Quantum Constraint Programming

Alessandra Di Pierro and Herbert Wiklicky

Dipartimento di Informatica, Universitá di Pisa, Italy
Department of Computing, Imperial College, London, UK

Abstract

Quantum computers are hypothetical machines which can perform many calculations simultaneously based on quantum-mechanical principles that allows a single bit to coexist in many states at once. This enormous potential of quantum computing has attracted substantial interest, especially during the last decade, and initiated a whole new field of research. As a contribution to this research we address the problem of the design of high level languages for programming quantum computers, and the definition of an appropriate formal semantics for such languages. To this purpose we consider the Constraint Programming paradigm and we show how computations in this paradigm can be seen as physical processes obeying the laws of quantum mechanics.

1 Introduction

Quantum Computing is currently one of the hottest topics in computer science, physics and engineering. One of the reason for such an interest is the dramatic miniaturisation in computer technology over the past 40 years, which will inevitably force (if the trend continues) the use of quantum physics to describe the elementary operations of a computer. The by now already flourishing research in quantum computing has arisen in anticipation of this inevitable step.

If technological issues have started the studies of the new computational model, the discovery of its potential has made the interest continuously increasing, especially after the presentation of Shor's quantum factorisation algorithm [30], and Grover's search algorithm [14]. These algorithms heavily exploits quantum mechanical phenomena. Thanks to quantum effects like superposition and entanglement, a function can be evaluated on a quantum machine so as all the outputs are computed in the time taken to evaluate just one output classically. Although a measurement of the final superposed state can yield only one output, it is still possible to obtain certain joint properties of all the outputs. As a consequence of this *quantum parallelism*, Shor and Grover's algorithms are faster than any known classical algorithm designed for solving the same problems.

A mathematical model for quantum computation cannot be developed by ignoring the connections between physics and computation. Contrary to the classical theory of computation pure mathematics is not sufficient and quantum mechanics becomes an essential ingredient of a model for a quantum computer. Indeed, first stated by Benioff [1] and Feynman [11], this idea was finally formalised by Deutsch [7], whose Quantum Turing Machine (QTM) makes full use of superposition and quantum mechanical interactions to accomplish its basic operations. Some of the foundational work goes back to the early 1970s, like Charles Bennett's work on *reversible computation* [2] and Samuel Eilenberg's notion of a general machine model using linear operators so-called *linear machines* [10, 34].

Quantum Turing Machines represent a basic model for quantum computation like Turing Machines for classical computation. The classical theory of computation provides however additional high-level descriptions of computation, by means of programming languages. Research in languages and their semantics is still at a very early stage in quantum computing where algorithms are still described by means of pseudo-code or similar low-level description like "quantum circuits". In fact, as the latter represents a universal language for describing quantum computations, implementing an algorithm in quantum computing is at this stage like programming a Turing Machine. Although this is in principle always possible it is quite a tedious task.

In this paper we propose the use of Constraint Programming as a high level language for implementing quantum algorithms. In particular, we consider a simple declarative constraint language and we provide it with an operational semantics which describes the computation of a program in the language as the evolution of a (closed) quantum system, that is a system subject to the laws of quantum mechanics.

The central idea for such a description is the notion of a *quantum transition system*. This is defined via a transition relation in the style of Plotkin's "Structured Operational Semantics" [25], where transitions represent unitary transformations of states, and the states of the system are described by vectors in a Hilbert space. The result of a computation in such a transition system has to be defined according to the quantum mechanical notion of *physical observables*. Roughly, in a quantum system one has to distinguish between the final state in the evolution of the system (computation in the transition system), and its measurement, which allows us to (partially) observe the information encoded in the final state. This observation destroys the current state.

We will model our notion of *computational observables* in terms of the (physically un-observed but complete) final state instead of the (physically observed but incomplete) measurements. This corresponds to modelling a computation as a black box, leaving it open which particular method the user wishes to adopt in order to observe the outputs.

2 Quantum Mechanics

The history of quantum mechanics started nearly exactly 100 years ago, when on the 14th December 1900, Max Planck coined the term in a talk on so-called black body radiation [31]. The theory was developed fully by Erwin Schrödinger (based on so-called wave function) and Werner Heisenberg (based on infinite matrices) in the mid 1920s, and after that combined (and shown to be equivalent) in particular by John von Neumann and Garrett Birkhoff [4, 33].

2.1 Physical Concepts

We recall a few epistemological concepts at the base of the postulates of the standard formulation of quantum mechanics:

1. The *physical state* of a system encodes all information needed to describe the system uniquely (within a certain physical theory).

2. *Physical observables* are certain entities or properties – like mass, electrical charge, velocity, etc – which we associate with a physical system.

3. The *measurement* of an observable for a given system relates its state to a measurement result corresponding to the observable in question.

For quantum mechanics it is essential to keep the state and the observables as separate notions. The measurement process is the conceptual bridge between the "true" state of a system and the "classical" information we can obtain about the system. The axiomatic formulation of quantum mechanics using a Hilbert space based mathematical model relates these concepts. We refer to [16] for such an axiomatic formulation as well as for any mathematical notions we will use throughout the paper.

2.2 Quantum Mechanical Postulates

The three basic axioms or postulates of quantum physics are the following:

1. The information describing the *state* of an (isolated) quantum mechanical system is represented mathematically by a normalised vector in a complex Hilbert space $\mathcal{H}$.

2. A physical *observable* is represented mathematically by a self-adjoint operator $\mathbf{A}$ acting on $\mathcal{H}$.

3. The *expected result* of measuring an observable $\mathbf{A}$ of a system in state $\vec{x} \in \mathcal{H}$ is given by: $\langle A \rangle_x = \langle \vec{x}, \mathbf{A}\vec{x} \rangle = \langle \vec{x} | \mathbf{A}\vec{x} \rangle$

We can refine the postulate 3 above in order to give a more detailed description of what can possibly happen when we measure a certain observable (applying the so-called *spectral theorem* for self-adjoint operators):

3' The only *possible* results obtained by measuring the physical observable $\mathbf{A}$ are the eigenvalues λ_i of $\mathbf{A}$.

