notin fv(\mathtt{P})$ because $\Gamma \backslash \{x\} \vdash \mathtt{P}$. By item 6 in Definition 4.3.1, $c!x.\mathtt{P}$ is a *qCCS* process if $x \notin fv(\mathtt{P})$. Moreover, $fv(c!x.\mathtt{P}) = fv(P) \cup \{x\} \subseteq \Gamma, x \subseteq \Gamma$, since $x \in \Gamma$.

7. Consider the restriction operator. We assume that $\Gamma \vdash \mathtt{P} \backslash \mathtt{L}$. The rule (RES) in Table 4 implies $\Gamma \vdash \mathtt{P}$. By induction $\mathtt{P}$ is a *qCCS* process with $fv(\mathtt{P}) \subseteq \Gamma$. By the item 9 in Definition 4.3.1, $\mathtt{P} \backslash \mathtt{L}$ is a *qCCS* process. Moreover $fv(\mathtt{P} \backslash \mathtt{L}) = fv(\mathtt{P}) \subseteq \Gamma$.

8. Consider the sum operator. We assume that $\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} + \mathtt{Q}$, so there is $\Delta_1 \vdash \mathtt{P}$ and $\Delta_2 \vdash \mathtt{Q}$ such that $\Delta_1 \cup \Delta_2 \subseteq \Gamma_1 \cup \Gamma_2$. The rule (SUM) in Table 4 implies $\Delta_1 \vdash \mathtt{P}$ and $\Delta_2 \vdash \mathtt{Q}$. By induction $\mathtt{P}$ and $\mathtt{Q}$ are *qCCS* processes with $fv(\mathtt{P}) \subseteq \Delta_1$ and $fv(\mathtt{Q}) \subseteq \Delta_2$. By item 7 in Definition 4.3.1 $\mathtt{P} + \mathtt{Q}$ is a *qCCS* process. Moreover $fv(\mathtt{P} + \mathtt{Q}) = fv(\mathtt{P}) \cup fv(\mathtt{Q}) \subseteq \Delta_1 \cup \Delta_2 \subseteq \Gamma_1 \cup \Gamma_2$.

9. At last, consider the parallel operator. We assume that $\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} \,||\, \mathtt{Q}$, so there is $\Delta_1 \vdash \mathtt{P}$ and $\Delta_2 \vdash \mathtt{Q}$ such that $\Delta_1 \cup \Delta_2 \subseteq \Gamma_1 \cup \Gamma_2$. The rule (COMM) in Table 4 entails $\Delta_1 \vdash \mathtt{P}$, $\Delta_2 \vdash \mathtt{Q}$ and $\Delta_1 \cap \Delta_2 = \emptyset$. By induction $\mathtt{P}$ and $\mathtt{Q}$ are *qCCS* processes with $fv(\mathtt{P}) \subseteq \Delta_1$ and $fv(\mathtt{Q}) \subseteq \Delta_2$. Furthermore, $fv(\mathtt{P}) \cap fv(\mathtt{Q}) = \emptyset$ because $\Delta_1 \cap \Delta_2 = \emptyset$. By item 8 in Definition 4.3.1 $\mathtt{P} \,||\, \mathtt{Q}$ is a *qCCS* process if $fv(\mathtt{P}) \cap fv(\mathtt{Q}) = \emptyset$. Moreover, $fv(\mathtt{P} \,||\, \mathtt{Q}) = fv(\mathtt{P}) \cup fv(\mathtt{Q}) \subseteq \Delta_1 \cup \Delta_2 \subseteq \Gamma_1 \cup \Gamma_2$.

□

4.5 COMPARISON BETWEEN CCS AND QCCS

In this section we make a comparison between the two process algebras, *CCS* and *qCCS*. The main differences between *CCS* and *qCCS* are related to the need to handle quantum data, more concretely to deal with the no-cloning theorem, the collapse of states when measured and the application of super-operators on qubits.

In the introduction of *CCS* we gave an informal view where the processes were represented by black boxes with ports illustrating the actions they could perform. In *qCCS*, this informal view was not given due to the difficulty of representing the states $\rho \in D(\mathcal{H})$ that are associated with processes $\mathtt{P} \in \mathcal{P}$ and the actions of the super-operators on the qubits. The Table 5 summarises the main differences between *CCS* and *qCCS* with respect to syntax.

	CCS	*qCCS*
Constant Process	$\mathtt{K}$	$\mathtt{A}(\tilde{x})$
Nil (Null Process)	$\mathtt{nil}$	$\mathtt{nil}$
Action Prefixing	$\alpha.\mathtt{P}$	$\tau.\mathtt{P}$ $\varepsilon[X].\mathtt{P}$ $c!x.\mathtt{P}$ $c?x.\mathtt{P}$
Choice Operator	$\sum_{i\in I} \mathtt{P}_i$	$\mathtt{P} + \mathtt{Q}$
Parallel Composition	$\mathtt{P} \mid \mathtt{Q}$	$\mathtt{P} \,||\, \mathtt{Q}$
Restriction	$\mathtt{P}\backslash\mathtt{L}$	$\mathtt{P}\backslash\mathtt{L}$
Relabelling	$\mathtt{P}[f]$	$\mathtt{P}f$

Table 5: *CCS* vs *qCCS* syntax

Before analysing Table 5 it is important to make a remark on the actions that allow communications between processes. In *CCS* those actions are denoted by a label, without specifying a channel name, and communications between processes are performed through complementary actions (e.g: $\mathtt{a}$ and $\overline{\mathtt{a}}$) . On the other hand, in *qCCS* the input and output actions involve a channel name, a symbol ('!' for output and '?' for input) and a value to be transmitted. With this the communication between processes happens when the names of the channels are equal and the symbols are different (e.g: $c!a$ and $c?a$). Constant processes in *CCS* have a similar definition to the one in *qCCS*, as the reader can see in Milner (1989). The choice operator in *CCS* is defined by a sum indexed by a set of values while in *qCCS* the choice operator is binary. The major distinction is located at action prefixing. While *CCS* uses α to represent all the actions that can occur, in *qCCS* we need to split it because of the existence of invisible actions, communication actions and super-operators that acts on qubits. While in *CCS* the operational semantics only refers to processes, in *qCCS* the operational semantics is composed by a pair $\langle \mathtt{P}, \rho \rangle$, where $\mathtt{P} \in \mathcal{P}$ represents quantum processes and ρ specifies the current state of the environment. This difference is made clearer by Table 6, which we analyse next. Moreover, in the first column of the Table 6, if there is only one name it means that the name of the rule is equal in *CCS* and *qCCS*.

CCS|*qCCS* rule names	*CCS*	*qCCS*
Act	$\dfrac{}{\alpha.\mathtt{P} \xrightarrow{\alpha} \mathtt{P}}$	$\dfrac{}{\langle \tau.\mathtt{P}, \rho \rangle \xrightarrow{\tau} \langle \mathtt{P}, \rho \rangle}$
Choice	$\dfrac{\mathtt{P}_j \xrightarrow{\alpha} \mathtt{P'}_j}{\sum_{i \in I} \mathtt{P}_i \xrightarrow{\alpha} \mathtt{P'}_j}\ j \in I$	$\dfrac{\langle \mathtt{P}, \rho \rangle \xrightarrow{\alpha} \langle \mathtt{P'}, \rho' \rangle}{\langle \mathtt{P} + \mathtt{Q}, \rho \rangle \xrightarrow{\alpha} \langle \mathtt{P'}, \rho' \rangle}$
Com | (Intl2 and Intl1)	$\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P'}}{\mathtt{P} \mid \mathtt{Q} \xrightarrow{\alpha} \mathtt{P'} \mid \mathtt{Q}}$	$\dfrac{\langle \mathtt{P}, \rho \rangle \xrightarrow{\alpha} \langle \mathtt{P'}, \rho' \rangle}{\langle \mathtt{P} \mid\mid \mathtt{Q}, \rho \rangle \xrightarrow{\alpha} \langle \mathtt{P'} \mid\mid \mathtt{Q}, \rho' \rangle}\alpha$ is not an input $\dfrac{\langle \mathtt{P}, \rho \rangle \xrightarrow{c?x} \langle \mathtt{P'}, \rho' \rangle}{\langle \mathtt{P} \mid\mid \mathtt{Q}, \rho \rangle \xrightarrow{c?x} \langle \mathtt{P'} \mid\mid \mathtt{Q}, \rho' \rangle}\ x \notin fv(\mathtt{Q})$
Com3 | Comm	$\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P'} \wedge \mathtt{Q} \xrightarrow{\overline{\alpha}} \mathtt{Q'}}{\mathtt{P} \mid \mathtt{Q} \xrightarrow{\tau} \mathtt{P'} \mid \mathtt{Q'}}$	$\dfrac{\langle \mathtt{P}, \rho \rangle \xrightarrow{c?x} \langle \mathtt{P'}, \rho \rangle\ \langle \mathtt{Q}, \rho \rangle \xrightarrow{c!x} \langle \mathtt{Q'}, \rho \rangle}{\langle \mathtt{P} \mid\mid \mathtt{Q}, \rho \rangle \xrightarrow{\tau} \langle \mathtt{P'} \mid\mid \mathtt{Q'}, \rho \rangle}$
Res	$\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P'}}{\mathtt{P}\backslash \mathtt{L} \xrightarrow{\alpha} \mathtt{P'}\backslash \mathtt{L}}\ \alpha, \overline{\alpha} \notin L$	$\dfrac{\langle \mathtt{P}, \rho \rangle \xrightarrow{\alpha} \langle \mathtt{P'}, \rho' \rangle}{\langle \mathtt{P}\backslash \mathtt{L}, \rho \rangle \xrightarrow{\alpha} \langle \mathtt{P'}\backslash \mathtt{L}, \rho' \rangle}\ cn(\alpha) \notin \mathtt{L}$
Rel	$\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P'}}{\mathtt{P}[\mathtt{f}] \xrightarrow{f(\alpha)} \mathtt{P'}[\mathtt{f}]}$	
Con | Def	$\dfrac{\mathtt{P} \xrightarrow{\alpha} \mathtt{P'}}{\mathtt{K} \xrightarrow{\alpha} \mathtt{P'}}\ \mathtt{K} \overset{\mathtt{def}}{=} \mathtt{P}$	$\dfrac{\langle \mathtt{P}\{\tilde{y}/\tilde{x}\}, \rho \rangle \xrightarrow{\alpha} \langle \mathtt{P'}, \rho' \rangle}{\langle \mathtt{A}(\tilde{y}), \rho \rangle \xrightarrow{\alpha} \langle \mathtt{P'}, \rho' \rangle}\ \mathtt{A}(\tilde{x}) \overset{def}{=} \mathtt{P}$

Oper		$\dfrac{}{\langle \varepsilon[X].\mathtt{P}, \rho\rangle \xrightarrow{\varepsilon[X]} \langle \mathtt{P}, \varepsilon_X(\rho)\rangle}$
Input		$\dfrac{y \notin fv(c?x.\mathtt{P})}{\langle c?x.\mathtt{P}, \rho\rangle \xrightarrow{c?y} \langle \mathtt{P}\{y/x\}, \rho\rangle}$
Output		$\dfrac{}{\langle c!x.\mathtt{P}, \rho\rangle \xrightarrow{c!x} \langle \mathtt{P}, \rho\rangle}$

Table 6: *CCS* vs *qCCS* operational semantics

The symmetric versions of Com, Com3, Intl2, Comm and Intl1 are not represented in the table. Note that the transitions in *CCS* and *qCCS* are similar, the differences being originated by the quantum nature of the latter as we can see in Com | Intl1 | Intl2, where it is needed to divide the corresponding rule in two and add side conditions. Another interesting distinction is that in *qCCS* the rule Rel is not included, as substitution in quantum computation is not as trivial as in the classical case and, therefore, can not be generalised as in *CCS*. At last, the rules Oper, Input and Output are only presented in *qCCS* due to reasons that were previously exposed. An important feature of both process algebras is the notion of behavioural equivalence. *CCS* developed strong bisimularity and weak bisimularity to see if two or more processes have an equivalent behaviour. In *qCCS* the notion of behavioural equivalence is used on configurations and for that reason a new notion of strong bisimularity was developed for these configurations. Unlike the comparison done at the levels of syntax and operational semantics, for behavioural equivalence it is not feasible to establish a direct relation between the methods that Milner (1989) and Ying et al. (2009) developed. This is because, in the latter we have the notion of configuration that involves a process and a density operator associated to the process, while in the former we only have a process. Even knowing that is possible to establish a behavioural equivalence between two processes in *qCCS*, we need to assure that the processes have the same density operator associated to them.

5

TIMED QCCS

5.1 MOTIVATION AND CONTEXT

In the current days, the quantum process algebras developed - *qCCS*, *CQP*, etc... - assume the existence of a perfect quantum computer, in which qubits are not affected by noise. However, the quantum computers that exist are all faulty, *i.e* the qubits are not immune to noise. Therefore, we need a quantum process algebra that is capable to reason about current quantum systems. For that, the introduction of temporal notions into the quantum process algebra is essential, once, among other advantages, it allows to control the coherence time of qubits.

As in the classical situation, the introduction of time into a quantum process algebra would also increase the accuracy of the descriptions of communication protocols, such like the BB84 communication protocol.

In 1978, Tony Hoare developed *CSP*, Hoare (1978), which was the first of the three important process algebras. Ten years later, in 1988, G.M. Reed and A.W. Roscoe introduced temporal notions in *CSP*, Roscoe and Reed (1988). With the introduction of a quantitative notion of time, it was possible to increase the accuracy of the descriptions of communication protocols.

In 2007, Yuan Feng *et al* developed a first version of *qCCS* that considered classical and quantum variables. A year later, in 2008, Mingsheng Ying *et al* presented a version of *qCCS* that only takes into consideration quantum variables. At last, in 2012, Yuan Feng *et al* presented another version of *qCCS* that considers, again, classical and quantum variables. Ten years passed since the first version of *qCCS* and we do not know any work related to the introduction of temporal notions on quantum process algebras.

We now make a walk-through of this chapter.

In order to enable classical communication between quantum and classical computers (which is needed for the QRAM architecture), we find convenient to consider classical variables in quantum process algebras. Having said this, we present a version of *qCCS* that supports both classical and quantum variables, Feng et al. (2012). We then update our previously introduced typing system with classical variables as well. After this, we prove

that the extended typing system still has the weakening property and that the updated version of the Theorem 4.4.2 still holds, *i.e.* there is an equivalence between *qCCS* processes and the typing system developed by us.

After that, we put the new version of *qCCS* under stress tests: we developed some examples to see if the new *qCCS* could encode them successfully. Unfortunately, that was not verified and we developed a new action. To see if the new action could solve the problem, we encoded again the examples. The result obtained was satisfactory. With this, we got a quantum process algebra that fulfil our objectives.

Later on, we introduced a temporal dimension in the quantum process algebra developed. We present the syntax, the operational semantics and, the typing system developed. We then rephrase and test the examples.

5.2 Quantum CCS with measurement operations and classical control

As mentioned earlier, we will use an extended version of *qCCS*, namely Feng et al. (2012). This version extends *qCCS* with classical variables and measurement operations. This allows sending/receiving classical information, and, in particular, storing results of measurements in classical variables.

Next, we present the syntax and semantics of the operations that extend *qCCS* with classical variables and measurement, Feng et al. (2012).

