Chapter 5
Formal Methods for Quantum Software Engineering

Carmelo R. Cartiere

5.1 Introduction

Although quantum computing (QC) is the future of computing systems, the tools for reasoning about the quantum model of computation, in which the laws obeyed are those on the quantum mechanical scale, are still a mix of linear algebra and Dirac notation—two subjects more suitable for physicists rather than computer scientists and software engineers [17, 18]. On this ground, we believe it is possible to provide a more intuitive but still high-integrity approach to thinking and writing about quantum computing systems, not only to foster the design of quantum algorithms but also to simplify the development of quantum software. Here, we move the first step in such a direction, introducing the Zed (Z) specification language as the means to represent the operations of a quantum computer via axiomatic definitions, also hiring the same symbolisms, semantics, and reasoning principles to which classical software engineers are already used to. We name this novel branch *formal quantum software engineering* (F-QSE) [1].

5.2 Overture to Formal Methods

Formal methods (FM) are a tool of classical software engineering, the distinguishing feature of which is the ability to model and work with complex systems by considering them as mathematical entities.

C. R. Cartiere (✉)
Kellogg College, University of Oxford, Oxford, UK
e-mail: carmelo.cartiere@oxon.org

M. A. Serrano et al. (eds.), *Quantum Software Engineering*,
https://doi.org/10.1007/978-3-031-05324-5_5

With FM, systems are represented with a rigorous mathematical model, which has not only the advantage of having its properties thoroughly verified but also of having its behavior tested via mathematical proof.

Indeed, the use of formal methods in a QC setting can help those who roam the world of computing to both (a) better "understanding and reasoning about the properties of quantum systems" with the adoption of a classical tool of software engineering [2] and (b) describe quantum structures and design quantum algorithms in a more spontaneous way while still adopting a particular form of a mathematically rigorous system [3].

Plus, if we design a QC system using formal specifications (FS), we are developing a set of theorems about that system which, by being proved correct, shall ensure the correct behavior of the system [20].

This is because the trait of FS is to adopt mathematical notations to accurately describe the characteristic properties of a system, without overly limiting how these properties are met, as well as describing the system's behavior, but without dictating how it should do it.

And FSs are helpful during the development process of a system for the reason that they allow to confidently answer the key questions regarding the functions of the system, without neither having to decipher any kind of information by immense amounts of code nor having to investigate the meaning of more or less detailed comments scattered across either the documentation or the code itself.

Since it is detached from the programming code, the guidelines of a FS can already be fulfilled at the early stages of development. Nevertheless, there may be a need to modify it along the way with any design change or addition, as well as when customer requests are changed. But, beyond everything, it is a valuable tool to promote a shared understanding of the system among all people involved in the project.

To say it with Jacky's words, "Using formal methods can be more difficult than programming in the usual way—because formal methods aim higher. Describing exactly what your program does is more difficult than letting testers or users figure it out for themselves. Making your program do the right thing in every situation is more difficult than just handling some typical cases. Any method that can handle hard problems will sometimes be hard to carry out; only superficial methods can be easy all the time. [. . .]. Formal methods make us confront the hard problems early. The difficulties cannot be escaped, only deferred. Superficial methods put off the hard parts until coding and testing—but then they appear with a vengeance. News stories about stressful projects tell of programmers who work eighty-hour weeks, sleep under their desks, punch holes in walls, have nervous breakdowns, and commit suicide [Markoff, 1993; Zachary, 1994]. Compared to that, formal methods don't seem so difficult after all. By making difficult issues more visible, formal methods encourage us to seek a more thorough understanding of the problem we are trying to solve. They require us to express our intentions with exceptional simplicity and clarity. They help us resist the usual tendency in programming to make things too complicated and therefore error-prone and difficult to use" [4].

5.3 The Z Specification Language

In our work, we adopted Z as the FS language of choice: not only because it is already the most (or one of the most) widely used formal languages for describing and modeling the classical computing systems[1] but also because, as Jacky pointed out: "Fortunately, most of the mathematics we need for formal methods is not terribly difficult. The discrete mathematics used in most practical applications of formal methods is easier than much of the calculus that students in the sciences and engineering must study" [4].

The Z specification language permits to build detailed and unambiguous specifications of the behavior of a system. Based on type theory, a branch of symbolic logic that not only formalizes mathematical entities like variables, functions, and operations on them but also formalizes the idea that each entity is of some definite type (e.g., the type $\mathbb{N}$ of natural numbers), it allows to reasoning over the properties of a system (e.g., inputs, transformations, outputs, boundaries) by adopting a *detailed mathematical notation* based on well-defined data structures (e.g., sets, relations, functions) and *logical expressions* written in first-order predicate logic.