3" The *probability* of measuring λ_n for a system in state $\vec{x}$ is given by the inner product: $Pr(A = \lambda_n, \vec{x}) = \langle \vec{x} | \mathbf{P}_n \vec{x} \rangle = \langle \vec{x} | \mathbf{P}_n | \vec{x} \rangle$ with $\mathbf{P}_n$ the (orthogonal) projection corresponding to λ_n.

The result of measuring an observable $\mathbf{A}$ for a system in state $\vec{x}$, produces two effects: first we obtain one of the possible measurement results λ_n, i.e. one of the eigen-values of $\mathbf{A}$, and secondly the system is "reduced", i.e. the new state of the system is now $\mathbf{P}_n\vec{x}$. Each of such pairs of effects takes place with a probability given by $\langle\vec{x}|\mathbf{P}_n\vec{x}\rangle$. Measurements are therefore *idempotent* and *irreversible* (unless $\mathbf{P} = \mathbf{Id}$).

One of the basic problems for physics is to determine the *time evolution* of a system, e.g. how the state of a system changes according to the physical constraints. The *dynamics* of a (closed) quantum system is described by a differential equation, the so-called *Schrödinger Equation*:

$$i\frac{d\vec{x}}{dt} = \mathbf{H}\vec{x},$$

where the self-adjoint operator $\mathbf{H}$ represents the physical observable "energy". It is usually referred to as the *Hamiltonian* operator $\mathbf{H}$ of the system. The *solution* to this equation is given by the operator $\mathbf{U}_t = \exp(it\mathbf{H})$.

Theorem 1 *For any self-adjoint operator* $\mathbf{A}$ *the operator*

$$\exp(i\mathbf{A}) = e^{i\mathbf{A}} = \sum_{n=0}^{\infty} \frac{(i\mathbf{A})^n}{n!}$$

is a unitary operator.

Therefore, the time evolution of a closed, non-measured quantum system is given by a unitary operator $\mathbf{U}$. The true difficulty for a physicist is to calculate $\mathbf{U}$ for various concrete operators $\mathbf{H}$, while conceptually the time evolution of the system is particularly simple. It is *deterministic* and *reversible*, as $\mathbf{U}^{-1} = \mathbf{U}^*$.

For further discussions of the physical background we mention just a few of the hundreds of monographs on the subject, like Feynmann's lectures [12] for an hands-on approach, or [20] and [16] for a presentation of the mathematics of the standard formulation. There exists also an elegant, abstract C*-algebraic formulation of quantum mechanics, e.g. [32] or [5].

3 Quantum Constraint Programming

Constraint Programming (CP,[21]) is a programming paradigm built upon constraints and constraint solving, in which computational problems are modelled as constraint problems which can be solved by a variety of techniques. The most popular approach to CP is Constraint Logic Programming (CLP, [17]) which was introduced to build user-defined constraints and for modelling constraint problems. Another important approach is Concurrent Constraint Programming (CCP,[29]) which extends CLP by allowing the parallel execution of concurrent processes, and introduces concurrency through an asynchronous communication between processes by way of constraint entailment. Both programming paradigms are provided with a

$$A \quad ::= \quad \mathbf{stop} \;\Big|\; \mathbf{tell}(c) \;\Big|\; \square_{i=1}^{n}\, p_i : A_i \;\Big|\; \|_{i=i}^{n}\, A_i \;\Big|\; proc$$

Table 1: Syntax of PCLP

clear and elegant mathematical semantics which represents a major advantage of these languages.

We show that Constraint Programming can be used as a high level specification of quantum computation, by presenting an implementation of a simple constraint programming language in a quantum setting. We do so by defining for this language an operational semantics which reflects the mathematical model used in quantum mechanics to represent the state and the evolution of a quantum system.

3.1 The Language

The constraint programming language we consider in this paper is a simplified version of Probabilistic Concurrent Constraint Programming (PCCP), which was introduced by the authors in [8, 9] on the base of Concurrent Constraint Programming (CCP) [29].

For the sake of simplicity, we assume here a minimal version of PCCP where both the hiding operator and the prefixing guard, **ask**, are removed. The latter simplification allows us to concentrate on a synchronisation-free sublanguage which corresponds roughly to a probabilistic version of Constraint Logic Programming (CLP). Thus, we we will refer to it as PCLP. Procedure calls are simplified by considering parameterless predicates and abstracting from the parameter passing operation. The syntax of a PCLP agent A is given in Table 1. In $\mathbf{tell}(c)$, c is a *finite* constraints in a constraint system $\mathcal{C}$ modelled as a complete lattice with respect to the entailment between constraints [29, 6]. We will indicate by $\sqcup$ the least upper bound in $\mathcal{C}$.

The basic PCLP agent is the primitive for communication **tell**, which is used to add information to the common store (see [6] for a description of the CCP execution model). The **stop** agent is introduced for convenience to indicate successful termination. Nondeterminism is introduced in the form of a probabilistic choice, $\square_{i=1}^{n}\, p_i : A_i$, where the p_i's represent "classical" probabilities and express the likelihood that agent A_i is chosen for execution [9]. The parallel construct, $\|_{i=i}^{n}\, A_i$, expresses the simultaneous execution of its components. Procedures are supposed to be defined in a declarations section, declarations being of the form $proc \leftarrow A$, where A is an agent.

3.2 The Quantum Computational Model

We model the execution of a PCLP program as the evolution of a quantum mechanical system. The nature of quantum mechanical processes restricts the way we can represent *data*, and what we can do to manipulate/change it, i.e. in order to *compute*. Data must be represented via the state of a quantum system. In quantum physics, the state of a quantum system is described by a vector in a Hilbert space. Moreover, any computational step may only be either a unitary transformation (i.e. a quantum evolution of the system or data in question) or a projection (i.e. a classical measurement of the system). While the first type of computational step is deterministic and reversible, the second one is probabilistic (i.e. we can only specify a set of possible projections which will be performed, depending on the state of the system) and irreversible (the measurement "collapses" or "reduces" the state of the system).