1. $c?x.\mathrm{P} \in \mathcal{P}$, and $qv(c?x.\mathrm{P})=qv(\mathrm{P})$
2. $c!x.\mathrm{P} \in \mathcal{P}$, and $qv(c!x.\mathrm{P})=qv(\mathrm{P})$
3. $M[\tilde{x};r].\mathrm{P} \in \mathcal{P}$, and $qv(M[\tilde{x};r].\mathrm{P})=qv(\mathrm{P})\cup\tilde{x}$
4. **if** b **then** $\mathrm{P} \in \mathcal{P}$, and qv(**if** b **then** $\mathrm{P})=qv(\mathrm{P})$

In the above formation rules c stands for a classical communication channel. The third formation rule refers to the measurement of a set of qubits, $\tilde{x}$, whose result is stored in a classical variable, r. *If clauses* are now relevant, because we now have at our disposal classical variables which we can use for tests.

In the rules above, the set of free quantum variables of a process is denoted as qv (previously it was fv).

Before presenting the operational semantics, we need to make a remark about the *spectral decomposition* (see Nielsen and Chuang (2000) for details). The spectral decomposition states that if an operator is diagonalizable then it is a normal operator ($AA^\dagger = A^\dagger A$). The following theorem is proved in Nielsen and Chuang (2000).

Theorem 5.2.1. Any normal operator M on a vector space V is diagonal with respect to some orthonormal basis for V. Conversely, any diagonalizable operator is normal.

Now, we can present the operational semantics for the previous elements and also the communication between classical channels.

$$\textbf{C-Inp:}\ \frac{}{\langle\ c?x.\mathtt{P},\ \rho\ \rangle \xrightarrow{c?r} \langle\ \mathtt{P}\{r/x\},\ \rho\ \rangle}\ r \in \text{Real}$$

$$\textbf{C-Outp:}\ \frac{}{\langle\ c!e.\mathtt{P},\ \rho\ \rangle \xrightarrow{c!r} \langle\ \mathtt{P},\ \rho\ \rangle}\ r = [\![e]\!]$$

$$\textbf{C-Com:}\ \frac{\langle\ \mathtt{P}_1,\ \rho\ \rangle \xrightarrow{c?r} \langle\ \mathtt{P}_1',\ \rho\ \rangle \quad \langle\ \mathtt{P}_2,\ \rho\ \rangle \xrightarrow{c!r} \langle\ \mathtt{P}_2',\ \rho\ \rangle}{\langle\ \mathtt{P}_1 \mid\mid \mathtt{P}_2,\ \rho\ \rangle \xrightarrow{\tau} \langle\ \mathtt{P}_1' \mid\mid \mathtt{P}_2',\ \rho\ \rangle}$$

$$\textbf{Cho:}\ \frac{\langle\ \mathtt{P},\ \rho\ \rangle \xrightarrow{\alpha} \mu}{\langle\ \textbf{if}\ b\ \textbf{then}\ \mathtt{P},\ \rho\ \rangle \xrightarrow{\alpha} \mu}\ [\![b]\!] = true$$

$$\textbf{Meas:}\ \frac{}{\langle\ \mathtt{M}[\tilde{x};r].\mathtt{P},\ \rho\rangle \xrightarrow{\tau} \sum_{i\in I} p_i \langle\ \mathtt{P}\{\lambda_i/x\}, E^i_{\tilde{x}}\rho E^i_{\tilde{x}}/p_i\ \rangle}$$
where $\mathtt{M}$ has the spectral decomposition $\mathtt{M} = \sum_{i\in I} \lambda_i E^i$ and $p_i = tr(E^i_{\tilde{x}}\rho)$

The transition system is now a probabilistic transition system, because of the measurement rule. This is because when one performs a measurement, the output of a certain state is associated with a probability. Note that, after a transition, the right-hand-side becomes a probabilistic distribution over configurations. Relatively to the operational semantics, there is some changes from Ying et al. (2009) to Feng et al. (2012). While in the first one, transitions through operators are visible, in the second, they are invisible, *i.e.* they are represented as τ-transitions.

In Section 4.4, we introduced a typing system for *qCCS* that only takes into consideration quantum variables. Here, we introduce a typing system that considers both quantum and classical variables, as in the extended version of *qCCS*, Feng et al. (2012). Thus, environments Γ will now consist of typed variables, specifically they will be non-repetitive lists of quantum variables $x : Q$ and classical variables $c : C$. Moreover, the rules presented in Section 4.4 can be easily extended to this version of *qCCS*, where quantum and classical variables are taken in consideration.

Next, we present the typing rules for the above process operations:

The rules (C-IN) and (C-OUT) in Table 7 represent the input and output of classical variables, respectively. Moreover, since these two rules are classical, the no-cloning theorem is not a restriction. Therefore, in the rule (C-OUT), we can make a copy of the classical variable x and send it. In the rule (C-IN) we could also do the same, but we prefer to expand

$$\text{(C-IN)}\quad \frac{\Gamma, x : C \vdash \mathtt{P}}{\Gamma \vdash c?x.\mathtt{P}} \qquad \text{(C-OUT)}\quad \frac{\Gamma, x : C \vdash \mathtt{P}}{\Gamma, x : C \vdash c!x.\mathtt{P}}$$

$$\text{(IF)}\quad \frac{\Gamma \vdash \mathtt{P} \quad \Delta \vdash b}{\Gamma, \Delta \vdash \textbf{if } b \textbf{ then } \mathtt{P}} \qquad \text{(MEAS)}\quad \frac{\Gamma, r : C \vdash \mathtt{P}}{\Gamma, x : Q \vdash \mathtt{M}[x; r].\mathtt{P}}$$

Table 7: Remaining rules for the typing system

the environment with a new classical variable instead of updating a classical variable. The rule (IF) merges in the conclusion the two environments of the premises, where Γ denotes the environment of the process $\mathtt{P}$ and Δ denotes the environment of clause b. Moreover, the latter is constituted only by classical variables, while the former can have quantum and classical variables. At last, we have the (MEAS) rule. Here, after performing the measurement, the variable $x : Q$ is eliminated from the environment and the variable $r : C$ is added. This last variable stores the measurement result of $x : Q$, as it happens in *qCCS*.

In the (COMM) rule of Table 4, the condition $\Gamma_1 \cap \Gamma_2 = \emptyset$ only takes into consideration the quantum variables. To prevent a situation where two processes want to exchange a qubit, but they have a common classical variable violating the condition $\Gamma_1 \cap \Gamma_2 = \emptyset$, we decided to denote the environment of quantum variables by Γ^Q, such that $\Gamma^Q \subseteq \Gamma$. This notation is going to be useful for the proofs we are going to make. The rule (COMM) is well suited for quantum and classical communication because when a classical communication is performed, the condition $\Gamma_1^Q \cap \Gamma_2^Q = \emptyset$ always holds. That is why we did not a rule for classical communication.

Having introduced the new rules of the typing system, we need to extend the proofs of the weakening property and the relation between *qCCS* processes and the typing system in Section 4.4 into the new setting of classical variables intermixed with quantum ones. In particular, since we now have classical variables, we will also need to prove that the extended typing system admits the "classical weakening" property. We will begin by showing this. Then, we will prove the "quantum weakening" property for the four typing rules presented above. The proof for the rules in Section 4.4.2 is not going to be detailed because it is very similar to the one introduced in Section 4.4. Finally, we adapt the Theorem 4.4.2, proving it for the four typing rules above.

The classical weakening property is formally defined as follows:

Theorem 5.2.2. Assume that $\Gamma \vdash \mathtt{P}$. Then, $\Gamma, x : C \vdash \mathtt{P}$ for any classical variable x.

Proof. The proof follows by induction on structure of quantum processes.

1. Starting with the nil process. We assume that $\Gamma \vdash \mathtt{nil}$. It follows by direct application of the rule (NIL) in Table 4 that $\Gamma, r : C \vdash \mathtt{nil}$.

2. Next, we consider the constant processes. We assume that $\Gamma \vdash \mathtt{A}(\Delta)$. By the rule (CONS) in Table 4 we have $\Delta \subseteq \Gamma$. Moreover, $\Delta \subseteq \Gamma, r : C$ for any classical variable r. By the rule (CONS) in Table 4, $\Delta \subseteq \Gamma, r : C$ is true and then we obtain $\Gamma, r : C \vdash \mathtt{A}(\Delta)$.

3. We now consider the invisible prefix. We assume that $\Gamma \vdash \tau.\mathtt{P}$, which entails $\Gamma \vdash \mathtt{P}$. By induction, we derive $\Gamma, r : C \vdash \mathtt{P}$. Applying the rule (INV) in Table 4, we obtain $\Gamma, r : C \vdash \tau.\mathtt{P}$.

4. Consider the super-operator prefix. We assume that $\Gamma \vdash \varepsilon[X].\mathtt{P}$, which entails $X \subseteq \Gamma$ and $\Gamma \vdash \mathtt{P}$. By induction we derive $\Gamma, r : C \vdash \mathtt{P}$, and as a consequence $X \subseteq \Gamma, r : C$. Applying the rule (OP) in Table 4 we obtain $\Gamma, r : C \vdash \varepsilon[X].\mathtt{P}$.

5. Next, we consider the quantum input prefix. We assume that $\Gamma \vdash q?x.\mathtt{P}$, which entails $\Gamma, x : Q \vdash \mathtt{P}$. By induction, we derive $\Gamma, x : Q, r : C \vdash \mathtt{P}$. Applying the rule (IN) in Table 4, we obtain $\Gamma, r : C \vdash q?x.\mathtt{P}$.

6. We now consider the quantum output prefix. We assume $\Gamma, x : Q \vdash q!x.\mathtt{P}$, which implies $\Gamma \vdash \mathtt{P}$. By induction, we derive $\Gamma, r : C \vdash \mathtt{P}$. Applying the rule (OUT) in Table 4, we obtain $\Gamma, x : Q, r : C \vdash q!x.\mathtt{P}$.

7. We now consider the restriction operator. We assume $\Gamma \vdash \mathtt{P}\backslash\mathtt{L}$, which entails $\Gamma \vdash \mathtt{P}$. By induction, we derive $\Gamma, r : C \vdash \mathtt{P}$. Applying the rule (RES) in Table 4, we obtain $\Gamma, r : C \vdash \mathtt{P}\backslash\mathtt{L}$.

8. Next, we consider the sum operator. We assume $\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} + \mathtt{Q}$, so there is $\Lambda_1 \vdash \mathtt{P}$ and $\Lambda_2 \vdash \mathtt{Q}$ such that $\Lambda_1 \cup \Lambda_2 \subseteq \Gamma_1 \cup \Gamma_2$. We have $\Lambda_1 \cup \Lambda_2 \vdash \mathtt{P} + \mathtt{Q}$, which entails $\Lambda_1 \vdash \mathtt{P}$ and $\Lambda_2 \vdash \mathtt{Q}$. By induction, we derive $\Lambda_1, r : C \vdash \mathtt{P}$ and $\Lambda_2 \vdash \mathtt{Q}$ or $\Lambda_1 \vdash \mathtt{P}$ and $\Lambda_2, r : C \vdash \mathtt{Q}$. In both cases, applying the rule (SUM) in Table 4, we obtain $\Lambda_1 \cup \Lambda_2, r : C \vdash \mathtt{P} + \mathtt{Q}$. Moreover, once $\Lambda_1 \cup \Lambda_2 \subseteq \Gamma_1 \cup \Gamma_2$, we have $\Gamma_1 \cup \Gamma_2, r : C \vdash \mathtt{P} + \mathtt{Q}$.

9. We now consider the parallel operator. We assume that $\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} \,||\, \mathtt{Q}$, so there is $\Lambda_1 \vdash \mathtt{P}$ and $\Lambda_2 \vdash \mathtt{Q}$ such that $\Lambda_1 \cup \Lambda_2 \subseteq \Gamma_1 \cup \Gamma_2$. We have $\Lambda_1 \cup \Lambda_2 \vdash \mathtt{P} \,||\, \mathtt{Q}$ that implies $\Lambda_1 \vdash \mathtt{P}$, $\Lambda_2 \vdash \mathtt{Q}$ and $\Lambda_1^Q \cap \Lambda_2^Q = \emptyset$. By induction, we derive $\Lambda_1, r : C \vdash \mathtt{P}$ and $\Lambda_2 \vdash \mathtt{Q}$ or $\Lambda_1 \vdash \mathtt{P}$ and $\Lambda_2, r : C \vdash \mathtt{Q}$, where the condition $\Lambda_1^Q \cap \Lambda_2^Q = \emptyset$ holds, once no quantum variable was added to the environment. In both cases, by applying the rule (COMM) in Table 4, we obtain $\Lambda_1 \cup \Lambda_2, r : C \vdash \mathtt{P} \,||\, \mathtt{Q}$. Moreover, once $\Lambda_1 \cup \Lambda_2 \subseteq \Gamma_1 \cup \Gamma_2$, we have $\Gamma_1 \cup \Gamma_2, r : C \vdash \mathtt{P} + \mathtt{Q}$.

10. We now consider the classic input prefix. We assume that $\Gamma \vdash c?r.\mathtt{P}$, which entails $\Gamma, r : C \vdash \mathtt{P}$. By induction, we derive $\Gamma, r : C, s : C \vdash \mathtt{P}$. Applying the rule (C-IN) in Table 7, we obtain $\Gamma, s : C \vdash c?r.\mathtt{P}$.

11. Next, we consider the classic output prefix. We assume that $\Gamma, r : C \vdash c!r.\mathtt{P}$. We want to show that for every classical variable s, the condition $\Gamma, r : C, s : C \vdash c!r.\mathtt{P}$ holds. If $r : C = s : C$ then we are done because $\Gamma, r : C, s : C = \Gamma, r : C, r : C = \Gamma, r : C$. Otherwise, if $r : C \neq s : C$, then we have $\Gamma, r : C \vdash c!r.\mathtt{P}$, which entails $\Gamma, r : C \vdash \mathtt{P}$. By induction, we derive $\Gamma, r : C, s : C \vdash \mathtt{P}$. Applying the rule (C-OUT) in Table 7, we obtain $\Gamma, r : C, s : C \vdash c!r.\mathtt{P}$.

12. We now consider the if operator. We assume that $\Gamma, \Delta \vdash \textbf{if } b \textbf{ then } \mathtt{P}$, so there is $\Lambda_1 \vdash \mathtt{P}$ and $\Lambda_2 \vdash b$ such that $\Lambda_1, \Lambda_2 \subseteq \Gamma, \Delta$. We have $\Lambda_1, \Lambda_2 \vdash \textbf{if } b \textbf{ then } \mathtt{P}$, which entails $\Lambda_1 \vdash \mathtt{P}$ and $\Lambda_2 \vdash b$. By induction, we derive $\Lambda_1, r : C \vdash \mathtt{P}$ and $\Lambda_2 \vdash b$ or $\Lambda_1 \vdash \mathtt{P}$ and $\Lambda_2, r : C \vdash b$. In both cases, applying the rule (IF) in Table 7, we obtain $\Lambda_1, \Lambda_2, r : C \vdash \textbf{if } b \textbf{ then } \mathtt{P}$. Moreover, once $\Lambda_1, \Lambda_2 \subseteq \Gamma, \Delta$ we have $\Gamma, \Delta, r : C \vdash \textbf{if } b \textbf{ then } \mathtt{P}$.

13. At last, we consider the measurement prefix. We assume $\Gamma, x : Q \vdash \mathtt{M}[x; r].\mathtt{P}$, which entails $\Gamma, r : C \vdash \mathtt{P}$. By induction, we derive $\Gamma, r : C, s : C \vdash \mathtt{P}$. Applying the rule (MEAS) in Table 7, we obtain $\Gamma, x : Q, s : C \vdash \mathtt{M}[x, r].\mathtt{P}$.