It was Jean-Raymond Abrial that in 1977 originally proposed the Z specification language. And when in the 1980s Abrial started working with the Programming Research Group at the University of Oxford, the language was more substantially developed.

Abrial alleged that Z is so named because "it is the ultimate language," but we can also assume that the Z specification language is so-called because it is based on a minimal-typed version of Zermelo-Fraenkel's set theory.

In our description of the F-QSE tools, we shall mainly use Z's *axiomatic definitions*, which are a formal description of the behavior of the system, or part of it, by the means of *declarations* and predicates [19].

An axiomatic definition is drafted in the following form:

$$\begin{array}{|l} x : S \\ \hline p \end{array}$$

In it, we can distinguish the two parts: the *declaration*, made up of the variable x and *basic* type S, and the *predicate* p.

The *declaration* (or signature) is the simplest way to define an object and can be expressed in two ways: if the object corresponds to an original set of elements, or basic type, then either we will write its name in brackets or, if the object is a variable of an already defined set, we shall give it the name of the set that it comes from (with no brackets). For example, the declaration [*Type*] establishes an original basic type

[1] The most widely used notations for developing model-based languages are Vienna Development Method (VDM), Zed (Z), and Bi (B) [5].

called *Type*. The other way is the declaration *x:A*, which establishes a new variable x drawn from the set A (but with the limitation that if this set is not the set $\mathbb{Z}$—i.e., the set of integers—then, in that case, the set must be defined somewhere in the specification) [2].

The *predicate* describes the behavior of the system: it takes as input one or more entities from the domain in question and returns an output that is either True or False [2]. In it, we can find the following logic symbols: $\forall$ (for all), $\exists$ (exists), $\in$ (belongs to), $\bullet$ (such as), $\wedge$ (and), $\vee$ (or), $\Leftrightarrow$ (if . . . and only if . . .), and $\Rightarrow$ (if . . . then . . .).

It is worth mentioning that the *basic type S* shall identify the *maximal set* of the system, that is, a set as much complete as possible within the boundaries of the given specification. This has the effect of making sure that any given value x in the specification shall be associated with exactly one type, that is, the largest set S for which $x \in S$ [2].

So, the adoption of Z for modeling a system requires the formalization of the building blocks of that system, which, in the case of a QC system, are the *observable* and the *observable operators*.

For the sake of simplicity, you can think about the *observable* as the data type that we shall use to declare qubits and about the *observable operators* as the operations that can be performed on qubits.

Once that the *observable* and the *observable operators* have been formalized, it is possible to proceed with the design and implementation of any QC model, i.e., the abstract representation of a QC system, and with its formal verification (FV), through a sequence of four rigorous yet intuitive steps: (a) the *specification*, which is the narrative of the QC system and describes what the system should do; (b) the *refinement*, which is an iterative fine-tuning of the FS and produces the polished QC system; (c) the *proof*, which walks us through the process to prove, or disprove, the properties of the QC system against its FS and demonstrate that the candidate system's design is correct; and, finally, (d) the *implementation*, which is the conversion of the specification into working code.[2]

5.4 An Introduction to the Quantum Computing Observable

Observables (or basis) can be considered the most significant entities of QM. Given that a quantum object (QO) holds many attributes (e.g., position, momentum, energy), one observable completely describes one attribute by conserving all of that attribute's possible states, or eigenstates, in a superposed configuration. In QC systems, QOs have only one observable, the *qubit*, which superposed configuration is the linear combination of its two possible eigenstates. Its quantum state vector,

[2] "The trick of using formal methods effectively is to know when proofs are worth doing and when they are not" [2].

commonly expressed in Dirac's bra-ket notation [6], is, therefore, the linear combination of the two eigenstates' associated eigenvectors $|0\rangle$, $|\ 1\rangle$ ket, which corresponding measurable eigenvalues (the scalars) are 0, 1:

$$\vec{\psi} = |\ 0\rangle + |\ 1\rangle$$

As per its QM counterpart, measuring an observable in a QC system will collapse that observable into one of its eigenstates that for a qubit are those corresponding to either $|0\rangle$ or $|1\rangle$, with probabilities c_0, c_1 [3] [7]:

An easy way to illustrate the concept of an unknown state of an observable (i.e., when the basis' states are in superposition) is by describing Schrödinger's cat: if we receive a cat in a closed box, it can be both dead and alive, with given probabilities, until we open the box (i.e., we observe it). In bra-ket notation, it is simply written:

$$\overrightarrow{\text{cat}} = c_0 \mid \text{alive}\rangle + c_1 \mid \text{dead}\rangle$$

5.4.1 Formalizing the Observable

By the third postulate of QM, an observable that has a finite number of quantum states can be represented via a Hermitian matrix.[4] As such, the three requirements that it must have can be described, with a sound formalism, by adopting strongly typed data and first-order logic [3]; i.e.:

1. It must be a complex square matrix of order n:

$$\mid \mathbb{O}^n : \mathbb{C}^{n \times n}$$

[3] The probability for an observable to collapse into any of its states is the squared modulus of the states' corresponding probability amplitudes, which are complex numbers that weight each eigenvector and such that it is $|c_0|^2 + \ldots + |c_n|^2 = 1$.