In computational terms, the unitarity condition corresponds to model the execution of a program as a *reversible* process: Given the outcome of a computation and the program, it is always possible to reconstruct the input uniquely.

In PCLP, **tell** is the only effective computational step. Thus, in order to formally define a quantum model for PCLP, we need to represent such basic agent as a unitary operator on an appropriate Hilbert space construction on the constraint system. This construction will be given in Section 3.4.

3.3 Reversible tell

The first step in the "quantisation" of the **tell** operation is to make it classically *reversible*, so that from the result of its application we can reconstruct the initial state. More precisely, given the resulting store d' after the application of $\mathbf{tell}(c)$, we have to be able to reconstruct the initial store d such that $\langle \mathbf{tell}(c), d\rangle \rightarrow \langle \mathbf{stop}, d'\rangle$.

In the classical case, where we represent the store by a single constraint, we cannot "undo" the effect of $\mathbf{tell}(c)$ as the initial store d is not uniquely determined by d'. All we know is that

$$\langle \mathbf{tell}(c), d\rangle \rightarrow \langle \mathbf{stop}, d' = d \sqcup c\rangle\,,$$

but we cannot conclude from $d' = d \sqcup c$ that the initial store was d. For example, d could be *true* or c and in both cases we would get $d' = c$.

To overcome this problem we will represent the store by a *multi-set* of constraints, e.g. $s = \{\!|c, d, e, e, \dots|\!\}$. Formally, we will represent a multi-set of constraints by a mapping $s : \mathcal{C} \mapsto \mathbb{N}$, such that $s(c)$ is the multiplicity of constraint $c \in \mathcal{C}$. The effect of adding a constraint is then simply to increase the multiplicity of that constraint:

$$\langle \mathbf{tell}(c), s\rangle \longrightarrow \langle \mathbf{stop}, s' = s \cup \{\!|c|\!\}\rangle$$

(where '$\cup$' denotes the multi-set union). This allows us to uniquely reconstruct the initial state: If the store s' is the one resulting after the execution of $\mathbf{tell}(c)$, then we can conclude that the initial state is $s = s' \setminus \{\!|c|\!\}$ (where '$\setminus$' denotes the multi-set difference).

Proposition 1 *The operation* $\mathbf{tell}(c)$ *applied to a multi-set store* s *is reversible.*

The usual representation of the store as a single constraint d can be reconstructed from the multi-set representation s simply as the least upper bound of the constraints in s, i.e. $d = \bigsqcup s$. Obviously, it is impossible to construct a unique s for a given d. Note that the store *true*, which usually indicates the initial store, could be represented in this case by an empty multi-set $\{\!| \ |\!\}$ or by $\{\!| true |\!\}$.

The next step is to define **tell**(c) as a unitary operator on the constraint system. This represents the set of all computational states which, according to the postulates of quantum mechanics (see Section 2), must be normalised vectors in a Hilbert space. We refer to the Appendix for the formal definition of a Hilbert space.

3.4 Quantum Constraint System

We will assume that the constraint systems $\mathcal{C}$ underlying PCLP are finite. A quantum representation of $\mathcal{C}$ by means of a Hilbert space must accommodate the multi-set based representation of the stores introduced in Section 3.3 for the definition of **tell** as a reversible operator. To this purpose we consider the complex free vector space $\mathcal{V}(\mathcal{C})$ over the constraint system, formally defined by:

$$\mathcal{V}(\mathcal{C}) = \{\sum_{c \in \mathcal{C}} x_c \cdot c \mid x_c \in \mathbb{C}\},$$

where the coefficients x_c encode the multiplicity of constraint c. Since $\mathcal{C}$ is finite, $\mathcal{V}(\mathcal{C})$ is a Hilbert space. In order to guarantee the linearity of the operator **tell** , we will represent the multiplicities as exponentials $\exp(ni)$, where n is the multiplicity and i is the imaginary number ($i^2 = -1$).

With this representation, for each multi-set of constraints s there is a unique vector $\vec{s} \in \mathcal{V}(\mathcal{C})$ which represents s.

Definition 1 *Let $s : \mathcal{C} \mapsto \mathbb{N}$ be a multi-set of constraints. The vector in $\mathcal{V}(\mathcal{C})$ representing s is defined as*

$$\frac{1}{\sqrt{|\mathcal{C}|}} \sum_{c \in \mathcal{C}} \exp(s(c)i) \cdot c,$$

where $|\mathcal{C}|$ is the cardinality of $\mathcal{C}$.

The factor $\frac{1}{\sqrt{|\mathcal{C}|}}$ is a normalisation factor which guarantees that every multi-set is represented by a normalised vector in $\mathcal{V}(\mathcal{C})$.

Example 1 *The multi-set $s = \{\!| c, c, e, f |\!\}$ will be represented by the vector $\vec{x} = \frac{1}{\sqrt{4}}(\exp(2i)\vec{c} + \exp(0)\vec{d} + \exp(i)\vec{e} + \exp(i)\vec{f}) = \frac{1}{2}(\exp(2i), 1, \exp(i), \exp(i))$ in the free complex vector space, $\mathbb{C}^4$, over $\mathcal{C} = \{c, d, e, f\}$.*

The basic identity: $\exp(m+n) = \exp(n)\exp(m)$ will now guarantee the linearity of the operator representing **tell**(c).

As we already mentioned, the correspondence between constraints and multi-sets of constraints is not one-to-one. It is nevertheless possible to obtain a bijection when equivalence classes of multi-sets are considered. More precisely, we can define an equivalence on multi-sets as follows:

$$s \sim t \text{ iff } \bigsqcup_{c \in s} c = \bigsqcup_{c \in t} c.$$

We can now associate to each constraint $c \in \mathcal{C}$ the equivalence class of the multiset $\{\!|c|\!\}$. As a consequence, the representation of a constraint $c \in \mathcal{C}$ as a vector in $\mathcal{V}(\mathcal{C})$ is unique modulo the natural extension of the equivalence $\sim$ to vectors in $\mathcal{V}(\mathcal{C})$.