□

Now that we proved Theorem 5.2.2, we can move forward to the admissibility of quantum weakening, which is formally defined as follows:

Theorem 5.2.3. Assume that $\Gamma \vdash \mathtt{P}$. Then, $\Gamma, x : Q \vdash \mathtt{P}$ for any quantum variable x.

Proof. We will focus only on the rules (C-IN), (C-OUT), (IF) and (MEAS). The proof follows by induction over structure of quantum processes.

1. Beginning with the measurement prefix. We assume that $\Gamma, x : Q \vdash \mathtt{M}[x; r].\mathtt{P}$. Moreover, we want to show that for every quantum variable y, the condition $\Gamma, x : Q, y : Q \vdash \mathtt{M}[x; r].\mathtt{P}$ holds. If $x : Q = y : Q$, then we are done because $\Gamma, x : Q, y : Q = \Gamma, x : Q, x : Q = \Gamma, x : Q$. Otherwise, if $x : Q \neq y : Q$, we return to the assumption $\Gamma, x : Q \vdash \mathtt{M}[x, r].\mathtt{P}$ that entails $\Gamma, r : C \vdash \mathtt{P}$. By induction, we derive $\Gamma, y : Q, r : C \vdash \mathtt{P}$. Applying the rule (MEAS) in Table 7, we obtain $\Gamma, y : Q, x : Q \vdash \mathtt{M}[x; r].\mathtt{P}$.

2. Next, we consider the classic input prefix. We assume that $\Gamma \vdash c?r.\mathtt{P}$, which entails that $\Gamma, r : C \vdash \mathtt{P}$. By induction, we derive $\Gamma,\ r : C,\ x : Q \vdash \mathtt{P}$. Applying the rule (C-IN) in Table 7, we obtain $\Gamma, x : Q \vdash c?r.\mathtt{P}$.

3. Let us now consider the classic output prefix. We assume that $\Gamma, r : C \vdash c!r.\mathtt{P}$, which entails that $\Gamma, r : C \vdash \mathtt{P}$. By induction, we derive $\Gamma, r : C, x : Q \vdash \mathtt{P}$. Applying the rule (C-OUT) in Table 7, we obtain $\Gamma, r : C, x : Q \vdash c!r.\mathtt{P}$.

4. At last, we consider the if operator. We assume that $\Gamma, \Delta \vdash \textbf{if } b \textbf{ then } \mathtt{P}$, which entails $\Gamma \vdash \mathtt{P}$ and $\Delta \vdash b$. By induction, we derive $\Gamma, x : Q \vdash \mathtt{P}$, once Δ only considers classical variables. Applying the rule (IF) in Table 7, we obtain $\Gamma, x : Q, \Delta \vdash \textbf{if } b \textbf{ then } \mathtt{P}$.

□

With this, we have proved that our typing system still admits the weakening rule. Now, we need to extend Theorem 4.4.2 to the setting of classical variables intermixed with quantum ones.

Theorem 5.2.4. $\mathtt{P}$ is a process in *qCCS* iff $\Gamma \vdash \mathtt{P}$ for some Γ.

Proof. We will begin by showing the left-to-right implication. We will need to strengthen the induction invariant to:

$$\mathtt{P} \text{ is a process in } qCCS \text{ implies } \Gamma \vdash \mathtt{P} \text{ with } \Gamma^{Q} \subseteq qv(\mathtt{P}).$$

1. We start with the measurement prefix. We assume that $\mathtt{M}[x;r].\mathtt{P}$ is a *qCCS* process. By induction, we derive that $\Gamma \vdash \mathtt{P}$ with $\Gamma^{Q} \subseteq qv(\mathtt{P})$. By classical weakening we obtain $\Gamma, r : C \vdash \mathtt{P}$. Applying the rule (MEAS) in Table 7, we obtain $\Gamma, x : Q \vdash \mathtt{M}[x;r].\mathtt{P}$. Moreover, $\Gamma^{Q} \subseteq \Gamma^{Q} \cup \{x\} \subseteq qv(\mathtt{P}) \cup \{x\} \subseteq qv(\mathtt{M}[x;r].\mathtt{P})$.

2. Next, we consider the classical input prefix. We assume that $c?r.\mathtt{P}$ is a *qCCS* process. By induction, we derive $\Gamma \vdash \mathtt{P}$ with $\Gamma^{Q} \subseteq qv(\mathtt{P})$. By classical weakening we obtain $\Gamma, r : C \vdash \mathtt{P}$. Applying the rule (C-IN) in Table 7, we obtain $\Gamma \vdash c?r.\mathtt{P}$. Moreover, $\Gamma^{Q} \subseteq qv(\mathtt{P}) \subseteq qv(c?r.\mathtt{P})$.

3. Consider now the classical output prefix. We assume that $c!r.\mathtt{P}$ is a *qCCS* process. By induction, we derive $\Gamma \vdash \mathtt{P}$ with $\Gamma^{Q} \subseteq qv(\mathtt{P})$. By classical weakening we obtain $\Gamma, r : C \vdash \mathtt{P}$. Applying the rule (C-OUT) in Table 7, we obtain $\Gamma, r : C \vdash c!r.\mathtt{P}$. Moreover, $\Gamma^{Q} \subseteq qv(\mathtt{P}) \subseteq qv(c!r.\mathtt{P})$.

4. At last, we consider the if operator. We assume that **if** b **then** $\mathtt{P}$ is a *qCCS* process. By induction, we derive $\Gamma \vdash \mathtt{P}$ with $\Gamma^{Q} \subseteq qv(\mathtt{P})$. Applying the rule (IF) in Table 7, we obtain $\Gamma \vdash$ **if** b **then** $\mathtt{P}$. Moreover, $\Gamma^{Q} \subseteq qv(\mathtt{P}) \subseteq qv($**if** b **then** $\mathtt{P})$.

For the converse statement we will use the following induction invariant:

$$\Gamma \vdash \mathtt{P} \text{ implies a } qCCS \text{ process } \mathtt{P} \text{ with } qv(\mathtt{P}) \subseteq \Gamma.$$

1. Starting with the measurement prefix. We assume that $\Gamma \vdash \mathtt{M}[x;r].\mathtt{P}$. This implies $x : Q \in \Gamma$ and $\Gamma \backslash \{x : Q\}, r : C \vdash \mathtt{P}$. By induction, we derive that $\mathtt{P}$ is a *qCCS* process with $qv(\mathtt{P}) \subseteq \Gamma \backslash \{x : Q\}$. By item 3 in the Enumeration 5.2, $\mathtt{M}[x;r].\mathtt{P}$ is a *qCCS* process. Moreover $qv(\mathtt{M}[x;r].\mathtt{P}) = qv(\mathtt{P}) \cup \{x\} \subseteq (\Gamma \backslash \{x : Q\}) \cup \{x\} = \Gamma$.

2. Next, we consider the classical input prefix. We assume that $\Gamma \vdash c?r.\mathtt{P}$, which entails $\Gamma \vdash \mathtt{P}$ with $r : C \in \Gamma$. By induction, we derive that $\mathtt{P}$ is a *qCCS* process with $qv(\mathtt{P}) \subseteq \Gamma$. By item 1 in Enumeration 5.2, $c?r.\mathtt{P}$ is a *qCCS* process. Moreover, $qv(c?r.\mathtt{P}) = qv(\mathtt{P}) \subseteq \Gamma$.

3. Let us now consider the classical output prefix. We assume that $\Gamma \vdash c!r.\mathtt{P}$, in which $r : C \in \Gamma$. Moreover, it implies that $\Gamma \vdash \mathtt{P}$. By induction, we derive that $\mathtt{P}$ is a *qCCS* process with $qv(\mathtt{P}) \subseteq \Gamma$. By item 2 in Enumeration 5.2, $c!r.\mathtt{P}$ is a *qCCS* process. Moreover, $qv(c!r.\mathtt{P}) = qv(\mathtt{P}) \subseteq \Gamma$.

4. At last, we consider the if operator. We assume that $\Gamma, \Delta \vdash \textbf{if } b \textbf{ then } \mathtt{P}$, so there is $\Lambda_1 \vdash \mathtt{P}$ and $\Lambda_2 \vdash b$ such that $\Lambda_1, \Lambda_2 \subseteq \Gamma, \Delta$. We have $\Lambda_1, \Lambda_2 \vdash \textbf{if } b \textbf{ then } \mathtt{P}$, which entails $\Lambda_1 \vdash \mathtt{P}$ and $\Lambda_2 \vdash b$. Here, we only care about quantum variables, so Λ_2 is not going to be considered. By induction, we derive that $\mathtt{P}$ is a *qCCS* process with $qv(\mathtt{P}) \subseteq \Lambda_1$. By item 4 in Enumeration 5.2 **if** b **then** $\mathtt{P}$ is a *qCCS* process. Moreover, $qv(\textbf{if } b \textbf{ then } \mathtt{P}) = qv(\mathtt{P}) \subseteq \Lambda_1 \subseteq \Gamma$.

□

Next, we consider some examples of quantum systems to guide the development of our work. To make things intuitive, we first consider these examples without taking into account notions of time, and later on we extend them with timing aspects.

The examples are briefly described as follows.

1. Quantum Random Number Generator

   As suggested by the name, this example encodes the generation of random numbers obtained by applying quantum effects, such as superposition and the collapse of a quantum state after measuring it. In particular, the example makes use of an Hadamard gate to put a qubit into a superposition state, and then measures the resulting qubit.

2. Quantum Cloud

   This example mimics the usual interaction pattern between a classical and a quantum computer: in processing a big, complex task the classical computer delegates computationally hard subtasks to a quantum computer. As an example of a task delegated to quantum computers, we consider the problem of determining whether certain functions $f : \{0,1\}^n \to \{0,1\}$ are constant or balanced. Deutsch-Jozsa algorithm, Nielsen and Chuang (2000), shows that this task can be performed faster on quantum computers than on classical ones.

Before presenting the examples in detail, we want to make a remark. With *qCCS* we cannot encode problems that potentially have an infinite number of quantum variables. An example of this situation is the *Quantum Random Walks*, more concretely, the discrete version of it. On a discrete quantum random walk, there are two Hilbert spaces to be considered, the *coin space* and the *position space*. Furthermore, we have two operators, *C operator* and *S operator*. The first acts on the coin space and the random movements of the

particle are decided by the results obtained from its application. The second operator acts on the position space and according to the results obtained from the application of the C operator, moves the particle. For more information related to quantum random walks see Kempe (2003). However, when one tries to encode a discrete quantum random walk we have a number of syntax and semantics restrictions. Related to the syntax, there is the impossibility to have a process that only sends quantum variables, since we have to respect the definition of constant processes, presented in Subsection 4.3.1. At the level of semantics, we have to take into account that the Hilbert space can not be expanded, this is, once we define an initial Hilbert space we cannot expand or shrink it. This is a setback, because if we want to expand the Hilbert space position, we need to start from the beginning.

For the quantum random number generator and the quantum cloud examples, we will consider a QRAM architecture, described in Knill (1996), where a classical computer sends tasks to the quantum computer solve. Furthermore, throughout the examples, c and d stands for classical communication channels, while q stands for quantum communication channels.

5.2.1 *Quantum Random Number Generator*

Random numbers are used in science, gambling, gaming and in others areas of interest. It is therefore interesting to encode a method that generates random numbers using quantum principles. Consider Figure 18, which represents a scheme of a quantum random number generator used by ID Quantique, IDQ (2016).

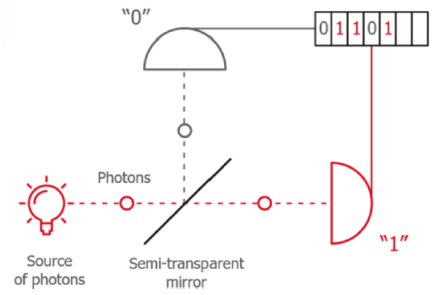


Figure 18: Scheme of a quantum random number generator

This scheme assumes a photon emitter, a beam splitter which puts photons into a superposition state, two detectors to verify the path took by photons and a memory to store the result of the measurement. When a photon is emitted, it interacts with the beam splitter. As a result of this interaction, the photon enters into a superposition state where the probabilities of being in path '0' or path '1' are equal. After the measurement, the superposition

is destroyed. Since the probability of being in path '0' or '1' is the same, the generation of random numbers is unbiased.

In the following scenario, we only encode the tasks performed by a quantum computer, because it is there the most relevant actions for this example occur.

Consider an user that wants to collect random numbers. To do that, he uses a quantum computer. After the quantum computer receives the task, $c?do$, the generation of the random numbers begins via process `Env` which sends a qubit to process `QC` to be manipulated and measured. After the measurement, the result is saved and sent to the user, who can continue to collect random numbers or stop.

Next, there is an implementation in *qCCS*.

$$\texttt{QC} \stackrel{def}{=} c?do.q?x.\texttt{H}[x].\texttt{M}[x;data].c!data.(c?stop.\texttt{QC} + d?go.\texttt{QC1})$$

$$\texttt{QC1} \stackrel{def}{=} q?x.\texttt{H}[x].\texttt{M}[x;data].c!data.(c?stop.\texttt{QC} + d?go.\texttt{QC1})$$

$$\texttt{Env} = q!x.\texttt{nil}$$

$$\texttt{QRNG} = \texttt{QC} \,||\, \texttt{Env}$$

Note that it is not possible to have a recursive definition of `Env`, because the defining equation, presented in Subsection 4.3.1, would not be obeyed. Thus, only *one* random number can be produced. To overcome this problem, we implemented a new action that creates new qubits on-demand. The new qubits are automatically initialised to $|0\rangle$. Before presenting the details of this new action, let us give the other example, which has a similar problem.

5.2.2 *Untimed Quantum Cloud*

At the current state of development, access to quantum computers is typically performed via the *cloud*, which implies communication between classical and quantum computers (e.g, the IBM quantum computer, IBM). Besides that, an order is established. This is, the classical computer is the master, because it sends a set of instructions to be executed, and the quantum computer is the slave, once it receives a set of instructions to be performed, as in the QRAM architecture.

This example illustrates an interaction pattern between a classical and a quantum computer. For this we resorted to the Deutsch algorithm. The Deutsch algorithm is a simple case of the Deutsch-Jozsa algorithm, Nielsen and Chuang (2000), aiming at verifying if a Boolean function ($f : \{0,1\} \to \{0,1\}$) is constant or balanced. A constant function outputs always '0' or '1' while a balanced function outputs '0' and '1' equally. We assume that the classical computer will send two functions, f and g, to be evaluated by the quantum computer, where:

$$f \text{ is a constant function, where } f(0) = f(1) = 0$$
$$g \text{ is a balanced function, where } g(0) = 0 \text{ and } g(1) = 1.$$

The Deutsch algorithm uses an *oracle* that is going to help us discover if the function is constant or balanced. Moreover, we have to introduce the notion of *register qubits* and *ancilla qubits*. The first ones have relevant information on them - in this case, the information presented on them tells us if the function is constant or balanced - while the second ones are auxiliary qubits used in calculations. With this, the oracle is applied to all the qubits in the system, but it only performs the following transformation to the register qubits:

$$|x\rangle \xrightarrow{\texttt{Oracle}} (-1)^{f(x)}|x\rangle$$

where x represents a qubit and $f(x)$ the value of x after applying the function f.