[4] But if the Hilbert space $\mathcal{H}$ is infinite-dimensional, the observable is described by a *symmetric operator*, which is represented as a map f between two domains of basis' states D and D^* dense in $\mathcal{H}$, such that $\forall x : D, y : D^* \ \exists f : D \mapsto D^* \bullet \langle f(x), y \rangle = \langle x, f(y) \rangle$. This is a *bijective function* (injective-surjective), in the sense that it cannot map two distinct states of the domain D onto the same state of the co-domain D^*, thus preserving its unitary quality. However, because an infinite-dimensional space is unbounded, also the operator is unbounded; therefore, it does not have a largest eigenvalue, leaving us with the conclusion that it might not be defined everywhere and, as such, classifying it as a *partial bijective* function, which implies graph inclusion: $D \leq D^*$.

2. It must be equivalent to its conjugate transpose:

$$\forall c : \mathbb{O}^n \; \exists_1 \; c' : \overline{\mathbb{O}^n}^T \; \bullet \left(c_{ij} = c'_{ij}\right)$$

3. For every eigenvector (or column) of the matrix, the eigenvalue must be a real number; and such that it is the element on the main diagonal of the matrix:

$$\forall V^{n\times 1} : \mathbb{PO}^n \; \exists_1 \; \lambda : \mathbb{R} \; \bullet \lambda = c_{jj} \epsilon \mathbb{O}^n$$

In Z, all three requirements can be summarized with the following axiomatic definition satisfying the principle of soundness promoted by FM [2]:

$$\mathbb{O}^n : \mathbb{C}^{n\times n}$$

$$\forall c : \mathbb{O}^n \; \exists_1 \; c' : \overline{\mathbb{O}^n}^T \; \bullet \left(c_{ij} = c'_{ij}\right) \epsilon \, \mathbb{C} \wedge c_{jj} \epsilon \, \mathbb{R}$$

5.4.2 The Observable Operators

After having introduced the new type $\mathbb{O}^n$, it is now possible to define the observable operators. They are elementary quantum gates that perform unitary transformations U_f (i.e., reversible computations) and that, applied to an observable, make it possible to write quantum programs.

As we will see, most of quantum gates only need to perform one operation during a transformation, for example, when they make a classical state into a superposition state, while only two operations are needed to form an entanglement between two qubits.

In the following paragraphs, we introduce the axiomatic definition of the most common quantum gates: *Identity*, *Pauli-X*, *Phase Shift*, *Pauli-Z*, *Hadamard*, and *C-Not*. In this way, we shall have the necessary mathematical toolkit to design quantum software in Z.

Identity Gate. It is the simplest, single qubit, quantum operator that maps the input to the output unchanged. It is required by any operation where the same qubits that are passed as arguments must be returned:

$$I : \mathbb{O}^2 \rightarrowtail \mathbb{O}^2$$

$$\forall x : \{0,1\}\, \exists r : \{|x\rangle \longmapsto |x\rangle\} \bullet r \in I \Leftrightarrow (|0\rangle \longmapsto |0\rangle \wedge |1\rangle \longmapsto |1\rangle) \in r$$

Pauli-X (or Bit-Flip) Gate. It is the quantum equivalent of the classical NOT gate:

$$X : \mathbb{O}^2 \rightarrowtail \mathbb{O}^2$$

$$\forall x : \{0,1\}\, \exists r : \{|x\rangle \longmapsto |x'\rangle\} \bullet r \in X \Leftrightarrow (|0\rangle \longmapsto |1\rangle \wedge |1\rangle \longmapsto |0\rangle) \in r$$

Phase Shift Gate. It represents a family of gates that rotate the basis' state $|1\rangle$ of any arbitrary angle ϕ:

$$R_\phi : \mathbb{O}^2 \rightarrowtail \mathbb{O}^2$$

$$\forall x : \{0,1\}\, \exists r : \{|x\rangle \longmapsto |x'\rangle\} \bullet r \in R_\phi \Leftrightarrow (|0\rangle \longmapsto |0\rangle \wedge |1\rangle \longmapsto |e^{i\phi}\rangle) \in r$$