3.5 Unitary tell

A unitary representation of the **tell** agent can be obtained by defining it as the diagonal matrix:

$$\mathbf{U}_c = \sum_{d \neq c} \mathbf{E}_{dd} + \exp(i)\mathbf{E}_{cc},$$

where the unit matrices $\mathbf{E}_{cd}$ are matrices with zero entries except a 1 in row c and column d.

Example 2 *Assume a constraint system $\mathcal{C}$ of cardinality n, with the enumeration $\{a, b, c, d, \ldots\}$ Then the* $\mathbf{tell}(c)$ *operator corresponds to the matrix*

$$\mathbf{U}_c = \begin{pmatrix} e^0 & 0 & 0 & 0 & \ldots \\ 0 & e^0 & 0 & 0 & \ldots \\ 0 & 0 & e^i & 0 & \ldots \\ 0 & 0 & 0 & e^0 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots \\ 0 & 1 & 0 & 0 & \ldots \\ 0 & 0 & e^i & 0 & \ldots \\ 0 & 0 & 0 & 1 & \ldots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

The effect of the execution of $\mathbf{tell}(c)$ *in the store $\{\!|c, d|\!\}$ is then the vector:*

$$\mathbf{U}_c \cdot \frac{1}{\sqrt{n}} \begin{pmatrix} 1 \\ 1 \\ e^i \\ e^i \\ \vdots \end{pmatrix} = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 \\ 1 \\ e^{2i} \\ e^i \\ \vdots \end{pmatrix},$$

that is the store $\{\!|c, c, d|\!\}$.

The adjoint operator of $\mathbf{U}_c$ is given by:

$$\mathbf{U}_c^* = \sum_{d \neq c} \mathbf{E}_{dd} + \exp(-i)\mathbf{E}_{cc}$$

Since $\mathbf{U}_c\mathbf{U}_c^* = \mathbf{Id}$ holds, $\mathbf{U}_c$ is a unitary operator.

A geometric intuition behind the representation of **tell** by the operator $\mathbf{U}_c$ is obtained by recalling that complex numbers of the form $\exp(ix)$, with $x \in \mathbb{R}$, correspond to points on the unit circle in the complex plain, or complex numbers with absolute value $|\exp(ix)| = 1$ (cf. Figure 1). Therefore, the multiplication of any complex number z with $\exp(ix)$ corresponds to a rotation of z on the imaginary unit circle by an angle of x radians.

With our representation of the multiplicity, the effect of $\mathbf{tell}(c)$ is a rotation by 1 radiant of the multiplicity of c.

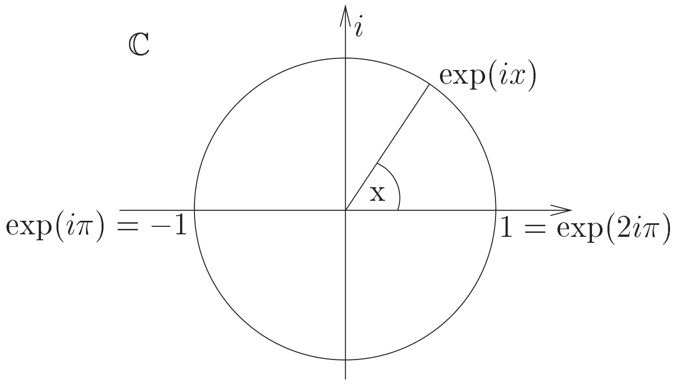

Figure 1: The Unit Circle in $\mathbb{C}$.

Note that this representation of the multiplicity via rotations is unique, periodicity notwithstanding: It can never happen that $\exp(in) = \exp(im)$ for any n, m such that $n \neq m$. A simple geometrical explanation of this assertion is as follows: The equation $\exp(in) = \exp(im)$ holds if and only if $n - m = 2k\pi$ holds too. In particular, as the equation $\exp(in) = \exp(i0)$ can never hold for any n, we can find out whether a given constraint c is in the store, by simply checking whether its multiplicity is different from 0.

4 Quantum Operational Semantics

In order to define a quantum operational semantics for PCLP we now introduce the notion of *quantum transition system* (QTS), which describe the general behaviour of a quantum system in terms of a transition relation among the system configurations. We will then consider a special kind of QTS which combines purely quantum transitions with an external probabilistic choice. This kind of QTS which we call *probabilistic quantum transition system* will be used to model computations in PCLP.

4.1 Quantum Transition System

The fact that we have to represent data by vectors in a Hilbert space leads us to introduce the following definition of a quantum configuration.

Let L be a language with T representing its generic syntactic agent, and let $\mathcal{H}$ be a complex Hilbert space.

Definition 2 *A* quantum configuration $C \in \mathrm{Conf}$ *as a pair* $C = \langle T, \vec{x} \rangle$ *of an agent* T *and a vector* $\vec{x} \in \mathcal{H}$.

Most literature assumes a two dimensional Hilbert space $\mathbb{C}^2$ to represent so-called *QBits*, and then considers a *tensor product* $\mathcal{H} = \bigotimes \mathbb{C}^2$ as the computational states (the so-called *quantum register*), [3, 22, 28].

However, since quantum computers are still far from receiving an effective realisation, and it is still an open question what physical phenomena might one day be exploited to implement a quantum computer, we will refer in our model to a generic Hilbert space.