Although the syntax of *qCCS* does not directly allow it, we assume that in both cases it is possible to send functions through communication channels.

The functions are sent by the classical computer through the same channel ($c!f(u).c!g(v)$). After receiving the functions, the quantum computer analyses them through the process `Deutsch`. This process receives four qubits, two for each function. After executing the procedure of the Deutsch algorithm, the final result is sent to the classical computer ($c!r.c!s$), as well as a signal to tell that the algorithm is over ($c!done$).

Now, we present the definitions of the processes.

$$\texttt{User} \stackrel{def}{=} c!f(u).c!g(v).c?r.c?s.c?done.\texttt{nil}$$
$$\texttt{QC} \stackrel{def}{=} c?f(u).c?g(v).\texttt{Deutsch}$$
$$\texttt{Deutsch} \stackrel{def}{=} q?x.q?y.q?a.q?b.\texttt{X}[y,b].\texttt{H}[x,y,a,b].\texttt{Oracle}[x,y,a,b].\texttt{H}[x,a]$$
$$.\texttt{M}[x;r].\texttt{M}[a;s].c!r.c!s.c!done.nil$$
$$\texttt{Env} = q!x.q!y.q!a.q!b.\texttt{nil}$$
$$\texttt{QRAM} = \texttt{User} \,||\, \texttt{QC} \,||\, \texttt{Env}$$

Due to reasons presented earlier, the process `Env` cannot be implemented recursively.

5.2.3 *New action*

The lack of capacity for generating new qubits recursively, as shown in the two previous examples, entails the need for creating an action to overcome this gap. Therefore, we now introduce the action $new(x)$ in *qCCS*. Following the structure of *qCCS*, we have:

$$new(x).\texttt{P} \in qProc \text{ and } qv(new(x).\texttt{P}) = qv(\texttt{P}) \setminus \{x\}$$

and

$$\textbf{NEW:}\ \frac{}{\langle\ new(x).\mathtt{P},\ \rho\ \rangle \xrightarrow{\tau} \langle\ \mathtt{P}\{y/x\},\ |0\rangle_y\langle 0| \otimes \rho\ \rangle}\ y\ is\ fresh$$

where to simplify the notation, we write $|0\rangle_y\langle 0|$ instead of $|0\rangle_{yy}\langle 0|$.

This rule has a side condition to prevent sharing of qubits between different quantum processes. Consider the following example.

Example 5.2.5. Consider the process $\mathtt{P}(y) \stackrel{def}{=} new(x).\mathtt{H}[x,y].\mathtt{nil}$, which possesses qubit y.

$\mathtt{P}(y)$ is a valid process. However, when we build the transition system, we need to guarantee that the qubit created is fresh, due to the side condition on the rule **NEW**. Otherwise, if the side condition did not exist, the qubit created could be labelled with an y and, as a consequence, we would have two qubits with the same assignment.

Moreover, it creates a new qubit in the state $|0\rangle$ and expands the density operator ρ in the configuration accordingly.

In terms of the typing system:

$$\text{(NEW)}\quad \frac{\Gamma, x : Q \vdash \mathtt{P}}{\Gamma \vdash new(x).\mathtt{P}}$$

This rule tells us that when the action $new(x)$ is performed, in an environment without the quantum variable x, the environment is updated with a new qubit x. With this, we guarantee that the no-cloning theorem is obeyed, because we require that the qubit that is going to be created needs to be fresh.

As we can see, this rule is very similar to the reception of a quantum variable, since both represent the introduction of a quantum variable in the system. This can be observed perfectly in the typed system, where the two rules only differ on the name of the action. However, in terms of the semantics of *qCCS*, there is a difference on the side conditions caused by the expansion of the Hilbert space. In the *new* action, the Hilbert space is expanded and because of that we need to guarantee always a fresh qubit. In the $q?x$ action, the Hilbert space is not expanded, so it suffices to assure that the new quantum variable is not present in the process.

One may think that, since we have created an action that generates qubits, we need to create an action to destroy them. We name that action *trash* and consider it for future work.

5.2.4 *Measurement*

In *qCCS*, according to Feng et al. (2012), after performing a measurement, the collapsed state stays in the density operator. However, we do not want that, *i.e* we want to retire the

qubit measured from the density operator. Therefore, we need to change the measurement rule. In terms of *qCCS* stays:

$$\textbf{MEAS: } \frac{}{\langle\, \mathtt{M}[x;r].\mathtt{P},\, \rho \,\rangle \xrightarrow{\tau} \boxplus_{i\in I} p_i \bullet \langle\, \mathtt{P}\{r_i/x\},\ tr_x(\rho) \,\rangle}$$

where I represents the configurations created through the measurement and the notation tr_x represents the partial trace presented in Subsection 4.1.7.

As one can see, after the measurement of a state, the collapsed state is removed by using a partial trace in order of the variables that were measured.

Even knowing that information is lost (it is not possible to rebuild the original state) after executing a partial trace on entangled states, we decided to continue with this idea, since it matches what is done in the typing system.

Now, having introduced the $new(x)$ action and changing the semantics of the measurement, we are ready to present the final version of the previous examples.

5.2.5 *Quantum Random Number Generator*

This example can be simplified thanks to the introduction of the action $new(x)$: all tasks performed by the quantum computer can now be encoded by a single process, as shown below.

$$\mathtt{QRNG} \stackrel{def}{=} c?do.new(x).\mathtt{H}[x].\mathtt{M}[x;data].c!data.\mathtt{QRNG}$$

The transition system is represented below:

$$\begin{aligned}
& \langle \mathtt{QRNG},\, \rho \rangle \\
\xrightarrow{\mathtt{c?do}}\ & \langle new(x).\mathtt{H}[x].\mathtt{M}[x;data].c!data.\mathtt{QRNG},\, \rho \rangle \\
\xrightarrow{\tau}\ & \langle \mathtt{H}[y].\mathtt{M}[y;data].c!data.\mathtt{QRNG},\, |0\rangle_y\langle 0| \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle \mathtt{M}[y;data].c!data.\mathtt{QRNG},\, \frac{1}{2}(|0\rangle_y + |1\rangle_y)(\langle_y 0| + \langle_y 1|) \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \frac{1}{2}\langle c!0.\mathtt{QRNG},\, \rho \rangle \boxplus \frac{1}{2}\langle c!1.\mathtt{QRNG},\, \rho \rangle \\
\xrightarrow{c!v}\ & \frac{1}{2}\langle \mathtt{QRNG},\, \rho \rangle \boxplus \frac{1}{2}\langle \mathtt{QRNG},\, \rho \rangle
\end{aligned}$$

Through the transition system one can see the evolution of the density operator throughout the execution of actions. Moreover, we can see that after the measurement we have probabilities associated to the configurations.

5.2.6 *Quantum Cloud*

We now study two versions of the quantum cloud example, which will be more interesting when time enters into the scene. In the first version, all data was sent to the quantum computer at once. In the second version, data is divided in two; moreover, the second part of the data is only sent after receiving the result relative to the first part of the data.

To simplify the representation of density operators, we now use the notation $[|\psi_i\rangle] = |\psi_i\rangle\langle\psi_i|$ (this convention is also used in Feng et al. (2012)). Moreover, instead of writing $\mathtt{Op}[x].\mathtt{Op}[y]$, we write $\mathtt{Op}[x,y]$, where $\mathtt{Op}$ denotes a super-operator and x and y are qubits.

The first version is encoded as follows:

$$\mathtt{User} \stackrel{def}{=} c!f(u).c!g(v).c?r.c?s.c?done.\mathtt{nil}$$

$$\mathtt{QC} \stackrel{def}{=} c?f(u).c?g(v).\mathtt{Deutsch}$$

$$\mathtt{Deutsch} \stackrel{def}{=} new(x,y,a,b).\mathtt{X}[y,b].\mathtt{H}[x,y,a,b].\mathtt{Oracle}[x,y,a,b].\mathtt{H}[x,a].\mathtt{M}[x;r].\mathtt{M}[a;s].$$
$$c!r.c!s.c!done.\mathtt{nil}$$

$$\mathtt{QCloud} = \mathtt{User} \;||\; \mathtt{QC}$$

The transition system of the first version is shown below:

$$\begin{aligned}
& \langle \mathtt{QCloud}, \rho \rangle \\
\xrightarrow{\tau} \;& \langle c!g(v).c?r.c?s.c?done.\mathtt{nil} \;||\; c?g(v).\mathtt{Deutsch}, \rho \rangle \\
\xrightarrow{\tau} \;& \langle c?r.c?s.c?done.\mathtt{nil} \;||\; \mathtt{Deutsch}, \rho \rangle \\
\xrightarrow{\tau} \;& \langle c?r... \;||\; \mathtt{X}[y,b].\mathtt{H}[x,y,a,b]..., |00\rangle_{x,y}\langle 00| \otimes |00\rangle_{a,b}\langle 00| \otimes \rho \rangle \\
\xrightarrow{\tau} \;& \langle c?r... \;||\; \mathtt{H}[x,y,a,b].\mathtt{Oracle}[x,y,a,b]..., |01\rangle_{x,y}\langle 01| \otimes |01\rangle_{a,b}\langle 01| \otimes \rho \rangle \\
\xrightarrow{\tau} \;& \langle c?r.c?s.c?done.\mathtt{nil} \;||\; \mathtt{Oracle}[x,y,a,b]\mathtt{H}[x,a]..., \rho_{H(x,y)} \otimes \rho_{H(a,b)} \otimes \rho \rangle \\
\xrightarrow{\tau} \;& \langle c?r.c?s.c?done.\mathtt{nil} \;||\; \mathtt{H}[x,a].\mathtt{M}[x;r]..., \rho_{Oracle(x,y)} \otimes \rho_{Oracle(a,b)} \otimes \rho \rangle \\
\xrightarrow{\tau} \;& \langle c?r.c?s.c?done.\mathtt{nil} \;||\; \mathtt{M}[x;r].\mathtt{M}[a;s]..., \rho_{H(x)} \otimes \rho_{H(a)} \otimes \rho' \rangle \\
\xrightarrow{\tau} \;& \langle c?r.c?s.c?done.\mathtt{nil} \;||\; \mathtt{M}[a;s].c!0..., \rho_{H(a)} \otimes \rho' \rangle \\
\xrightarrow{\tau} \;& \langle c?r.c?s.c?done.\mathtt{nil} \;||\; c!0.c!1.c!done.\mathtt{nil}, \rho' \rangle \\
\xrightarrow{\tau} \;& \langle c?s.c?done.\mathtt{nil} \;||\; c!1.c!done.\mathtt{nil}, \rho' \rangle \\
\xrightarrow{\tau} \;& \langle c?done.\mathtt{nil} \;||\; c!done.\mathtt{nil}, \rho' \rangle \\
\xrightarrow{\tau} \;& \langle \mathtt{nil} \;||\; \mathtt{nil}, \rho' \rangle
\end{aligned}$$

where:

$$\rho_{H(x,y)} = \frac{1}{\sqrt{2}}[(|0\rangle_x + |1\rangle_x)] \cdot \frac{1}{\sqrt{2}}[(|0\rangle_y] - |1\rangle_y)];$$

$$\rho_{H(a,b)} = \frac{1}{\sqrt{2}}[|0\rangle_a + |1\rangle_a] \cdot \frac{1}{\sqrt{2}}[|0\rangle_b] - |1\rangle_b];$$

$$\rho_{Oracle(x,y)} = \frac{1}{\sqrt{2}}[(-1)^{f(0)}|0\rangle_x + (-1)^{f(1)}|1\rangle_x] \cdot \frac{1}{\sqrt{2}}[|0\rangle_y - |1\rangle_y];$$
$$\rho_{Oracle(a,b)} = \frac{1}{\sqrt{2}}[(-1)^{f(0)}|0\rangle_a + (-1)^{f(1)}|1\rangle_a] \cdot \frac{1}{\sqrt{2}}[|0\rangle_b - |1\rangle_b];$$
$$\rho_{H(x)} = \frac{1}{2}[((-1)^{f(0)} + (-1)^{f(1)})|0\rangle_x + ((-1)^{f(0)} - (-1)^{f(1)})|1\rangle_x] \cdot \frac{1}{\sqrt{2}}[|0\rangle_y - |1\rangle_y];$$
$$\rho_{H(a)} = \frac{1}{2}[((-1)^{f(0)} + (-1)^{f(1)})|0\rangle_a + ((-1)^{f(0)} - (-1)^{f(1)})|1\rangle_a] \cdot \frac{1}{\sqrt{2}}[|0\rangle_b - |1\rangle_b];$$
$$\rho' = \frac{1}{\sqrt{2}}[|0\rangle_y - |1\rangle_y] \otimes \frac{1}{\sqrt{2}}[|0\rangle_b - |1\rangle_b] \otimes \rho$$

The encoding of the second version is presented below:

$$\texttt{User} \stackrel{def}{=} c!f(u).c?r.c!g(v).c?s.\texttt{nil}$$
$$\texttt{QC} \stackrel{def}{=} c?f(u).\texttt{Deutsch}$$
$$\texttt{Deutsch} \stackrel{def}{=} new(x,y).\texttt{X}[y].\texttt{H}[x,y].\texttt{Oracle}[x,y].\texttt{H}[x].\texttt{M}[x;r].c!s.\texttt{QC}$$
$$\texttt{QCloud} = \texttt{User} \,||\, \texttt{QC}$$

The respective transition system is as follows:

$\langle \texttt{QCloud}, \rho \rangle$
$\xrightarrow{\tau}$ $\langle c!f(u).c?r.c!g(v).c?s.\texttt{nil} \,||\, c?f(u).Deutsch, \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?r.c!g(v).c?s.\texttt{nil} \,||\, \texttt{Deutsch}, \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?r.c!g(v).c?s.\texttt{nil} \,||\, \texttt{X}[y].\texttt{X}[x,y]..., [|00\rangle_{xy}] \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?r.c!g(v).c?s.\texttt{nil} \,||\, \texttt{H}[x,y].\texttt{Oracle}[x,y]..., [|01\rangle_{xy}] \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?r.c!g(v).c?s.\texttt{nil} \,||\, \texttt{Oracle}[x,y].\texttt{H}[x]..., \rho_{H(x,y)} \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?r.c!g(v).c?s.\texttt{nil} \,||\, \texttt{H}[x].\texttt{M}[x;r].c!s.\texttt{QC}, \rho_{Oracle(x,y)} \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?r.c!g(v).c?s.\texttt{nil} \,||\, \texttt{M}[x;r].c!s.\texttt{QC}, \rho_{H(x)} \otimes \rho_y \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?r.c!g(v).c?s.\texttt{nil} \,||\, c!0.\texttt{QC}, \rho_y \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c!g(v).c?s.\texttt{nil} \,||\, \texttt{QC}, \rho_y \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?s.\texttt{nil} \,||\, \texttt{Deutsch}, \rho_y \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?r.c!g(v).c?s.\texttt{nil} \,||\, \texttt{X}[a].\texttt{H}[x,a]..., [|00\rangle_{xa}] \otimes \rho_a \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?s.\texttt{nil} \,||\, \texttt{H}[x,a].\texttt{Oracle}[x,a]..., [|01\rangle_{xa}] \otimes \rho_y \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?s.\texttt{nil} \,||\, \texttt{Oracle}[x,a].\texttt{H}[x]..., \rho_{H(x,a)} \otimes \rho_y \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?s.\texttt{nil} \,||\, \texttt{H}[x].\texttt{M}[x;r].c!s.\texttt{QC}, \rho_{Oracle(x,a)} \otimes \rho_y \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?s.\texttt{nil} \,||\, \texttt{M}[x;r].c!s.\texttt{QC}, \rho_{H(x)} \otimes \rho_a \otimes \rho_y \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle c?s.\texttt{nil} \,||\, c!1.\texttt{QC}, \rho_a \otimes \rho_y \otimes \rho \rangle$
$\xrightarrow{\tau}$ $\langle \texttt{nil} \,||\, \texttt{QC}, \rho_a \otimes \rho_y \otimes \rho \rangle$

As mentioned above, the difference between these examples is more interesting when time comes to play. Then, we will be able to reason about which example is more efficient or which one guarantees the coherence of qubits.