Pauli-Z (or π Phase Shift) Gate. It is a special case of the Phase Shift gate that rotates the basis' state $|1\rangle$ a π angle:

$$\mathrm{Z} : \mathbb{O}^2 \rightarrowtail \mathbb{O}^2$$

$$\forall x : \{0,1\}\, \exists r : \{|x\rangle \longmapsto |x'\rangle\} \bullet r \in Z \Leftrightarrow (|0\rangle \longmapsto |0\rangle \wedge |1\rangle \longmapsto -|1\rangle) \in r$$

Hadamard Gate. It is perhaps the most useful quantum operator because it maps any basis' state to one qubit with balanced superposition and vice versa:

$$H : \mathbb{O}^2 \rightarrowtail \mathbb{O}^2$$

$$\forall x : \{0,+,1,-\}\, \exists r : \{|x\rangle \longmapsto |x'\rangle\} \bullet r \in H \Leftrightarrow (|0\rangle \longmapsto |+\rangle \wedge |+\rangle \longmapsto -|0\rangle) \in r \wedge (|1\rangle \longmapsto |-\rangle \wedge |-\rangle \longmapsto -|1\rangle) \in r$$

C-Not Gate. The Controlled Not gate is the most popular two-qubit operator because it puts two qubits in a separable state, where a tensor product pairs the first

qubit with the result of an addition modulo-2 between both. As such, it is used to entangle two qubits or disentangle the EPR pair:

$$N : \mathbb{O}^4 \nrightarrow \mathbb{O}^4$$

$$\forall x, y : \{0,1\}\ \exists r : \{|xy\rangle \longmapsto |xy'\rangle\} \bullet r \in N \Leftrightarrow (|0y\rangle \longmapsto |0y\rangle \land |1y\rangle \longmapsto -|1\bar{y}\rangle) \in r$$

Similar to what happens in any conventional computation, quantum computations are just a sequence of gates applied in a particular order: each gate takes an input and, after having performed its operation on that input, returns an output. However, in QC, the single use of an operator simultaneously applies to all basis' states [8].

5.5 A Practical Example of F-QSE: Programming the Deutsch Algorithm from Specifications

By using FM, it is possible to describing and implementing quantum algorithms despite their complexity.

The Deutsch algorithm, the foundation model of QC [9, 10, 16], proves if a quantum oracle, i.e., a black box that performs a unitary transformation U_f on a qubit, is either *constant* (always maximizing the same state) or *balanced* (returning each state half of the time). It exploits the quantum entanglement principle [9] and requires the use of two quantum operators: a Hadamard gate, for preparing two qubits in balanced superposition, and a C-Not gate, for entangling the two qubits.

In Dirac notation, it is represented as a ket taking a pair of qubits, prepared from two different basis' states (x and y), and mapping them to an entangled pair where the second qubit performs as the register storing the state (solution) that will be set on the first qubit by the quantum oracle. The observation (measurement) of the first qubit shall, therefore, make it collapse into the state ($|0\rangle$ for *constant*, $|1\rangle$ for *balanced*) that is held by the second qubit, to which it is entangled:

$$|x, y\rangle \overset{U_f}{\rightarrow} |x, f(x) \oplus y\rangle$$

With the Z notation, the algorithm can be described through axiomatic definitions, either by importing within the constraining predicate the conventional Dirac representation (which is sound but doesn't add much in a SE perspective) (Fig. 5.1):

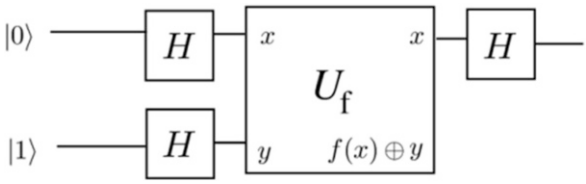

Fig. 5.1 The quantum circuit for the Deutsch algorithm

$$deutsch : (\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2) \longrightarrow \mathbb{Z}_2$$

$$\forall f : (\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2);\ x, y : \{0,1\} \bullet deutsch\ (f) = |x, f\ (x) \oplus y\rangle$$

or by taking advantage of the axiomatic definitions already shaped for the observable operators, writing:

$$deutsch : (\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2) \longrightarrow \mathbb{Z}_2$$

$$\forall f : (\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2);\ x, y : \{0,1\} \bullet deutsch\ (f) = N_f \mid x \leftarrow H\ |0\rangle, y \leftarrow H\ |1\rangle\rangle \wedge H\ |x\rangle$$

rather than:

$$deutsch : (\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2) \longrightarrow \mathbb{Z}_2$$

$$\forall f : (\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2);\ x, y : \{0,1\} \bullet deutsch\ (f) = x \leftarrow H\ |0\rangle \wedge y \leftarrow H\ |1\rangle \wedge N_f\ |x, y\rangle \wedge H\ |x\rangle$$

Indeed, with the last two definitions, by describing the algorithm through a sequence of formal operators, we offer guidance for coding it by directly following the stepwise logic represented.