Definition 3 *A* quantum transition system *is a pair* (Conf, $\longrightarrow_q$) *of a set of quantum configurations and a labelled transition relation,* $\longrightarrow_q \subseteq$ Conf $\times$ Conf, *on* Conf, *which we denote by* $C_1 \longrightarrow_q C_2$*. A transition can be of one of the two forms: Either*

$$\langle T_0, \vec{x}_0 \rangle \longrightarrow_1 \langle T_1, \vec{x}_1 \rangle = \langle T_1, \mathbf{U}\vec{x}_0 \rangle$$

with $\mathbf{U}$ *a unitary operator on* $\mathcal{H}$*, or*

$$\langle T_0, \vec{x}_0 \rangle \longrightarrow_{q_i} \langle T_i, \vec{x}_i \rangle = \langle T_i, \mathbf{P}_i\vec{x}_0 \rangle$$

where the $\mathbf{P}_i$ *(for* $i \in I$*, a finite or infinite index set) are pairwise orthogonal projections on* $\mathcal{H}$ *and* q_i *is given by the inner product* $q_i = \langle \vec{x} | \mathbf{P}_i \vec{x} \rangle = \langle \vec{x} | \mathbf{P}_i | \vec{x} \rangle$ *.*

Note that we can have either a (unique) unitary transition from a configuration $\langle T_0, \vec{x}_0 \rangle$ or several projection transitions. If $\vec{x}$ is normalised, then the sum:

$$\sum_i q_i = \sum_i \langle \vec{x} | \mathbf{P}_i | \vec{x} \rangle \leq 1.$$

If the set of projections $\{\mathbf{P}_i\}$ is *complete* on $\mathcal{H}$, i.e. $\mathcal{H} = \bigoplus_i \mathbf{P}_i(\mathcal{H})$ or $\sum_i \mathbf{P}_i = \mathbf{Id}$, then $\sum_i q_i = 1$. In general, we will require that $\sum_i q_i = 1$ for all quantum transitions stemming from a configuration $\langle T, \vec{x} \rangle$. This corresponds to assume that there is either only a single unitary transition, or a complete set of projection transitions $\{\mathbf{P}_i\}$.

4.2 Probabilistic QTS

If we think of a quantum computer as a black box which we are not allowed to interact with (e.g. by measuring it), we can still combine several such boxes in various ways without any classical interaction in order to eventually obtain different results. In particular we can look at computations where an "external", classical choice is made among several possible continuations of a computation. This classical choice could be modelled as a non-deterministic choice, but we will consider here only a probabilistic version.

A black box can be represented by a QTS where only unitary transitions are allowed. A probabilistic transition system is then a quantum transition system where transitions are of the form

$$\langle A_0, \vec{x}_0 \rangle \longrightarrow_{p_i} \langle A_i, \vec{x}_1 \rangle = \langle A_i, \mathbf{U}_i\vec{x}_0 \rangle \,,$$

where $\{p_i\}$ is the distribution associated to the classical choice. We require – as usual – that the sum of classical probabilities related to the transitions from one configuration $\langle A_0, \vec{x}_0 \rangle$ add up to one, i.e. $\sum_i p_i = 1$.

Definition 4 *A* probabilistic quantum transition system *is a QTS where only unitary transitions or* classical probabilistic choices *occur.*

R0	$\langle \mathbf{stop}, \vec{x} \rangle \longrightarrow_1 \langle \mathbf{stop}, \mathbf{Id}\, \vec{x} \rangle$
R1	$\langle \mathbf{tell}(c), \vec{x} \rangle \longrightarrow_1 \langle \mathbf{stop}, \mathbf{U}_c\, \vec{x} \rangle$
R2	$\left\langle \square_{i=1}^n\, p_i : A_i, \vec{x} \right\rangle \longrightarrow_{p_j} \langle A_j, \vec{x} \rangle$
R3	$\dfrac{\langle A_j, \vec{x} \rangle \longrightarrow_p \left\langle A'_j, \vec{x'} \right\rangle}{\langle \Vert_{i=1}^n \;\; A_i, \vec{x} \rangle \longrightarrow_p \left\langle \Vert_{i\in\{1,\ldots,n\}\setminus\{j\}}\, A_i \parallel A'_j, \vec{x'} \right\rangle}$
R4	$\langle proc, \vec{x} \rangle \longrightarrow_1 \langle A, \vec{x} \rangle$ with $proc : -A \in P$

Table 2: A probabilistic quantum transition system for PCLP

4.3 Quantum Operational Semantics for PCLP

The definition of a quantum operational semantics for PCLP can now be given by means of the probabilistic quantum transition system in Table 2.

Both the probabilistic choice and the parallel construct are modelled classically, as in (P)CCP. In particular parallelism is modelled as interleaving, which simulates parallelism on a single processor. However, we observe that, due to the absence of synchronisation in our model all different schedulings produce the same result; thus the sequential execution of all the A_i's is equivalent to any other interleaving.

The only true "quantum operation" is the **tell** operation which is realised via the operator $\mathbf{U}_c$ introduced in Section 3.5.

Definition 5 *A* computational path *for a program P starting in state $\vec{x}$ is a finite sequence of transitions*

$$\langle A_0, \vec{x}_0 \rangle \longrightarrow_{p_1} \langle A_1, \mathbf{U}_1 \vec{x}_0 \rangle \longrightarrow_{p_2} \langle A_2, \mathbf{U}_2 \vec{x}_1 \rangle \ldots \longrightarrow_{p_n} \langle A_n, \mathbf{U}_n \vec{x}_n \rangle\,,$$

where $A_0 = P$, $\vec{x} = \vec{x}_0$ and for each $i \in \{1,\ldots,n\}$, $\vec{x}_i = \mathbf{U}_i \ldots \mathbf{U}_2 \mathbf{U}_1 \vec{x}_0$.

Given a program P, we can represent a computational path for P by a tuple $\langle \mathbf{U}, p \rangle$ where $\mathbf{U} = \mathbf{U}_n \ldots \mathbf{U}_2 \mathbf{U}_1 = \prod_{i=i}^n \mathbf{U}_i$ is the multiplication of the matrices $\mathbf{U}_i$ and $p = \prod_{i=1}^n p_i$ is the product of the probabilities associated to the transitions in a computational path for P. We will denote by $Comp(P, \vec{x})$ the set of all computational paths for P starting in state $\vec{x}$.

We can now define a quantum operational semantics for a program in PCLP as a distribution on vectors in $\mathcal{V}(\mathcal{C})$, each representing a final state in a computational path starting from store *true*.