5.3 TIMED QUANTUM CCS

With the introduction of time in classical process algebras, it becomes possible to trigger actions according to temporal constraints and verify time-based protocols more accurately. In the quantum setting the existence of actions triggered by time is also important. Moreover, one cannot forget that qubits have a very restricted time limit during which they are immune to noise. This means that all operations must be performed within this time limit. So in quantum process algebras, time also plays a key role in ensuring that a protocol or algorithm behaves as expected.

In the theory developed for us, we designed a typing system that verifies if the no-cloning theorem is obeyed.

We store the information related to the lifetime of qubits in process configurations, due to a problem in a rule of the typing system. This choice will be discussed later. Relatively to the temporal properties, shown in Chapter 3, we present our choices in Table 8.

Time Domain	Discrete or Dense
Measurement of Time	Absolute
Execution Sequence	First phase of the two-phase sequence
Introduction of time	Delay action
Model Properties	Time determinism Time additivity Maximal Progression

Table 8: Temporal choices for the quantum process algebra

Our process algebra is designed to support either a discrete or a dense time domain. Independently of the time domain chosen, our goal is to study the impact that time progression has in quantum processes. We thus consider that the measurement of time is absolute, meaning that we assume the existence of a global clock for all processes. In this way, we can have a better control of the lifetime of qubits. Now, recall the Figure 16 that represents an execution sequence of two-phases, where in the first phase processes perform synchronous and asynchronous actions, and in the second phase time elapses. For our theory, we only want to have the first phase. With this, the configurations are always performing actions and time is elapsed when a delay occurs. Moreover, after each action the configuration is updated. The introduction of time is made by a delay action, $\sigma(t)$, where t denotes the time delay. Relatively to the model properties, we chose time determinism because we do not want time to interfere with the non-deterministic operator; time additivity because a

configuration must reach the same configuration independently of the delay $\sigma(t)$ being performed once or n-times $\sigma(t_i)$, where $\sum_{i=1}^{n} t_i = t$; maximal progression because information is transmitted by communication, so the faster communication occurs the faster information is transmitted. We did not chose persistency, because "*it seems counter-intuitive to not allow time to change system's capabilities to perform actions once time is often used as a parameter to control action executability - as in timeouts*" (quote from Nicollin and Sifakis (1992)).

In the following subsections, we will outline the syntax, the operational semantics and the typing system of our timed-quantum process algebra.

5.3.1 *Syntax*

We consider as time domain, $\mathcal{T}$, the non-negative real numbers $\mathcal{R}_0^+$. We also consider as data types `Bool` for Booleans, the real numbers `Real` for classical variables, the non-negative real numbers $\texttt{Real}_0^+$ for time delay, and `Qbt` for qubits. We contemplate the following sets:

- $cVar$ for classical variables (countably infinite)

  ranged over by $r, s, ...$

- $qVar$ for quantum variables (countably infinite)

  ranged over by $x, y, ...$

- Exp for classical data expressions over `Real`, where $cVar \subseteq Exp$

  ranged over by $e, e', ...$

- $BExp$ for Boolean-valued expressions

  ranged over by $b, b', ...$ with the usual Boolean operators: `true`, `false`, $\neg$, $\wedge$, $\vee$ and $\rightarrow$

  a Boolean expression is of the form $e \bowtie e'$, where $e, e' \in Exp$ and $\bowtie \in \{<, >, \leq, \geq, =\}$

- $cChan$ for classical communication channels

  ranged over by $c, d, ...$

- $qChan$ for quantum communication channels

  ranged over by $q, ...$

- $Chan = cChan \cup qChan$ for communication channels

- $\mathcal{C} = \{c?r, c!r \mid c \in cChan,\ r \in cVar\}$ for classical communication actions

- $\mathcal{Q} = \{q?x, q!x \mid q \in qChan,\ x \in qVar\}$ for quantum communication actions

- $\delta_{\mathcal{T}} = \{\sigma(t) \mid t \in \mathcal{T} - \{0\}\}$ for time actions
  ranged over by t, t'...
- $\{\tau\}$ for invisible actions
- $\mathcal{A} = \mathcal{C} \cup \mathcal{Q} \cup \{\tau\} \cup \delta_{\mathcal{T}}$ for actions

We are now ready to present the syntax of *Timed Quantum CCS*, in short *TqCCS*.

Definition 5.3.1. The syntax of *TqCCS* is given by the following grammar:

$$\mathtt{P},\mathtt{Q} ::= \mathtt{A}(\tilde{x}) \mid \mathtt{nil} \mid \tau.\mathtt{P} \mid new(x).\mathtt{P} \mid \sigma(t).\mathtt{P} \mid \varepsilon[X].\mathtt{P} \mid c?r.\mathtt{P} \mid c!r.\mathtt{P} \mid q?x.\mathtt{P} \mid q!x.\mathtt{P} \mid \mathtt{M}[x;r].\mathtt{P} \mid \mathtt{P} + \mathtt{Q} \mid \mathtt{P} \mid\mid \mathtt{Q} \mid \mathtt{P}\backslash\mathtt{L} \mid \textbf{if } b \textbf{ then } \mathtt{P}$$

where:

1. $\mathtt{A}(\tilde{x})$: is the set of constant processes with non-negative arity
2. $\mathtt{nil}$: represents a state with no outgoing transitions
3. $\tau.\mathtt{P}$: represents the silent action
4. $new(x).\mathtt{P}$: is the action that creates a new qubit
5. $\sigma(t).\mathtt{P}$: delays t units of time the execution of a process
6. $\varepsilon[X].\mathtt{P}$: can be understood as the action prefixing of *CCS* and corresponds to the application of a super-operator to a set of qubits
7. $c?r$: receives a classical variable
8. $c!r$: sends a classical variable
9. $q?x$: receives a quantum variable
10. $q!x$: sends a quantum variable
11. $\mathtt{M}[x;r].\mathtt{P}$: represents the measurement of a quantum variable and the storage of the measurement result on a classical variable
12. $\mathtt{P} + \mathtt{Q}$: non-deterministic choice operator between processes
13. $\mathtt{P} \mid\mid \mathtt{Q}$: parallel communication of processes
14. $\mathtt{P}\backslash\mathtt{L}$: hides all channels names in $\mathtt{L}$

15. **if** b **then** P: represents an if clause

Here, we only present the formation rule of $\sigma(t).\mathrm{P}$.

$$\text{if } \mathrm{P} \in \mathcal{P}, \text{ then } \sigma(t).\mathrm{P} \in \mathcal{P} \text{ and } qv(\sigma(t).\mathrm{P}) = qv(\mathrm{P})$$

The remaining formation rules are absent, because the formation rule of the *new* action was already presented in Subsection 5.2.3 and the other rules are the same as in Feng et al. (2012).

5.3.2 *Operational Semantics*

By adding time to the quantum process algebra, we felt the necessity to include the lifetime of qubits in the configurations. Due to that, the configuration is going to be expanded with qubits lifetime, denoted generally as t_q. Therefore, a configuration is represented as follows:

$$\langle \text{Process; Qubits lifetime; Density Operator} \rangle$$

For the majority of the *qCCS* rules previously shown, pure quantum (Section 4.3) and expanded (Section 5.2), we only need to add qubits lifetime.

Tau: $$\frac{}{\langle \tau.\mathrm{P};\ t_q;\ \rho \rangle \xrightarrow{\tau} \langle \mathrm{P};\ t_q;\ \rho \rangle}$$

C-Inp: $$\frac{}{\langle\ c?r.\mathrm{P};\ t_q;\ \rho\ \rangle \xrightarrow{c?s} \langle\ \mathrm{P}\{s/r\};\ t_q;\ \rho\ \rangle}$$

C-Outp: $$\frac{}{\langle\ c!r.\mathrm{P};\ t_q;\ \rho\ \rangle \xrightarrow{c!r} \langle\ \mathrm{P};\ t_q;\ \rho\ \rangle}$$

C-Com: $$\frac{\langle\ \mathrm{P}_1;\ t_q;\ \rho\ \rangle \xrightarrow{c?r} \langle\ \mathrm{P}_1';\ t_q;\ \rho\ \rangle \qquad \langle\ \mathrm{P}_2;\ t_q;\ \rho\ \rangle \xrightarrow{c!r} \langle\ \mathrm{P}_2';\ t_q;\ \rho\ \rangle}{\langle\ \mathrm{P}_1\ ||\ \mathrm{P}_2;\ t_q;\ \rho\ \rangle \xrightarrow{\tau} \langle\ \mathrm{P}_1'\ ||\ \mathrm{P}_2';\ t_q;\ \rho\ \rangle}$$

Q-Inp: $$\frac{}{\langle\ q?x.\mathrm{P};\ t_q;\ \rho\ \rangle \xrightarrow{q?y} \langle\ \mathrm{P}\{y/x\};\ t_q;\ \rho\ \rangle}\ y \notin qv(q?x.\mathrm{P})$$

Q-Outp: $$\frac{}{\langle\ q!x.\mathrm{P};\ t_q;\ \rho\ \rangle \xrightarrow{q!x} \langle\ \mathrm{P};\ t_q;\ \rho\ \rangle}$$

Q-Com: $$\frac{\langle\ \mathrm{P}_1;\ t_q;\ \rho\ \rangle \xrightarrow{q?x} \langle\ \mathrm{P}_1';\ t_q;\ \rho\ \rangle \qquad \langle\ \mathrm{P}_2;\ t_q;\ \rho\ \rangle \xrightarrow{q!x} \langle\ \mathrm{P}_2';\ t_q;\ \rho\ \rangle}{\langle\ \mathrm{P}_1\ ||\ \mathrm{P}_2;\ t_q;\ \rho\ \rangle \xrightarrow{\tau} \langle\ \mathrm{P}_1'\ ||\ \mathrm{P}_2';\ t_q;\ \rho\ \rangle}$$

Oper: $$\frac{}{\langle\ \varepsilon[X].\mathrm{P};\ t_q;\ \rho\ \rangle \xrightarrow{\tau} \langle\ \mathrm{P};\ t_q;\ \varepsilon_X(\rho)\ \rangle}$$

Meas: $$\frac{}{\langle\, \mathtt{M}[x;r].\mathtt{P};\, t_q, t_x;\ \rho\,\rangle \xrightarrow{\tau} \boxplus_{i\in I} p_i \bullet \langle\, \mathtt{P}\{r_i/x\};\, t_q;\ tr_x(\rho)\,\rangle}$$

Sum: $$\frac{\langle\, \mathtt{P};\, t_q;\ \rho\,\rangle \xrightarrow{\alpha} \mu}{\langle \mathtt{P} + \mathtt{Q};\, t_q;\ \rho\rangle \xrightarrow{\alpha} \mu}$$

Rel: $$\frac{\langle\, \mathtt{P};\, t_q;\, \rho\,\rangle \xrightarrow{\alpha} \boxplus_{i\in I} p_i \bullet \langle\, \mathtt{P_i};\ t_q;\, \rho_i\,\rangle}{\langle\, \mathtt{P}[\mathtt{f}];\, t_q;\, \rho\,\rangle \xrightarrow{f(\alpha)} \boxplus_{i\in I} p_i \bullet \langle\, \mathtt{P_i}[\mathtt{f}];\ t_q;\, \rho_i\,\rangle}$$

Res: $$\frac{\langle\, \mathtt{P};\, t_q;\, \rho\,\rangle \xrightarrow{\alpha} \boxplus_{i\in I} p_i \bullet \langle\, \mathtt{P_i};\ t_q;\, \rho_i\,\rangle}{\langle\, \mathtt{P}\backslash \mathtt{L};\, t_q;\, \rho\,\rangle \xrightarrow{\alpha} \boxplus_{i\in I} p_i \bullet \langle\, \mathtt{P_i}\backslash \mathtt{L};\ t_q;\, \rho_i\,\rangle}\ cn(\alpha) \notin L$$

If: $$\frac{\langle\, \mathtt{P};\, t_q;\ \rho\,\rangle \xrightarrow{\alpha} \mu}{\langle \textbf{if } b \textbf{ then } \mathtt{P};\, t_q;\ \rho\rangle \xrightarrow{\alpha} \mu}\ [\![b]\!] = true$$

Def: $$\frac{\langle\, \mathtt{P}\{\tilde{y}/\tilde{x}\};\, t_q;\ \rho\,\rangle \xrightarrow{\alpha} \mu}{\langle \mathtt{A}(\tilde{y});\, t_q;\ \rho\rangle \xrightarrow{\alpha} \mu}\ \mathtt{A}(\tilde{x}) \overset{def}{=} \mathtt{P}$$

New: $$\frac{}{\langle\, new(x).\mathtt{P};\, t_q;\ \rho\rangle \xrightarrow{\tau} \langle\, \mathtt{P}\{y/x\};\, t_q, 0_y;\ |0\rangle_y\langle 0| \otimes \rho\,\rangle}\ y \text{ is fresh}$$

Inp-Int: $$\frac{\langle\, \mathtt{P}_1;\, t_q;\, \rho\,\rangle \xrightarrow{q?x} \langle\, \mathtt{P}_1';\ t_q, t_x;\, \rho\,\rangle}{\langle\, \mathtt{P}_1 \,||\, \mathtt{P}_2;\, t_q;\, \rho\,\rangle \xrightarrow{q?x} \langle\, \mathtt{P}_1' \,||\, \mathtt{P}_2;\ t_q, t_x;\, \rho\,\rangle}\ x \notin qv(\mathtt{P}_2)$$

New-Int: $$\frac{\langle\, \mathtt{P}_1; t_{\mathtt{P}_1}; t_q; \rho\,\rangle \xrightarrow{\tau} \langle\, \mathtt{P}_1'; t_{\mathtt{P}_1}; t_q, 0_x; |0\rangle_x\langle 0| \otimes \rho\,\rangle}{\langle\, \mathtt{P}_1 \,||\, \mathtt{P}_2; t_{\mathtt{P}_1}, t_{\mathtt{P}_2}; t_q; \rho\,\rangle \xrightarrow{\tau} \langle\, \mathtt{P}_1' \,||\, \mathtt{P}_2; t_{\mathtt{P}_1}, t_{\mathtt{P}_2}; t_q, 0_x; |0\rangle_x\langle 0| \otimes \rho\,\rangle}\ x \text{ is fresh}$$

Oth-Int: $$\frac{\langle\, \mathtt{P}_1;\, t_q;\, \rho\,\rangle \xrightarrow{\alpha} \boxplus_{i\in I} p_i \bullet \langle\, \mathtt{P}_1';\ t_q;\, \rho_i\,\rangle}{\langle\, \mathtt{P}_1 \,||\, \mathtt{P}_2;\, t_q;\, \rho\,\rangle \xrightarrow{\alpha} \boxplus_{i\in I} p_i \bullet \langle\, \mathtt{P}_1' \,||\, \mathtt{P}_2;\ t_q;\, \rho_i\,\rangle}\ \alpha \neq q?x \wedge \alpha \neq new(x)$$

Wait: $$\frac{}{\langle\, \sigma(\mathtt{t}).\mathtt{P};\, t_q;\ \rho\,\rangle \xrightarrow{\tau} \langle\, \mathtt{P};\, t_q + t;\ \rho\,\rangle}$$

T-Add: $$\frac{}{\langle\, \sigma(t+u).\mathtt{P};\ t_q;\ \rho\,\rangle \xrightarrow{\tau} \langle\, \sigma(u).\mathtt{P},\ t_q + t;\ \rho\,\rangle}$$

Table 9: *SOS* rules for *TqCCS*

In Table 9, the symmetric versions of **Inp-Int**, **New-Int**, **Oth-Int** and **Sum** are not presented. With the introduction of time on configurations, we need to define what happens when qubits are exchanged. Reasonably, when a qubit is exchanged, sent or received, all the information associated to that qubit must follow it. This can be seen in **Q-Inp**, **Q-Outp**, **Q-Com** and **Inp-Int**. In **New**, the time of the qubit created is set to 0, and the state of the qubit is $|0\rangle$. In **Wait**, we sum t units of time to the lifetime of the qubits. Notice that if we do not have any qubit in the system and a delay action happens, then the lifetime of qubits remains unchanged. The side conditions presented in the rules **Inp-Int** and **New-Int** are necessary to prevent the no-cloning theorem. In the former, we need to make sure that the process $\mathtt{P}_2$ does not have the quantum variable x, because we cannot have the same variable in two different processes. In the latter, we need to assure that the qubit is fresh, also because we cannot have a variable being shared in different processes.