Of course, the coding part can be done in any quantum programming language. For our case, to match the formal definitions introduced, we worked out an instructions' set in Haskell that leans on Green's QIO library [11].

The Deutsch algorithm can now be, straightforwardly, translated into the following QC program:

```
deutsch :: (Bool ⟶ Bool) ⟶ QIO ( Bool )
deutsch f = do
x ←— qb( "H|0⟩" )
```

$y \longleftarrow qb(\text{"}H|1\rangle\text{"})$
$qN(f)\ x\ y$
$qH(x)$
$mq(x)$

5.6 Another Practical Example of F-QSE: The Quantum Teleportation Protocol

The Quantum Teleportation Protocol (QTP) is an algorithm that was firstly published by Bennett et al. in 1993 and which can be used to transfer a quantum state between two remote endpoints A and B (say, Alice and Bob).

The QTP is at the base of the so-called superdense coding; that is, you communicate two bits of classical information by only sending out one single qubit.

The foundation of the QTP is the entanglement principle (EP): when two remote and not physically connected objects have in the past interacted within the same local system, they remain linked forever; and each modification of the state of one of them induces a modification into the state of the other one.

One practical use of the QTP is the possibility to carry out secure communications in such a way that the cryptographic key does not need to be transferred between the two endpoints but can just be teleported. By doing this, any risk of eavesdropping is completely cancelled.

Now, with the help of a short storytelling, we will show a handy example of how to implement the QTP with the use of FM.

Alice and Bob are two secret agents who met a long time ago but now live far apart. During the time spent together, they generated an EPR (Einstein-Podolsky-Rosen) pair or Bell state[5] [13]:

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

The simplest way to do it is to set one qubit in superposition with the use of a Hadamard gate and, applying a C-Not gate, entangle it with a second qubit of known state:

[5] Bell states represent the simplest example of quantum entanglement and are a form of two maximally entangled basis' state vectors (qubits) which are pure (cannot be represented as a combination of other basis' states) and normalized (the overall probability of the particle to be in one of the two basis' states is 1): $\langle\Phi|\Phi\rangle = 1$.

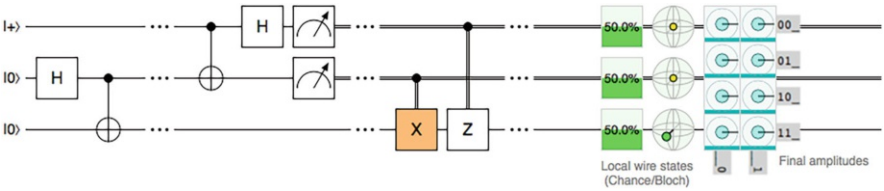

Fig. 5.2 The quantum teleportation circuit proposed by Bennett et al. [12]

$$|\beta_{xy}\rangle \overset{U_f}{\rightarrow} |x\rangle \oplus y$$

As already seen in Deutsch's algorithm, thanks to the use of Zed's axiomatic definition, we can express the constraining predicate with the conventional Dirac form as:

$$\begin{array}{|l} bell : \mathbb{Z}_2 \longrightarrow (\mathbb{O}^2 \rightarrow \mathbb{O}^2) \\ \hline \forall f \in bell;\ x : \{0,1\};\ y : \{0\} \bullet f = |x\rangle \oplus y \end{array}$$

However, our aim is to provide a clearer definition of the algorithm for SE: something that can help the classically trained base of software engineers to go from zero (the definition) to hero (the code). And it can be easily done by recruiting the observable operators already defined, as:

$$\begin{array}{|l} bell : \mathbb{Z}_2 \longrightarrow (\mathbb{O}^2 \longrightarrow \mathbb{O}^2) \\ \hline \forall f \in bell;\ x : \{0,1\};\ y : \{0\} \bullet f = N \mid qa \leftarrow H\,|x\rangle, qb \leftarrow |y\rangle\ \rangle \end{array}$$

rather than as:

$$\begin{array}{|l} bell : \mathbb{Z}_2 \longrightarrow (\mathbb{O}^2 \longrightarrow \mathbb{O}^2) \\ \hline \forall f \in bell;\ x : \{0,1\};\ y : \{0\} \bullet f = qa \leftarrow H\,|x\rangle \wedge qb \leftarrow |y\rangle \wedge N \mid qa, qb\ \rangle \end{array}$$

When Alice and Bob had to part away, each of them took one piece of the EPR pair (*qa* and *qb*). At some point in his life, Bob has to hide himself, and Alice's mission is to deliver a message to Bob (*qdata*). This must be done by both preventing that the message can be eavesdropped and that anybody can use the transmission to track down Bob's location.