Definition 6 *The* computational observables *of a program P is the set of all final states of computational paths for P starting with $\vec{true}$ and their associated probabilities:*

$$\mathcal{O}(P) = \{\left\langle \mathbf{U}\vec{true}, p \right\rangle \mid \langle \mathbf{U}, p \rangle \in Comp(P, \vec{true})\}.$$

An alternative way to formulate these observables could be via *density matrices* [13, 16, 27], which represent a convenient language quite popular in quantum mechanics. Following Gleason's Theorem [13], such a formulation would be equivalent to our formulation in terms of state vectors.

Example 3 *Let us consider a simple constraint system* $\mathcal{C} = \{c, d, c \sqcup d\}$ *and two agents* $A = \mathbf{tell}(c) \parallel \mathbf{tell}(d)$ *and* $B = \mathbf{tell}(c \sqcup d)$*. If we represent the store true by the empty multi-set, i.e. by the vector* $\vec{true} = (1, 1, 1) \in \mathbb{C}^3$*, we obtain the following quantum observables for A:*

$$\mathcal{O}(A) = \{\langle 1, (\exp(i), \exp(i), 1)\rangle\}$$

as we have:

$$\mathbf{U}_d \cdot \mathbf{U}_c \cdot \vec{true} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & e^i & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} e^i & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} e^i \\ e^i \\ 1 \end{pmatrix}$$

In a similar way we obtain for B the following observables:

$$\mathcal{O}(B) = \{\langle 1, (1, 1, \exp(i))\rangle\}$$

because:

$$\mathbf{U}_{c \sqcup d} \cdot \vec{true} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & e^i \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ e^i \end{pmatrix}$$

Although $\mathcal{O}(A)$ *and* $\mathcal{O}(B)$ *are different, it is easy to check that the lub of all constraints with non-real coefficients is the same. In fact the multi-set* $\{\!|c, d|\!\}$ *represented by* $(\exp(i), \exp(i), 1)$ *and the multi-set* $\{\!|c \sqcup d|\!\}$ *represented by* $(1, 1, \exp(i))$ *have the same lub, namely* $c \sqcup d$*.*

5 Conclusions and Related Work

Although concrete realisations of quantum computers may be years ahead, we think it is time to start in this area investigations in semantics, in particular on high-level quantum languages. Our optimistic view is that in this way we can anticipate all the hard work which in classical computing has been necessary to gradually progress from COBOL and Fortran to more high-level languages.

Most work on quantum computation use so-called "quantum circuits" to describe, specify, and reason about quantum algorithms. These are basically graphical representations of networks of so-called "quantum gates", i.e. standard (unitary) operators. They specify a concrete receipt how to build more complicated quantum processes out of a few basic (hypothetical) building blocks. The abstraction level using such a kind of "quantum programming language" is obviously very low, and comparable to the use of boolean circuits or assembler languages. At this level there are obviously severe problems in describing higher level aspects, like data structures, etc.

Up to now, the only two higher level attempts to define a true quantum programming *language* we are aware of are two imperative languages: The first approach by Ömer in Vienna [23, 24] has a C-like syntax while a second proposal by Sanders and Zuliani in Oxford [28] is based on Dijkstra's guarded-command language. In both cases a formal syntax is specified, but the semantics is discussed in a quite informal and ad-hoc way. Ömer essentially provides an implementation which simulates a quantum computer, while Sanders and Zuliani refer to an explicit QBit based representation. Their main motivation is to provide a language for formally specifying quantum algorithms like Grover's search algorithm and Shor's factorisation algorithm.

In our work we aim at a more general objective, namely the development of a semantical framework which could also allow for the study of the expressive possibilities and limitations of quantum programming languages. Ultimately, we aim to develop appropriate quantum semantical notions, tools and methodologies (like quantum type systems, quantum program logics, etc.) which generalise (or represent the analogous of) the ones available for classical languages. The definition of a quantum operational semantics based on the notion of a quantum transition system, and the discussion of a quantum version of a declarative programming language presented in this paper intends to be a first step towards the development of a *quantum semantics*.

References

[1] Paul A. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*, 22(5):563–591, 1980.

[2] Charles H. Bennett. Logical reversibility of computations. *IBM Journal of Research and Development*, 17(6):525–532, 1973.

[3] Gennady P. Berman, Gary D. Doolen, Ronnie Mainieri, and Vladimir I. Tsifrinovich. *Introduction to Quantum Computers*. World Scientific, Singapore, 1998.

[4] Garrett Birkhoff and John von Neumann. The logic of quantum mechanics. *Annals of Mathematics*, 37:823–843, 1936. in [15].

[5] Ola Bratteli and Derek W. Robinson. *Operator Algebras and Quantum Statistical Mechanics*, volume 1 – C^*- and W^*-Algebras, Symmetry Groups, Decomposition of States. Springer Verlag, New York – Heidelberg – Berlin, 1979.

[6] Frank S. de Boer, Alessandra Di Pierro, and Catuscia Palamidessi. Nondeterminism and Infinite Computations in Constraint Programming. *Theoretical Computer Science*, 151(1):37–78, 1995.

[7] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London*, A400:97–117, 1985.

[8] Alessandra Di Pierro and Herbert Wiklicky. An operational semantics for Probabilistic Concurrent Constraint Programming. In P. Iyer, Y. Choo, and D. Schmidt, editors, *ICCL'98 – International Conference on Computer Languages*, pages 174–183. IEEE Computer Society Press, 1998.

[9] Alessandra Di Pierro and Herbert Wiklicky. Quantitative observables and averages in Probabilistic Concurrent Constraint Programming. In K.R. Apt, T. Kakas, E. Monfroy, and F. Rossi, editors, *New Trends in Constraints*, number 1865 in Lecture Notes in Computer Science, Berlin — Heidelberg — New York, 2000. Springer Verlag.

[10] Samuel Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, New York – London, 1974.

[11] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7):467–488, 1982.