The rule **New-Int** tells us that a new qubit can be created if it is fresh in $\mathtt{P}_1$ and $\mathtt{P}_1 \parallel \mathtt{P}_2$. In **Oth-Int**, we just want to assure that the actions that receives or creates qubits can not be executed by this rule, once specific rules for them were developed.

Now, we will discuss why the qubits lifetime is presented in the transition system instead of the typing system. In the former, the quantum system has information of all the transitions between configurations, since we are using a *LTS*. Therefore, when a qubit is shared, we have access to the information related with the lifetime of qubits. In the latter, when a qubit is received, no information related to time is known. That is because, in the typing system, processes do not know the existence of others processes.

After presenting the syntax, the semantics and justifying why the information related with time is presented in configurations, we can specify the model properties that we wish our theory to respect. In this dissertation we will not prove that our process algebra has desirable model properties. However we would like to present next some of the properties that we intend to show in the near future.

- Time determinism

  If $\langle \mathtt{P};\ t_q;\ \rho\rangle \xrightarrow{\tau} \langle \mathtt{P}_1;\ t_q + t;\ \rho\rangle$ and $\langle \mathtt{P};\ t_q;\ \rho\rangle \xrightarrow{\tau} \langle \mathtt{P}_2;\ t_q + t;\ \rho\rangle$, then $\langle \mathtt{P}_1;\ t_q + t;\ \rho\rangle = \langle \mathtt{P}_2;\ t_q + t;\ \rho\rangle$

  where $=$ is the syntactic equality

- Maximal progress

  If $\langle \mathtt{P};\ t_q;\ \rho\rangle \xrightarrow{\tau} \langle \mathtt{P}_1;\ t_q;\ \rho\rangle$ then $\langle \mathtt{P};\ t_q;\ \rho\rangle \not\xrightarrow{\tau} \langle \mathtt{P}_2;\ t_q + t;\ \rho\rangle$

- Time additivity

  Described by the rule (T-Add) in Table 9

5.3.3 *Typing System*

Previously, we developed a typing system to verify if the no-cloning theorem was respected or not. The typing system could not possess any information about the lifetime of qubits, because when a qubit is received "from the outside" nothing is known of its lifetime. This particular problem is found in the (Q-IN) rule. In this dissertation, we present instead the following rules for the typing system of our timed-quantum process algebra.

$$\text{(NIL)}\quad \frac{}{\Gamma \vdash nil} \qquad \text{(CONST)}\quad \frac{\Delta \subseteq \Gamma}{\Gamma \vdash \mathtt{A}(\Delta)}$$

$$\text{(INV)}\quad \frac{\Gamma \vdash \mathtt{P}}{\Gamma \vdash \tau.\mathtt{P}} \qquad \text{(OP)}\quad \frac{\Gamma \vdash \mathtt{P} \quad X \subseteq \Gamma^{Q}}{\Gamma \vdash \varepsilon[X].\mathtt{P}}$$

$$\text{(C-OUT)}\quad \frac{\Gamma, r : C \vdash \mathtt{P}}{\Gamma, r : C \vdash c!r.\mathtt{P}} \qquad \text{(C-IN)}\quad \frac{\Gamma, r : C \vdash \mathtt{P}}{\Gamma \vdash c?r.\mathtt{P}}$$

$$\text{(Q-IN)}\quad \frac{\Gamma, x : Q \vdash \mathtt{P}}{\Gamma \vdash q?x.\mathtt{P}} \qquad \text{(Q-OUT)}\quad \frac{\Gamma \vdash \mathtt{P}}{\Gamma, x : Q \vdash q!x.\mathtt{P}}$$

$$\text{(MEAS)}\quad \frac{\Gamma, r : C \vdash \mathtt{P}}{\Gamma, x : Q \vdash \mathtt{M}[x; r].\mathtt{P}} \qquad \text{(SUM)}\quad \frac{\Gamma_1 \vdash \mathtt{P} \qquad \Gamma_2 \vdash \mathtt{Q}}{\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} + \mathtt{Q}}$$

$$\text{(REL)}\quad \frac{\Gamma \vdash \mathtt{P}}{\Gamma \vdash \mathtt{P}[f]} \qquad \text{(RES)}\quad \frac{\Gamma \vdash \mathtt{P}}{\Gamma \vdash \mathtt{P}\backslash \mathtt{L}}$$

$$\text{(IF)}\quad \frac{\Gamma \vdash \mathtt{P} \quad \Delta \vdash b}{\Gamma, \Delta \vdash \textbf{if } b \textbf{ then } \mathtt{P}} \qquad \text{(NEW)}\quad \frac{\Gamma, x : Q \vdash \mathtt{P}}{\Gamma \vdash new(x).\mathtt{P}}$$

$$\text{(COMM)}\quad \frac{\Gamma_1 \vdash \mathtt{P} \quad \Gamma_2 \vdash \mathtt{Q} \quad \Gamma_1^{Q} \cap \Gamma_2^{Q} = \emptyset}{\Gamma_1 \cup \Gamma_2 \vdash \mathtt{P} \mid\mid \mathtt{Q}} \qquad \text{(WAIT)}\quad \frac{\Gamma \vdash \mathtt{P}}{\Gamma \vdash \sigma(t).\mathtt{P}}$$

Table 10: *TqCCS* typing rules

In Section 4.4 we proved that the typing system admits the weakening rule and Theorem 4.4.2. In Section 5.2 we proved that with the addition of classical variables the typing system still has implicitly the weakening rule, classical and quantum, and that Theorem 7 holds. Here, we need to prove that the typing system admits the weakening rule for both the classic and quantum cases, and that there is an equivalence between the processes defined in Definition 5.3.1 and the rules in Table 10. Actually, we only need to exhibit the proofs for formulas (NEW) and (WAIT), since the remaining proofs are very similar to the ones done previously.

The classical weakening, in *TqCCS*, is defined as follows:

Theorem 5.3.2. Assume that $\Gamma \vdash \mathtt{P}$. Then $\Gamma, x : C \vdash \mathtt{P}$ for any classical variable x.

The proof is shown next.

Proof. The proof follows by induction on the structure of quantum processes.

1. We consider the case of the action *new*. We assume that $\Gamma \vdash new(x).\mathtt{P}$, which implies $\Gamma, x : Q \vdash \mathtt{P}$. By induction we derive $\Gamma, x : Q, r : C \vdash \mathtt{P}$. Applying the rule (NEW) in Table 10 we obtain $\Gamma, r : C \vdash new(x).\mathtt{P}$.

2. Now, we consider the case of the delay actions. We assume that $\Gamma \vdash \sigma(t).\mathtt{P}$, which entails $\Gamma \vdash \mathtt{P}$. By induction, we derive $\Gamma, r : C \vdash \mathtt{P}$. Applying the rule (WAIT) in Table 10 we obtain $\Gamma, r : C \vdash \sigma(t).\mathtt{P}$.

□

Now, we need to deal with the quantum case. For *TqCCS* this theorem is as follows:

Theorem 5.3.3. Assume that $\Gamma \vdash \mathtt{P}$. Then $\Gamma, x : Q \vdash \mathtt{P}$ for any quantum variable x.

Proof. The proof follows by induction on the structure of quantum processes.

1. We consider the case of the action *new*. We assume that $\Gamma \vdash new(x).\mathtt{P}$, which implies $\Gamma, x : Q \vdash \mathtt{P}$. By induction we derive $\Gamma, x : Q, y : Q \vdash \mathtt{P}$. Applying the rule (NEW) in Table 10 we obtain $\Gamma, y : Q \vdash new(x).\mathtt{P}$.

2. Now, we consider the case of the delay actions. We assume that $\Gamma \vdash \sigma(t).\mathtt{P}$, which entails $\Gamma \vdash \mathtt{P}$. By induction, we derive $\Gamma, x : Q \vdash \mathtt{P}$. Applying the rule (WAIT) in Table 10 we obtain $\Gamma, x : Q \vdash \sigma(t).\mathtt{P}$.

□

We have proved that the quantum weakening is also admitted by our typing system. Finally, we need to formulate a theorem that establishes a relation between Table 10 and Table 5.3.1. The theorem is as follows:

Theorem 5.3.4. $\mathtt{P}$ is a process in *TqCCS* iff $\Gamma \vdash \mathtt{P}$ for some Γ.

Again, we will only show the proof for the formulas (NEW) and (WAIT).

Proof. We begin by showing the left-to-right implication. The induction invariant is as follows:

$$\mathtt{P} \text{ is a process in } \textit{TqCCS} \text{ implies that } \Gamma \vdash \mathtt{P} \text{ with } \Gamma^{Q} \subseteq qv(\mathtt{P}).$$

1. Consider the case of the action *new*. We assume that $new(x).\mathtt{P}$ is a process in *TqCCS*. By induction we derive $\Gamma \vdash \mathtt{P}$ with $\Gamma^Q \subseteq qv(\mathtt{P})$. Applying the rule (NEW) in Table 10 we obtain $\Gamma\backslash\{x:Q\} \vdash new(x).\mathtt{P}$ with $\Gamma^Q\backslash\{x:Q\} \subseteq qv(\mathtt{P})\backslash\{x\} \subseteq qv(new(x).\mathtt{P})$.

2. Let us now consider the case of the delay action. We assume that $\sigma(t).\mathtt{P}$ is a *TqCCS* process. By induction we derive $\Gamma \vdash \mathtt{P}$ with $\Gamma^Q \subseteq qv(\mathtt{P})$. Applying the rule (WAIT) in Table 10 we obtain $\Gamma \vdash \sigma(t).\mathtt{P}$ with $\Gamma^Q \subseteq qv(\mathtt{P}) \subseteq qv(\sigma(t).\mathtt{P})$.

Let us now show the right-to-left implication. The induction invariant is now:

$$\Gamma \vdash \mathtt{P} \text{ entails a } \textit{TqCCS} \text{ process } \mathtt{P} \text{ with } qv(\mathtt{P}) \subseteq \Gamma$$

1. Consider the action *new*. We assume $\Gamma \vdash new(x).\mathtt{P}$, which entails $\Gamma, x:Q \vdash \mathtt{P}$. By induction we derive that $\mathtt{P}$ is a process in *TqCCS* with $qv(\mathtt{P}) \subseteq \Gamma, x:Q$. By item 4 in Definition 5.3.1 $new(x).\mathtt{P}$ is a process in *TqCCS*. Moreover $qv(new(x).\mathtt{P}) \subseteq qv(\mathtt{P})\backslash\{x\} \subseteq \Gamma\backslash\{x\}$.

2. At last, consider the delay action. We assume $\Gamma \vdash \sigma(t).\mathtt{P}$, which entails $\Gamma \vdash \mathtt{P}$. By induction we derive that $\mathtt{P}$ is a process in *TqCCS* with $qv(\mathtt{P}) \subseteq \Gamma$. By item 5 in Definition 5.3.1 $\sigma(t).\mathtt{P}$ is a *TqCCS* process. Moreover $qv(\sigma(t).\mathtt{P}) = qv(\mathtt{P}) \subseteq \Gamma$.

□

5.3.4 *Timed Quantum Random Number Generator*

In the quantum random number generator, the inclusion of time dimension aims at controlling the amount of time the system is collecting data. It also serves to verify if the coherence time was exceeded or not.

In order to mimic quantum computers producing non-instantaneous computations, we consider a delay action, which happens before the measurement and after putting a qubit into superposition. This delay can be seen as the time elapsed from the moment the order *do* was received until performing the Hadamard operation on the created qubit, $\mathtt{H}[x]$. Furthermore, when the time delay t is big, problems in imperfect quantum systems arose. Such problems are related to the decoherence of qubits, since they are not kept stable, and therefore more susceptible to noise.

We now present the corresponding QRNG process:

$$\mathtt{QRNG} \stackrel{def}{=} c?do.new(x).\mathtt{H}[x].\sigma(t).\mathtt{M}[x;r].c!r.\mathtt{QRNG}$$

Before presenting the transition system, we need to verify that this process does not violate the no-cloning theorem. Therefore, we need to use the typing system.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\emptyset \subseteq do : C, r : C}}{do : C, r : C \vdash \texttt{QRNG}}}{do : C, r : C \vdash c!r.\texttt{QRNG}}}{x : Q, do : C \vdash \texttt{M}[x;r].c!r.\texttt{QRNG}}}{x : Q, do : C \vdash \sigma(t).\texttt{M}[x;r].c!r.\texttt{QRNG} \quad x : Q \subseteq \{x : Q\}}}{x : Q, do : C \vdash \texttt{H}[x].\sigma(t).\texttt{M}[x;r].c!r.\texttt{QRNG}}}{do : C \vdash new(x).\texttt{H}[x].\sigma(t).\texttt{M}[x;r].c!r.\texttt{QRNG}}}{\emptyset \vdash c?do.new(x).\texttt{H}[x].\sigma(t).\texttt{M}[x;r].c!r.\texttt{QRNG} \quad \emptyset \subseteq \emptyset}}{\emptyset \vdash \texttt{QRNG}}$$

Since the derivation was successful, we conclude that the no-cloning theorem was not violated. Therefore, the process is valid and its execution be calculated as follows,

$$\begin{aligned}
& \langle \texttt{QRNG}; \emptyset; \rho \rangle \\
\xrightarrow{\texttt{c?do}} \quad & \langle new(x).\texttt{H}[x].\sigma(t).\texttt{M}[x;r].c!r.\texttt{QRNG}; \emptyset; \rho \rangle \\
\xrightarrow{\tau} \quad & \langle \texttt{H}[x].\sigma(t).\texttt{M}[x;r].c!r.\texttt{QRNG}; 0_y; |0\rangle_y\langle 0| \otimes \rho \rangle \\
\xrightarrow{\tau} \quad & \langle \sigma(t).\texttt{M}[x;r].c!r.\texttt{QRNG}; 0_y; \frac{1}{2}(|0\rangle_y + |1\rangle_y)(\langle_y 0| + \langle_y 0|) \otimes \rho \rangle \\
\xrightarrow{\tau} \quad & \langle \texttt{M}[x;r].c!r.\texttt{QRNG}; \mathfrak{t}_y; \frac{1}{2}(|0\rangle_y + |1\rangle_y)(_y\langle 0| +_y \langle 1|) \otimes \rho \rangle \\
\xrightarrow{\tau} \quad & \frac{1}{2}\langle c!0.\texttt{QRNG}; \emptyset; \rho \rangle \boxplus \frac{1}{2}\langle c!1.\texttt{QRNG}; \emptyset; \rho \rangle \\
\\
\xrightarrow{c!v} \quad & \frac{1}{2}\langle \texttt{QRNG}; \emptyset; \rho \rangle \boxplus \frac{1}{2}\langle \texttt{QRNG}; \emptyset; \rho \rangle
\end{aligned}$$

In this transition system we can observe the evolution of the density operator, and how the qubit lifetime is changed over transitions. We also see that after performing the measurement, two configurations are created, where one represents the measurement result '0' and the other '1'.