Alice does pair *qa* and *qdata*, performs a joint measurement with the intention of detecting on which of the four Bell states they have been projected, and sends (somewhere) to Bob the two classical bits obtained (*cdata*):

Table 5.1 The *cdata* map

In	Out
00	$(\|\mathbf{00}\rangle + \|\mathbf{11}\rangle)/\sqrt{\mathbf{2}}\rangle \equiv \|\boldsymbol{\beta}_{\mathbf{00}}\rangle$
01	$(\|\mathbf{01}\rangle + \|\mathbf{10}\rangle)/\sqrt{\mathbf{2}}\rangle \equiv \|\boldsymbol{\beta}_{\mathbf{01}}\rangle$
10	$(\|\mathbf{00}\rangle - \|\mathbf{11}\rangle)/\sqrt{\mathbf{2}}\rangle \equiv \|\boldsymbol{\beta}_{\mathbf{10}}\rangle$
11	$(\|\mathbf{01}\rangle - \|\mathbf{10}\rangle)/\sqrt{\mathbf{2}}\rangle \equiv \|\boldsymbol{\beta}_{\mathbf{11}}\rangle$

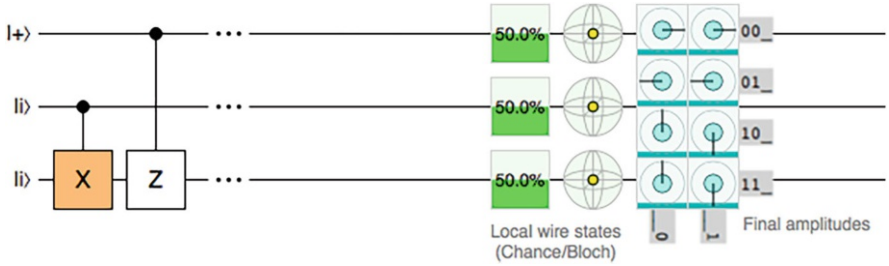

Fig. 5.3 The unitary operations that Bob must perform, controlled by *cdata*

$$\begin{array}{|l} alice : \mathbb{O}^2 \longrightarrow \mathbb{O}^2 \longrightarrow (\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2) \\ \overline{\forall f \in alice;\ qa, qdata : \mathbb{O}^2} \bullet f = N\ |qdata, qa\rangle \wedge H\ |qdata\rangle \end{array}$$

For his part, Bob, who possesses a qubit of the EPR pair, that is now collapsed due to the measurement performed by Alice, receives the two classical bits that will let him to conditionally apply any of four given quantum gates to his part of the (collapsed) EPR pair, obtaining in return the original message (Table 5.1).

$$\begin{array}{|l} bobcond : (\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2) \to \mathbb{O}^2 \\ \overline{\forall f \in bobcond;\ qb : \mathbb{O}^2} \bullet f = (|00\rangle \Rightarrow I(qb)) \vee (|01\rangle \Longrightarrow X(qb)) \vee (|10\rangle \\ \qquad\qquad \Longrightarrow Z(qb)) \vee (|11\rangle \Rightarrow X(z(qb))) \end{array}$$

$$\begin{array}{|l} bob : \mathbb{O}^2 \to (\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2) \to \mathbb{O}^2 \\ \overline{\forall f \in bob;\ qb : \mathbb{O}^2, cdata : (\mathbb{Z}_2 \longrightarrow \mathbb{Z}_2)} \bullet f = bobcond\ cdata\ qb \end{array}$$

The Quantum Teleportation Protocol is now completely described, and we can translate it into an axiomatic definition and a corresponding QC program:[6]

[6]This program will not break the no-cloning theorem, because the state of the original qubit shall be lost during the process.