[12] Richard P. Feynman, Robert B. Leighton, and Mattew Sand. *The Feynman Lectures on Physics – Quantum Mechanics*, volume 3. Addison-Wesley, Reading, Massachusetts, 1965.

[13] Andrew M. Gleason. Measures of closed subspaces of a Hilbert space. *Journal of Mathematics and Mechanics*, 6:885–893, 1957. reprint in [15].

[14] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of STOC'96 – Symposium on the Theory of Computing*, pages 212–219, Philadelphia, Pennsylvania, 1996. ACM.

[15] Clifford A. Hooker, editor. *The Logico-Algebraic Approach to Quantum Mechanics*, volume I – Historical Evolution. Reidel, Dordrecht – Bosten, 1975.

[16] Chris J. Isham. *Lectures on Quantum Theory — Mathematical and Structural Foundations.* Imperial College Press, London, 1995.

[17] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *The Journal of Logic Programming*, 19 & 20:503–582, May 1994.

[18] Richard V. Kadison and John R. Ringrose. *Fundamentals of the Theory of Operator Algebras*, volume I – Elementary Theory of *Pure and Applied Mathematics.* Academic Press, New York – London, 1983.

[19] Richard V. Kadison and John R. Ringrose. *Fundamentals of the Theory of Operator Algebras*, volume II – Advanced Theory of *Pure and Applied Mathematics.* Academic Press, Orlando – London, 1986.

[20] George W. Makey. *The Mathematical Foundations of Quantum Mechanics.* W.A. Benjamin, New York – Amsterdam, 1963.

[21] Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An Introduction.* MIT Press, Cambridge, Massachusetts – London, England, 1998.

[22] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, 2000.

[23] Bernhard Ömer. A procedural formalism for quantum computing. Technical report, Department of Theoretical Physics – Technical University Vienna, Vienna, 1998.

[24] Bernhard Ömer. Quantum programming in qcl. Technical report, Institute of Information Systems – Technical University Vienna, Vienna, 2000.

[25] Gordon D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[26] Viktor V. Prasolov. *Problems and Theorems in Linear Algebra*, volume 134 of *Translation of Mathematical Monographs*. American Mathematical Society, Providence, Rhode Island, 1994.

[27] John Preskill. Quantum information and computation. Technical Report 229, California Institute of Technology, `http://theory.caltech.edu/~preskill`, September 1998.

[28] Jeff W. Sanders and Paolo Zuliani. Quantum programming. In R. Backhouse and J.S. Nuno Oliveira, editors, *Proceedings of MPC 2000, Mathematics of Program Construction - 5th International Conference*, volume 1837 of *Lecture Notes in Computer Science*, pages 80–99. Springer-Verlag, 2000.

[29] Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. Semantics foundations of concurrent constraint programming. In *Proceedings of POPL'91 – Symposium on Principles of Programming Languages*, pages 333–353. ACM, 1991.

[30] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of FOCS'94 – Symposium on Foundations of Computer Science*, pages 124–134, Santa Fe, New Mexico, 1994. IEEE Press.

[31] Max Tegmark and John Archibald Wheeler. 100 years of quantum mysteries. *Scientific American*, 284(2):54–61, 2001.

[32] Walter Thirring. *A Course in Mathematical Physics – Quantum Mechanics of Atoms and Molecules*, volume 3. Springer-Verlag, New York – Wien, 1981.

[33] John von Neumann. *Mathematical Foundations of Quantum Mechanics*. Princeton University Press, Princeton, 1955.

[34] Herbert Wiklicky. Quantitative computation by Hilbert machines. In C.S Calude, J. Casti, and M.J. Dinneen, editors, *UMC'98 – Unconventional Models of Computation*, pages 200–220, Singapore, 1998. Springer Verlag.

[35] Kôsaku Yosida. *Functional Analysis*. Springer Verlag, Berlin – Heidelberg – New York, sixth edition, 1980.

A Quantum Mathematics

As a reference for some the most important notions related to Hilbert space we provide in this appendix a list of definitions. For more detailed explanations we refer to the standard text books on linear algebra, Hilbert spaces, functional analysis, or operator theory (e.g. [26, 35, 18, 19]).

A.1 Linear Spaces

Definition 7 *A* vector space *or* linear space *over a field $\mathbb{K}$ is a set $\mathcal{V}$ together with two operations:* scalar product $.\cdot. : \mathbb{K} \times \mathcal{V} \mapsto \mathcal{V}$ *and* vector addition $.+. : \mathcal{V} \times \mathcal{V} \mapsto \mathcal{V}$*; and a* null vector $\vec{o} \in \mathcal{V}$ *satisfying the following axioms ($\forall \vec{x}, \vec{y} \in \mathcal{V}$ and $\forall \alpha, \beta \in \mathbb{C}$):*

1. $\vec{x} + (\vec{y} + \vec{z}) = (\vec{x} + \vec{y}) + \vec{z}$
2. $\vec{x} + \vec{y} = \vec{y} + \vec{x}$
3. $\vec{x} + \vec{o} = \vec{x}$
4. $\exists - \vec{x} \in \mathcal{V} : \vec{x} + (-\vec{x}) = \vec{o}$
5. $\alpha(\vec{x} + \vec{y}) = \alpha\vec{x} + \alpha\vec{y}$
6. $(\alpha + \beta)\vec{x} = \alpha\vec{x} + \beta\vec{x}$
7. $(\alpha\beta)\vec{x} = \alpha(\beta\vec{x})$
8. $1\,\vec{x} = \vec{x}$

with 1 *the (multiplicative) identity in* $\mathbb{K}$.

For $\mathbb{K} = \mathbb{R}$ we speak of a *real vector space*, and for $\mathbb{K} = \mathbb{C}$ we have a *complex vector space*. The standard examples of finite dimensional vector spaces are $\mathbb{R}^n$ and $\mathbb{C}^n$.