The developed syntax and semantics do not consider the application of noise to qubits. We intend to extend both cases in this direction in the near future.

5.3.5 *Timed Quantum Cloud*

In the example of the quantum cloud we aim at verifying if the coherence time is exceeded. Moreover, we want to verify which approach is more efficient - send all the data or split the data to be sent - and which guarantees the coherence of qubits. To have a fair comparison, the delay action will be placed before the measurement. This delay action can have the

same interpretation as in the previous case. First, we present the version where all data is set at once.

$$\texttt{User} \stackrel{def}{=} c!f(u).c!g(v).c?r.c?s.c!done.\texttt{nil}$$
$$\texttt{QC} \stackrel{def}{=} c?f(u).c?g(v).\texttt{Deutsch}$$
$$\texttt{Deutsch} \stackrel{def}{=} new(x,y,a,b).\texttt{X}[y,b].\texttt{H}[x,y,a,b].\texttt{Oracle}[x,y,a,b].\texttt{H}[x,a].\sigma(t)$$
$$.\texttt{M}[x;r].\texttt{M}[a;s].c!r.c!s.c?done.\texttt{nil}$$

We now need to verify if the no-cloning theorem is violated or not. We proceed as follows:

$$\frac{\vdots \quad \vdots \quad \Gamma^{Q}_{\texttt{User}} \cap \Gamma^{Q}_{\texttt{QC}} = \emptyset}{\Gamma_{\texttt{User}} \cup \Gamma_{\texttt{QC}} \vdash \texttt{User}||\texttt{QC}}$$

where $\Gamma_{\texttt{User}} = f(u) : C, g(v) : C$ and $\Gamma_{\texttt{QC}} = done : C$.

The first step introduced the condition $\Gamma^{Q}_{\texttt{User}} \cap \Gamma^{Q}_{\texttt{QC}} = \emptyset$, which one can see that is obeyed: $\Gamma^{Q}_{\texttt{User}} \cap \Gamma^{Q}_{\texttt{QC}} = \emptyset \cap \emptyset = \emptyset$.

Since it is hard to depict the total calculation of the validity of the process above (due to page size restrictions), we discern the `User` and `QC` cases separately. The calculation for `User` is as follows:

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f(u) : C, g(v) : C, r : C, s : C, done : C \vdash \texttt{nil}}}{f(u) : C, g(v) : C, r : C, s : C \vdash c?done.\texttt{nil}}}{f(u) : C, g(v) : C, r : C \vdash c?s.c?done.\texttt{nil}}}{f(u) : C, g(v) : C \vdash c?r.c?s.c?done.\texttt{nil}}}{f(u) : C, g(v) : C \vdash c!g(v).c?r.c?s.c?done.\texttt{nil}}}{\dfrac{f(u) : C, g(v) : C \vdash c!f(u).c!g(v).c?r.c?s.c?done.\texttt{nil} \quad \emptyset \subseteq \{f(u), g(v)\} : C}{f(u) : C, g(v) : C \vdash \texttt{User}}}$$

In QC we make $\Omega = f(u) : C, g(v) : C, done : C$ for simplification. The derivation for QC is as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y : Q, b : Q, f(u) : C, g(v) : C, r : C, s : C, done : C \vdash \texttt{nil}}}{y : Q, b : Q, \Omega, r : C, s : C \vdash c!done.\texttt{nil}}}{y : Q, b : Q, \Omega, r : C, s : C \vdash c!s.c!done.\texttt{nil}}}{y : Q, b : Q, \Omega, r : C, s : C \vdash c!r.c!s...}}{y : Q, a : Q, b : Q; \Omega, r : C \vdash \texttt{M}[a; s].c!r...}}{x : Q, y : Q, a : Q, b : Q, \Omega \vdash \texttt{M}[x; r].M[a; s]...}}{x : Q, y : Q, a : Q, b : Q, \Omega \vdash \sigma(t).\texttt{M}[x; r]... \quad \{x, a\} : Q \subseteq \{x, y, a, b\} : Q}}{x : Q, y : Q, a : Q, b : Q, \Omega \vdash \texttt{H}[x, a].\sigma(t)... \quad \{x, y, a, b\} : Q \subseteq \{x, y, a, b\} : Q}}{x : Q, y : Q, a : Q, b : Q, \Omega \vdash \texttt{Oracle}[x, y, a, b].\texttt{H}[x, a]... \quad \{x, y, a, b\} : Q \subseteq \{x, y, a, b\} : Q}}{x : Q, y : Q, a : Q, b : Q, \Omega \vdash \texttt{H}[x, y, a, b].\texttt{Oracle}[x, y, a, b]... \quad \{y, b\} : Q \subseteq \{x, y, a, b\} : Q}}{x : Q, y : Q, a : Q, b : Q, \Omega \vdash \texttt{X}[y, b].\texttt{H}[x, y, a, b]...}}{\Omega \vdash new(x, y, a, b).\texttt{X}[y, b]... \quad \emptyset \subseteq \{f(u), g(v), done\} : C}}{done : C, f(u) : C, g(v) : C \vdash \texttt{Deutsch}}}{done : C, f(u) : C \vdash c?g(v).\texttt{Deutsch}}}{done : C \vdash c?f(u).c?g(v).\texttt{Deutsch} \quad \emptyset \subseteq \{done\} : C}}{done : C \vdash \texttt{QC}}$$

Since the derivation was successful, we are able to calculate the execution of the process.

$$\begin{aligned}
& \langle \texttt{QCloud}; \emptyset; \rho \rangle \\
\xrightarrow{\tau} \quad & \langle c!g(v).c?r.c?s.c?done.\texttt{nil} \mid\mid c?g(v).\texttt{Deutsch}; \emptyset; \rho \rangle \\
\xrightarrow{\tau} \quad & \langle c?r.c?s.c?done.\texttt{nil} \mid\mid \texttt{Deutsch}; \emptyset; \rho \rangle \\
\xrightarrow{\tau} \quad & \langle c?r... \mid\mid \texttt{X}[y, b].\texttt{H}[x, y, a, b]...; 0_x, 0_y, 0_a, 0_b; [|00\rangle_{x,y}]] \otimes [|00\rangle_{a,b}] \otimes \rho \rangle \\
\xrightarrow{\tau} \quad & \langle c?r... \mid\mid \texttt{H}[x, y, a, b].\texttt{Oracle}[x, y, a, b]...; 0_x, 0_y, 0_a, 0_b; [|01\rangle_{x,y}] \otimes [|01\rangle_{a,b}] \otimes \rho \rangle \\
\xrightarrow{\tau} \quad & \langle c?r.c?s.c?done.\texttt{nil} \mid\mid \texttt{Oracle}[x, y, a, b].\texttt{H}[x, a]..; 0_x, 0_y, 0_a, 0_b; \rho_{H(x,y)} \otimes \rho_{H(a,b)} \otimes \rho \rangle \\
\xrightarrow{\tau} \quad & \langle c?r.c?s.c?done.\texttt{nil} \mid\mid \texttt{H}[x, a].\sigma(t)...; 0_x, 0_y, 0_a, 0_b; \rho_{Oracle(x,y)} \otimes \rho_{Oracle(a,b)} \otimes \rho \rangle \\
\xrightarrow{\tau} \quad & \langle c?r.c?s.c?done.\texttt{nil} \mid\mid \sigma(t).\texttt{M}[x; r]...; 0_x, 0_y, 0_a, 0_b; \rho_{H(x)} \otimes \rho_{H(a)} \otimes \rho' \rangle \\
\xrightarrow{\tau} \quad & \langle c?r.c?s.c?done.\texttt{nil} \mid\mid \texttt{M}[x; r].\texttt{M}[a; s]...; t_x, t_y, t_a, t_b; \rho_{H(x)} \otimes \rho_{H(a)} \otimes \rho' \rangle \\
\xrightarrow{\tau} \quad & \langle c?r.c?s.c?done.\texttt{nil} \mid\mid \texttt{M}[a; s].c!0...; t_y, t_a, t_b; \rho_{H(a)} \otimes \rho' \rangle \\
\xrightarrow{\tau} \quad & \langle c?r.c?s.c?done.\texttt{nil} \mid\mid c!0.c!1.c!done.\texttt{nil}; t_y, t_b; \rho' \rangle \\
\xrightarrow{\tau} \quad & \langle c?s.c?done.\texttt{nil} \mid\mid c!1.c!done.\texttt{nil}; t_y, t_b; \rho' \rangle \\
\xrightarrow{\tau} \quad & \langle c?done.\texttt{nil} \mid\mid c!done.\texttt{nil}; t_y, t_b; \rho' \rangle \\
\xrightarrow{\tau} \quad & \langle \texttt{nil} \mid\mid \texttt{nil}; t_y, t_b; \rho' \rangle
\end{aligned}$$

The density operators are the same as in the untimed example, and thus omitted here.

Consider now the second version of this example, in which the data is going to be divided. The example is encoded as follows:

$$\texttt{User} \stackrel{def}{=} c!f(u).c?r.c!g(v).c?s.\texttt{nil}$$

$$\texttt{QC} \stackrel{def}{=} c?f.\texttt{Deutsch}$$

$$\texttt{Deutsch} \stackrel{def}{=} new(x,y).\texttt{X}[y].\texttt{H}[x,y].\texttt{Oracle}[x,y].\texttt{H}[x].\sigma(t).\texttt{M}[x;r].c!r.\texttt{QC}$$

Let us now verify if the no-cloning theorem is obeyed.

$$\frac{\dfrac{\vdots}{\Gamma_{\texttt{User}} \vdash \texttt{User}} \qquad \dfrac{\vdots}{\Gamma_{\texttt{QC}} \vdash \texttt{QC}} \qquad \Gamma^{Q}_{\texttt{User}} \cap \Gamma^{Q}_{\texttt{QC}} = \emptyset}{\Gamma_{\texttt{User}} \cup \Gamma_{\texttt{QC}} \vdash \texttt{User} \,||\, \texttt{QC}}$$

where $\Gamma_{\texttt{User}} = f(u) : C, g(v) : C$ and $\Gamma_{\texttt{QC}} = \emptyset$. The condition $\Gamma^{Q}_{\texttt{User}} \cap \Gamma^{Q}_{\texttt{QC}} = \emptyset$ is satisfied, once $\Gamma^{Q}_{\texttt{User}} \cap \Gamma^{Q}_{\texttt{QC}} = \emptyset \cap \emptyset = \emptyset$.

Again, the total derivation needs to be split. So, we present first the part related to the User:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f(u) : C, g(v) : C, r : C, s : C \vdash \texttt{nil}}}{f(u) : C, g(v) : Cr : C \vdash c?s.\texttt{nil}}}{f(u) : C, g(v) : C, r : C \vdash c!g(v).c?s.\texttt{nil}}}{f(u) : C, g(v) : C \vdash c?r.c!g(v).c?s.\texttt{nil}}}{f(u) : C, g(v) : C \vdash c!f(u).c?r.c!g(v).c?s.\texttt{nil} \quad \emptyset \subseteq \{f(u), g(v)\} : C}}{f(u) : C, g(v) : C \vdash \texttt{User}}$$

And now the part related to QC:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y : Q, f : C, r : C \vdash \texttt{QC}}}{y : Q, f : C, r : C \vdash c!r.\texttt{QC}}}{x : Q, y : Q, f : C \vdash \texttt{M}[x;r].c!r.\texttt{QC}}}{x : Q, y : Q, f : C \vdash \sigma(t).\texttt{M}[x;r]... \quad \{x\} : Q \subseteq \{x,y\} : Q}}{x : Q, y : Q, f : C \vdash \texttt{H}[x].\sigma(t)... \quad \{x,y\} : Q \subseteq \{x,y\} : Q}}{x : Q, y : Q, f : C \vdash \texttt{Oracle}[x,y].\texttt{H}[x]... \quad \{x,y\} : Q \subseteq \{x,y\} : Q}}{x : Q, y : Q, f : C \vdash \texttt{H}[x,y].\texttt{Oracle}[x,y]... \quad \{y\} : Q \subseteq \{x,y\} : Q}}{x : Q, y : Q, f : C \vdash \texttt{X}[y].\texttt{H}[x,y]...}}{f : C \vdash new(x,y).\texttt{X}[y]... \quad \emptyset \subseteq \emptyset}}{f : C \vdash \texttt{Deutsch}}}{\emptyset \vdash c?f.\texttt{Deutsch}}}{\emptyset \vdash c?f.\texttt{Deutsch} \quad \emptyset \subseteq \emptyset}}{\emptyset \vdash \texttt{QC}}$$

Once more, the no-cloning theorem is respected. Therefore, the transition system can be calculated as follows:

$$
\begin{aligned}
& \langle \texttt{QCloud};\ \emptyset;\ \rho \rangle \\
\xrightarrow{\tau}\ & \langle c!f(u).c?r.c!g(v).c?s.\texttt{nil} \ ||\ c?f(u).\texttt{Deutsch};\ \emptyset;\ \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?r.c!g(v).c?s.\texttt{nil} \ ||\ \texttt{Deutsch};\ \emptyset;\ \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?r.c!g(v).c?s.\texttt{nil} \ ||\ \texttt{X}[y].\texttt{H}[x,y]...;\ 0_x, 0_y;\ [|00\rangle_{xy}] \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?r.c!g(v).c?s.\texttt{nil} \ ||\ \texttt{H}[x,y].\texttt{Oracle}[x,y]...;\ 0_x, 0_y;\ [|01\rangle_{xy}] \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?r.c!g(v).c?s.\texttt{nil} \ ||\ \texttt{Oracle}[x,y].\texttt{H}[x]...;\ 0_x, 0_y;\ \rho_{H(x,y)} \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?r.c!g(v).c?s.\texttt{nil} \ ||\ \texttt{H}[x].\sigma(t).\texttt{M}[x;r].c!s.\texttt{QC};\ 0_x, 0_y;\ \rho_{Oracle(x,y)} \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?r.c!g(v).c?s.\texttt{nil} \ ||\ \sigma(t).\texttt{M}[x;r].c!s.\texttt{QC};\ 0_x, 0_y;\ \rho_{H(x)} \otimes \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?r.c!g(v).c?s.\texttt{nil} \ ||\ \texttt{M}[x;r].c!s.\texttt{QC};\ t_x, t_y;\ \rho_{H(x)} \otimes \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?r.c!g(v).c?s.\texttt{nil} \ ||\ c!0.\texttt{QC};\ t_y;\ \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c!g(v).c?s.\texttt{nil} \ ||\ c?f(u).\texttt{Deutsch};\ t;\ t_y;\ \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?s.\texttt{nil} \ ||\ new(x,a).\texttt{X}[a]...;\ t_y;\ \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?r.c!g(v).c?s.\texttt{nil} \ ||\ \texttt{X}[a].\texttt{H}[x,a]...;\ 0_x, 0_a, t_y;\ [|00\rangle_{xa}] \otimes \rho_a \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?s.\texttt{nil} \ ||\ \texttt{H}[x,a].\texttt{Oracle}[x,a]...;\ 0_x, 0_a, t_y;\ [|01\rangle_{xa}] \otimes \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?s.\texttt{nil} \ ||\ \texttt{Oracle}[x,a].\texttt{H}[x]...;\ 0_x, 0_a, t_y;\ \rho_{H(x,a)} \otimes \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?s.\texttt{nil} \ ||\ \texttt{H}[x].\sigma(t).\texttt{M}[x;r].c!s.\texttt{QC};0_x, 0_a, t_y;\ \rho_{Oracle(x,a)} \otimes \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?s.\texttt{nil} \ ||\ \sigma(t).\texttt{M}[x;r].c!s.\texttt{QC};\ 0_x, 0_a, t_y;\ \rho_{H(x)} \otimes \rho_a \otimes \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?s.\texttt{nil} \ ||\ \texttt{M}[x;r].c!s.\texttt{QC};\ t_x, t_a, 2t_y;\ \rho_{H(x)} \otimes \rho_a \otimes \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle c?s.\texttt{nil} \ ||\ c!1.\texttt{QC};\ t_a, 2t_y;\ \rho_a \otimes \rho_y \otimes \rho \rangle \\
\xrightarrow{\tau}\ & \langle \texttt{nil} \ ||\ \texttt{QC};\ t_a, 2t_y;\ \rho_a \otimes \rho_y \otimes \rho\ \rangle
\end{aligned}
$$

In both cases, through the typing and transition systems we can see that garbage accumulates along the unfolding of the recursive process. This is caused by the qubits y and a that are not erased from the density operator, since they are not measured. From that, we can conclude that a trash action would indeed be useful.