$$
\begin{array}{|l}
qtp : \mathbb{O}^2 \to \mathbb{O}^2 \\
\hline
\forall f \in qtp;\ qdata, qa, \\
\qquad\qquad\qquad qb : \mathbb{O}^2 \bullet f = (qa, qb) \leftarrow bell\ (0) \wedge cdata \\
\qquad\qquad\qquad \leftarrow alice\ (qa)\ qdata \wedge tdata \leftarrow bob\ (qb)\ cdata
\end{array}
$$

$$
\begin{array}{l}
qtp :: Qbit \to QIO\ (Qbit) \\
qtp\ qdata = do \\
(qa, qb) \leftarrow bell\ (b0) \\
cdata \leftarrow alice\ (qa)\ qdata \\
tdata \leftarrow bob\ (qb)\ cdata \\
return\ (tdata)
\end{array}
$$

From the point of view of a classical software engineer, the QTP circuit is intrinsically complex in order to be used as a guide for coding, but even by following the verbal description of the QTP, it is not easy to interpret and transform the algorithm into code.

Therefore, we cannot fail to appreciate both the clarity of the axiomatic definition to describe the QTP and the guidance it offers to write the code needed to perform such a powerful quantum function. And, with it, we also have the advantage of eliminating (or, at least, minimizing) the risk of introducing either conceptual errors during the drafting of the algorithm or coding errors during the transformation of the algorithm into a working program.

Finally, the formalization of the algorithm produced by Z can help to reason beyond its primitive use, with the possibility to extend the same logical structure for identifying use cases that go beyond the particular instance, as, for example, in the QTP paradigm, to describe the operations required to teleport matter and energy [21]. But this is a topic for another study.

5.7 Conclusions and Outlooks

The diffusion of QC cannot be forever relegated within a narrow circle of experts, but many computer scientists and software engineers entering the field of QC are quickly put off by the existing conceptual and notational barriers [14]. This is not only due to the intrinsic difficulty of the subject but also because it can only be seen through a dark glass (as the complete knowledge of the state of a quantum system is forbidden) [15].

Not only does QC require a completely different mindset, but in order to make quantum computers available to everyone, we need to prepare a QC-ready workforce capable of translating old and new challenges into problems that quantum computers can understand.

One possible way to overcome this stasis is to tap into the existing broad base of software engineers, introducing a vocabulary inspired by formal SE tools. In this

work, you learned how the main notions of QC can take the form of axiomatic definitions in Z notation so that they can be used throughout specifications [2]. The result is a notational system that, ideally, can open the doors of QC to the wider audience of players, helping them to understand, describe, and, ultimately, translate the structure of a quantum algorithm into fully working code, adopting any quantum programming language that is available.

Appendix

A.1 Coding of Typical Quantum Operators

In the following sections, you will find the complete implementation of the quantum operators (QO) required to run the code used in the proposed examples.

These QO have been designed based on the QIO Monad, which is a Haskell library of purely functional interfaces for quantum programming [11].

A.1.1 QO for the Deutsch Algorithm

— return a qubit in a specified state

```
qb :: [Char] -> QIO ( Qbit )
qb qstate
  | qstate == "|0>" = mkQ( False )
  | qstate == "|1>" = mkQ( True )
  | qstate == "|+>" || qstate == "H|0>" = do
            qBit <- qb( "|0>" )
            applyU( uhad( qBit ) )
            return( qBit )
  | qstate == "|->" || qstate == "H|1>" = do
            qBit <- qb( "|1>" )
            applyU( uhad( qBit ) )
            return( qBit )
  | otherwise = error "qb: wrong argument"
```

— apply the C-Not (N) gate to a qubit

```
qN :: (Bool -> Bool) -> Qbit -> Qbit -> QIO ()
qN f qx qy = applyU( cond (qx) (\ a → if f(a) then unot(qy) else mempty ))
```

— apply the Hadamard (H) gate to a qubit

```
qH :: Qbit -> QIO ()
qH qbit = applyU( uhad( qbit ) )
```

— measure a qubit

```
mq :: Qbit -> QIO ( Bool )
mq qbit = measQ( qbit )
```

A.1.2 QO for the Quantum Teleportation Protocol

— return False

```
b0 :: Bool
b0 = (0==1)
```

— return True

```
b1 :: Bool
b1 = (1==1)
```

— apply the C-Not (N) gate to a qubit

```
qN :: Qbit -> Qbit -> QIO ()
qN qx qy = applyU( cond ( qx ) (\ a -> if a then unot( qy ) else mempty ) )
```

— apply the Hadamard (H) gate to a qubit

```
qH :: Qbit -> QIO ()
qH qb = applyU( uhad( qb ) )
```

— apply the Identity (I) gate to a qubit

```
qI :: Qbit -> U
qI qb = mempty
```

— apply the Not (X) gate to a qubit

```
qX :: Qbit -> U
qX qb = unot( qb )
```

— apply the Pi Phase Shift (Z) gate to a qubit

```
qZ :: Qbit -> U
qZ qb = (uphase qb pi)
```