Definition 8 *A (complex) vector space $\mathcal{V}$ is called a* normed vector space *if there is a real valued function $\|.\|$ on $\mathcal{V}$ that satisfies ($\forall \vec{x}, \vec{y} \in \mathcal{V}$ and $\forall \alpha \in \mathbb{C}$):*

1. $\|\vec{x}\| \geq 0$
2. $\|\vec{x}\| = 0$ *iff* $\vec{x} = \vec{o}$
3. $\|\alpha\vec{x}\| = |\alpha|\ \|\vec{x}\|$
4. $\|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$

The function $\|.\|$ is called norm *on* $\mathcal{V}$.

Definition 9 *A (complex) vector space $\mathcal{V}$ is called an* inner product space *(or* pre-Hilbert space*) if there is a complex-valued function $\langle ., . \rangle$ on $\mathcal{V} \times \mathcal{V}$ that satisfies ($\forall \vec{x}, \vec{y}, \vec{z} \in \mathcal{V}$ and $\forall \alpha \in \mathbb{C}$):*

1. $\langle \vec{x}, \vec{x} \rangle \geq 0$ *(by (5) we have: $\langle \vec{x}, \vec{x} \rangle \in \mathbb{R}$)*
2. $\langle \vec{x}, \vec{x} \rangle = 0$ *iff* $\vec{x} = \vec{o}$
3. $\langle \vec{x}, \alpha\vec{y} \rangle = \alpha \langle \vec{x}, \vec{y} \rangle$
4. $\langle \vec{x}, \vec{y} + \vec{z} \rangle = \langle \vec{x}, \vec{y} \rangle + \langle \vec{x}, \vec{z} \rangle$
5. $\langle \vec{x}, \vec{y} \rangle = \overline{\langle \vec{y}, \vec{x} \rangle}$

where $\bar{.}$ *denotes the complex conjugation. The function* $\langle ., . \rangle$ *is called an* inner product *on* $\mathcal{V}$.

Note that in physics the inner product is often denoted by $\langle .|. \rangle$ (which also makes it somehow easier to distinguish it from pairs $\langle ., . \rangle$). Furthermore, there is some inconsistency regarding the issue if the inner product is linear in the first argument or in the second, cf. axiom (4).

Definition 10 *An inner product space $\mathcal{H}$ which is complete with respect to the canonical metric* $d(\vec{x}, \vec{y}) = \sqrt{\langle \vec{x} - \vec{y}, \vec{x} - \vec{y} \rangle}$ *is called a* Hilbert space.

Theorem 2 *Any two (separable) Hilbert spaces are isomorphic iff they have the same dimension.*

Any (separable) Hilbert space is isomorphic to $\ell^2(\mathcal{I}) = \left\{ (x_i)_{i \in \mathcal{I}} \mid \sum_{i \in \mathcal{I}} |x_i|^2 < \infty \right\}$, *where $\mathcal{I}$ is a countable (index) set. A Hilbert space of finite dimension n is isomorphic to $\mathbb{C}^n$.*

A.2 Linear Operators

Definition 11 *A map $\mathbf{T}$ between two vector spaces $\mathcal{V}$ and $\mathcal{W}$ is called* linear *if it satisfies ($\forall \vec{x}, \vec{y} \in \mathcal{V}$ and $\forall \alpha \in \mathbb{K}$):*

1. $\mathbf{T}(\vec{x} + \vec{y}) = \mathbf{T}(\vec{x} + \mathbf{T}(\vec{y})$
2. $\mathbf{T}(\alpha\vec{x}) = \alpha\mathbf{T}(\vec{x})$.

The set of all linear maps between two vector spaces $\mathcal{V}$ and $\mathcal{W}$ is denoted by $\mathcal{L}(\mathcal{V}, \mathcal{W})$. If $\mathcal{V} = \mathcal{W}$ we call $\mathbf{T}$ a (linear) operator *on $\mathcal{V}$.*

Theorem 3 *Given (bounded, linear) operator $\mathbf{T}$ on a Hilbert Space $\mathcal{H}$, then there exists a unique, so-called* adjoint operator $\mathbf{T}^*$ *satisfying:*

$$\langle \mathbf{T}(x), y \rangle = \langle x, \mathbf{T}^*(y) \rangle$$

For finite dimensional operators, i.e. $n \times n$ matrices, the adjoint operator is given as the transpose conjugate matrix, i.e. $(t_{ij})^* = \overline{t_{ji}}$.

Definition 12 *A (bounded, linear) operator $\mathbf{A}$ on a Hilbert Space $\mathcal{H}$ is called* self-adjoint *or* hermitian *iff*

$$\mathbf{A}^* = \mathbf{A}.$$

Definition 13 *A (bounded, linear) operator* $\mathbf{U}$ *on a Hilbert Space* $\mathcal{H}$ *is called* unitary *iff*

$$\mathbf{U}^*\mathbf{U} = \mathbf{U}\mathbf{U}^* = \mathbf{Id}$$

where $\mathbf{Id}$ *denotes the identity operator.*

Definition 14 *A (bounded, linear) operator* $\mathbf{P}$ *on a Hilbert Space* $\mathcal{H}$ *is called* projection *iff*

$$\mathbf{P}^2 = \mathbf{P}\mathbf{P} = \mathbf{P}.$$

A (bounded, linear) operator $\mathbf{P}$ *on a Hilbert Space* $\mathcal{H}$ *is called* orthogonal projection *iff*

$$\mathbf{P}^2 = \mathbf{P} = \mathbf{P}^*.$$

Note that in the physical literature, *projections* usually refers to *orthogonal projections*.

The the so-called *spectral theorem* for self-adjoint operators allows us to "decompose" any self-adjoint (or hermitian) operator as a unique sum of projections:

Theorem 4 *A self-adjoint (or hermitian) operator* $\mathbf{A}$ *(on a finite dimensional Hilbert space) can be represented uniquely as a linear combination of (orthogonal) projections:* $\mathbf{A} = \sum_m \lambda_m \mathbf{P}_m$*, where* $\lambda_m \in \mathbb{R}$ *are the so-called* eigen-values *of* $\mathbf{A}$.