Now that both approaches were presented enriched with temporal dimension, we will compare them with respect to two aspects: data and qubit decoherence. Let us begin with data. When data is all sent, the quantum computer only needs to run the Deutsch algorithm once. We need to keep in mind that the quantum computer puts all the states into a superposition state, and performs all the calculations needed only once. When data is split, the quantum computer runs the Deutsch algorithm two times. Therefore, it seems reasonable to conclude that sending all the data is better, because it exploits the parallelism naturally inherent in quantum computing. Regarding quantum decoherence, the qubits age is included in t_q, which increases according to delay actions. More concretely, the value t presented in the delay action represents the time elapsed since the creation of the first qubit until its measurement. However we cannot predict the amount of time elapsed so, we need

to make this experience in a quantum computer. Therefore, the only conclusion that can be made, is that the qubits lifetime is dependent on the delay t and, the greater the delay t the greater the chance of the qubits becoming decoherent.

5.4 TQCCS VS QCCS VS CQP

In this section we make a comparison between our theory, *qCCS*, Feng et al. (2012), and *CQP*, Gay and Nagarajan (2006).

TqCCS is a timed-quantum process algebra based on *qCCS*, but whose typing system is based on *CQP*. We framed our work in two quantum process algebras in order to merge their best features. First, consider *qCCS* and *CQP*.

qCCS	CQP
Recursive processes	
Labelled Transition System	
	Transition system
	Typing system
	Qubit allocation

Table 13: *qCCS* vs *CQP*

In Table 13, one can see that *qCCS* exhibits recursive processes and transitional dynamics. But, it does not allow the expansion of Hilbert space, meaning that the creation of new qubits is impossible. Because of that, the Hilbert space of a configuration is always pre-defined. In the same table, one can see that *CQP* does not have recursion, but a simple transition system, based on reductions, and a typing system. Moreover, it allows the creation of new qubits, making the Hilbert space to grow along the computation process.

By analysing the advantages and disadvantages of the previous process algebras, we took the advantages of *qCCS* and *CQP* and merged them to form our theory. With that we were able to construct a quantum process algebra that allows recursion on processes, the creation of new qubits and has both typing and transition systems. Having said that, we can now extend Table 13 with a column for *TqCCS*.

	qCCS	*CQP*	*TqCCS*
Recursive processes	×		×
Labelled Transition system	×		×
Typing system		×	×
Qubit Allocation		×	×

Table 14: *qCCS* vs *CQP* vs *TqCCS*

The Table 14 shows that *TqCCS* merges the advantages of *qCCS* and those of *CQP*. Moreover, we did not include the transition system of *CQP* in this table, because we have followed the *LTS* present in *qCCS*.

6

CONCLUSIONS AND FUTURE WORK

6.1 CONCLUSION

The goal of this dissertation was to develop a quantum process algebra with a temporal notion and a typing system. Through the first feature we intend to verify the lifetime of qubits, and trigger actions. With the second feature we aim at providing a better control of the no-cloning theorem, as well as deriving and analysing logical properties such as weakening.

We started our work by studying the concept of concurrency. We concluded that concurrency is very important, because it is inherently present in a myriad of systems and can be used to improve system's performance. Next, we studied different process algebras, which form a standard class of tools for modelling and analysing concurrency. By studying Baeten et al. (2010), we saw that a process can be seen as an automaton. Among other things, this allows to use automata theory – and in particular trace equivalence and bisimilarity – to compare the behaviour of two processes.

Later on, the introduction of a temporal dimension in process algebra was studied. We saw that in designing a timed process algebra there are multiples choices to be made: *e.g.* time domain, measurement of time, execution sequence, introduction of time at the syntactic level, and model properties. Then, we presented an extension of *CCS* with timed-variables and delay actions, *Timed CCS*, Yi (1991)

Next, an introduction to the basics of quantum mechanics was made, so that the reader could recall some concepts that are crucial in developing a quantum process algebra - the no-cloning theorem, density operator, unitary operator, etc. Then, a quantum version of *CCS*, *qCCS*, Ying et al. (2009) was outlined. In the syntax, the set of quantum free variables was used to prevent the no-cloning theorem. In the semantics, the concept of configuration was introduced, as composed of a process and a density operator. Here, the creation of new qubits is not allowed, which is a limitation, but the definition of recursive processes, making use of process constants, is possible and a great advantage. Having in mind the typing system of *CQP*, we introduced a typing system for *qCCS*. We proved the weakening

property. Moreover, we proved that exists an equivalence between *qCCS* processes and the ones derived by our typing system.

After, we extended *qCCS* with classical variables, Feng et al. (2012), and developed two examples - Quantum Random Number Generator and Quantum Cloud - with the intent to verify if the extended *qCCS* was able to codify them. At a first attempt, the result was bad, because it was impossible to have a recursive definition of the process `Env` for both examples. Therefore, inspired by the creation of qubits in *CQP*, we introduced an action $new(x)$ in the syntax of *qCCS*, and changed the measurement rule of the *qCCS* semantics: the application of the partial trace to the density operator in the order of the measured qubits. With this, we were able to encode both examples as desired.

After developing a quantum process algebra with a typing system, we introduced a temporal dimension, more specifically, a delay action. We called the resulting process algebra *TqCCS*. In our process algebra, information related with the lifetime of qubits is available in the configurations. We also considered timed versions of the aforementioned examples, in order to study and discuss how our process algebra handles them.

With the development of the typing system, we saw that not all the functionalities of linear logic were necessary.

As already mentioned, the introduction of time in a quantum process algebra is useful to control the coherence time of qubits, which is critical for the current NISQ (Noisy Intermediate-Scale Quantum) quantum computing. Furthermore, the presence of a typing system is useful to verify the no-cloning theorem and meta-logical properties, such as the result for weakening.

6.2 Future Work

We intend to explore the following research lines in the near future.

1. Implementation of super-operators that simulate noisy qubits

   When the lifetime of qubits is exceeded, they become susceptible to noise. We wish to extend our timed-quantum process algebra with super-operators to simulate this scenario. Specifically, we aim to introduce super-operators that make qubits noisy.

2. Expansion Law

   One important result in the theory of *CCS*, Milner (1989), is the expansion law. It would thus be interesting to develop an analogous expansion law for our timed-quantum process algebra.

3. Quantum Random Walks

Since the Hilbert space can be expanded, via the *new* action, and contracted, via measurements, we believe that it is possible to encode quantum random walks with our quantum process algebra. This would be important because quantum random walks have many applications, Shenvi et al. (2003); Childs et al. (2003); Ambainis (2003).

4. Trash

   In the Timed Quantum Cloud example we saw that there exists qubits that are not measured. These qubits are considered *garbage* once they do not have any utility. Therefore, the necessity of an erasing action.

5. Experimental results for the Timed Quantum Cloud example

   The conclusions drawn from the Timed Quantum Cloud example were purely based on our calculations and not on experimental evidence. Therefore, it would be interesting to have experimental results confirming or refuting our conclusions.

6. Integrate qubits lifetime in the typing system

   In the current state, the typing system of *TqCCS* allows us to verify if the no-cloning theorem is obeyed before making the transition system. With the introduction of qubits lifetime into the typing system, we would also be able to know if the coherence time of qubits was, in a certain point of the process, exceeded.

7. Model properties

   To go from "we wish that our theory respects" to "our theory respects" we need to prove the model properties described in the Subsection 5.3.2 for *TqCCS*.

BIBLIOGRAPHY

Ibm quantum computing. https://www.ibm.com/quantum-computing/. Accessed: 2019-11-1.

Id quantique quantum random number generator. https://www.idquantique.com/random-number-generation/products/quantis-random-number-generator/, 2016. Accessed: 2019-8-25.

Luca Aceto, Kim G Larsen, and Anna Ingólfsdóttir. An introduction to milner's ccs. *Course Notes for Semantics and Verification. Constantly under revision. The most recent version is available at the URL http://www. cs. auc. dk/ luca/SV/Intro21ccs. pdf, BRICS, Department of Computer Science, Aalborg, Denmark*, 2005.

Andris Ambainis. Quantum walks and their algorithmic applications. *International Journal of Quantum Information*, 1(04):507–518, 2003.

J.C.M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2):131 – 146, 2005. ISSN 0304-3975. doi: https://doi.org/10.1016/j.tcs.2004.07.036. URL http://www.sciencedirect.com/science/article/pii/S0304397505000307. Process Algebra.

Jos CM Baeten and Jan A Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

Josephus CM Baeten, Twan Basten, Twan Basten, and MA Reniers. *Process algebra: equational theories of communicating processes*, volume 50. Cambridge university press, 2010.

J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1):109 – 137, 1984. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(84)80025-X. URL http://www.sciencedirect.com/science/article/pii/S001999588480025X.

Patrick Blackburn, Maarten De Rijke, and Yde Venema. *Modal Logic: Graph. Darst*, volume 53. Cambridge University Press, 2002.

Olav Bunte, Jan Friso Groote, Jeroen J. A. Keiren, Maurice Laveaux, Thomas Neele, Erik P. de Vink, Wieger Wesselink, Anton Wijs, and Tim A. C. Willemse. The mcrl2 toolset for analysing concurrent systems. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 21–39, Cham, 2019. Springer International Publishing.

Andrew M Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 59–68. ACM, 2003.

David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439 (1907):553–558, 1992.

E. W. Dijkstra. Solution of a problem in concurrent programming control. *Commun. ACM*, 8(9):569–, September 1965. ISSN 0001-0782. doi: 10.1145/365559.365617. URL http://doi.acm.org/10.1145/365559.365617.

Paul Adrien Maurice Dirac. *The principles of quantum mechanics*. Number 27. Oxford university press, 1981.

Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying. Probabilistic bisimulations for quantum processes. *Information and Computation*, 205(11):1608–1639, 2007.

Yuan Feng, Runyao Duan, and Mingsheng Ying. Bisimulation for quantum processes. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 34(4):17, 2012.

Richard Feynman. Simulating physics with computers, 1982.

Simon J Gay and Rajagopal Nagarajan. Communicating quantum processes. In *ACM SIGPLAN Notices*, volume 40, pages 145–157. ACM, 2005.

Simon J Gay and Rajagopal Nagarajan. Types and typechecking for communicating quantum processes. *Mathematical Structures in Computer Science*, 16(3):375–406, 2006.

Jan Friso Groote and Mohammad Reza Mousavi. *Modeling and Analysis of Communicating Systems*. The MIT Press, 2014. ISBN 0262027712, 9780262027717.

Lov K Grover. A fast quantum mechanical algorithm for database search. *arXiv preprint quant-ph/9605043*, 1996.

C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, August 1978. ISSN 0001-0782. doi: 10.1145/359576.359585. URL http://doi.acm.org/10.1145/359576.359585.

Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN 0321455363.

J Kempe. Quantum random walks: An introductory overview. *Contemporary Physics*, 44 (4):307–327, Jul 2003. ISSN 1366-5812. doi: 10.1080/00107151031000110776. URL http://dx.doi.org/10.1080/00107151031000110776.

Emmanuel Knill. Conventions for quantum pseudocode. Technical report, Los Alamos National Lab., NM (United States), 1996.

Marie Lalire and Philippe Jorrand. A process algebraic approach to concurrent and distributed quantum computation: operational semantics. *arXiv preprint quant-ph/0407005*, 2004.

John McCarthy. A basis for a mathematical theory of computation1). In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 33 – 70. Elsevier, 1963. doi: https://doi.org/10.1016/S0049-237X(08)72018-4. URL http://www.sciencedirect.com/science/article/pii/S0049237X08720184.

Robin Milner. *Communication and concurrency*, volume 84. Prentice hall New York etc., 1989.

Xavier Nicollin and Joseph Sifakis. An overview and synthesis on timed process algebras. In Kim G. Larsen and Arne Skou, editors, *Computer Aided Verification*, pages 376–398, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg. ISBN 978-3-540-46763-2.

Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge University Press, Cambridge, 2000.

Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962.

Gordon D Plotkin. A structural approach to operational semantics. 1981.

AW Roscoe and GM Reed. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58, 1988.

Dana Scott and C Strachey. Towards a mathematical semantics for computer languages. *Proceedings of the Symposium on Computers and Automata*, 21, 01 1971.

Neil Shenvi, Julia Kempe, and K Birgitta Whaley. Quantum random-walk search algorithm. *Physical Review A*, 67(5):052307, 2003.

Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

John von Neumann and RT Beyer. Mathematical foundations of quantum mechanics. 1955.

Wang Yi. Real-time behaviour of asynchronous agents. In *CONCUR '90, Theories of Concurrency: Unification and Extension, Amsterdam, The Netherlands, August 27-30, 1990, Proceedings*, pages 502–520, 1990. doi: 10.1007/BFb0039080. URL https://doi.org/10.1007/BFb0039080.

Wang Yi. CCS + time = an interleaving model for real time systems. In *Automata, Languages and Programming, 18th International Colloquium, ICALP91, Madrid, Spain, July 8-12, 1991, Proceedings*, pages 217–228, 1991. doi: 10.1007/3-540-54233-7_136. URL https://doi.org/10.1007/3-540-54233-7_136.

Mingsheng Ying, Yuan Feng, Runyao Duan, and Zhengfeng Ji. An algebra of quantum processes. *ACM Transactions on Computational Logic (TOCL)*, 10(3):19, 2009.