— apply the ZX sequence of gates to a qubit

```
qZX :: Qbit -> U
qZX qb = qX( qb ) `mappend` qZ( qb )
```

— create a Bell state by sharing a qubit in superposition with a qubit in given state

```
bell :: Bool -> QIO (Qbit,Qbit)
bell qf = do
   qa <- if not qf then qb("|+>") else qb("|->")
   qb <- qb("|0>")
   qN qa qb
   return (qa,qb)
alice :: Qbit -> Qbit -> QIO (Bool,Bool)
alice qa qdata = do
```

— Alice applies the C-Not gate to qa, controlled by qdata (the information to be sent)

```
qN (qdata) qa
```

— Alice applies the Hadamard gate to qdata

```
qH (qdata)
```

```
— Alice measures her qubits, collapsing them; and stores the result in two classical bits
cdata <- mq (qdata,qa)
return (cdata)

bobcond :: (Bool, Bool) -> Qbit -> U


bobcond (False, False) qb = qI qb — do nothing

bobcond (False, True ) qb = qX qb — apply the X gate (not gate)

bobcond (True , False) qb = qZ qb — apply the Z gate (pi phase shift gate)

bobcond (True , True ) qb = qZX qb — apply the ZX sequence of gates

bob :: Qbit -> (Bool, Bool) -> QIO ( Qbit )
bob qb cdata = do

— Bob applies the relevant gate to qb, which choice is controlled by the classical bits received
applyU (bobcond cdata qb)

— Bob now finally has the result of the manipulation of qb
return (qb)
```

References[7]

1. Cartiere CR (2020) Formal quantum software engineering: introducing the formal methods of software engineering to quantum computing. https://doi.org/10.13140/RG.2.2.26157.10725/2
2. Woodcock J, Davies J (1996) Using Z. Specification, refinement, and proof. Prentice Hall
3. Cartiere CR (2013) Quantum software engineering: bringing the classical software engineering into the quantum domain. Master's Thesis, University of Oxford, Department of Computer Science, Software Engineering Programme
4. Jacky J (1996) The way of Z: practical programming with formal methods. Cambridge University Press
5. Ruhela V (2012) Z formal specification language – an overview. Int J Eng Res Technol (IJERT) 01(06)
6. Dirac P (1958) The principles of quantum mechanics, 4th edn. Oxford University Press
7. Mateus P, Sernadas A (2004) Reasoning about quantum systems. In: Alferes JJ, Leite J (eds) Logics in artificial intelligence. JELIA 2004. Lecture Notes in Computer Science, vol 3229. Springer, Berlin

[7]The quantum circuits of Figs. 5.2 and 5.3 have been drawn with the help of quirk, the quantum circuit simulator by Craig Gidney (https://algassert.com/quirk).

8. Barenco A (1998) Quantum computation: an introduction. In: Lo H, Popescu S, Spiller T (eds) Introduction to quantum computation and information. World Scientific
9. Feynman R (1982) Simulating physics with computers. Int J Theor Phys 21:467–488
10. Deutsch D (1985) Quantum theory, the church-turing principle and the universal quantum computer. Proc R Soc Lond A 400:97–117
11. Green AS. The QIO package. Haskell community's central package archive of open source soft. https://hackage.haskell.org/package/QIO, v1.3
12. Bennett CH, Brassard G, Crépeau C, Jozsa R, Peres A, Wootters WK (1993) Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels. Phys Rev Lett 70:1895
13. Nielsen M, Chuang I (2010) Quantum computation and quantum information: 10th Anniversary Edition. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9780511976667
14. Greenwood GW (2001) Finding solutions to NP problems: philosophical difference between quantum and evolutionary search algorithms. Portland State University, Portland, OR
15. Gross AM, Stallard J (2007) Implementing Grover's algorithm using linear transformations in Haskell. In: Proceedings of the Eighth Symposium on Trends in Functional Programming, vol 8. p XXV
16. Deutsch D, Jozsa R (1992) Rapid solutions of problems by quantum computation. Proc R Soc Lond A 439:553
17. Simon DR (1997) On the power of quantum computation. SIAM J Comput 26(5):1474–1483
18. Kaye P, Laflamme R, Mosca M (2007) An introduction to quantum computing. Oxford University Press
19. Spivey JM (1992) The Z notation: a reference manual. Prentice Hall International
20. Saaltink M (1993) Z and EVES. Technical Report TR-91-5449-02
21. Roberts D, Nelms J, Starkey D, Thomas S (2012) Travelling by teleportation. Phys Spl Top J. University of Leicester