Superoperators for Quantum Software Engineering

Wolfgang Mauerer

Abstract As implementations of quantum computers grow in size and maturity, the question of how to program this new class of machines is attracting increasing attention in the software engineering domain. Yet, many questions from how to design expressible quantum languages augmented with formal semantics via implementing appropriate optimising compilers to abstracting details of machine properties in software systems remain challenging. Performing research at this intersection of quantum computing and software engineering requires sufficient knowledge of the physical processes underlying quantum computations, and how to model these. In this chapter, we review a superoperator-based approach to quantum dynamics, as it can provide means that are sufficiently abstract, yet concrete enough to be useful in quantum software and systems engineering, and outline how it is uses in several important applications in the field.

Key words: Quantum Computing, Software Engineering, Quantum Software Engineering, Density Operator, Superoperators, Formal Semantics

1 Introduction

The actual and hypothesised capabilities of performing computational tasks based on the laws of quantum mechanics have made the implementation of quantum computers a target of interest to physics and engineering. Yet, producing software (and algorithms) for this class of machines has by far not reached the level of productivity and ease of handling that computer science has come to expect for classical machines following decades of development.

Wolfgang Mauerer
Technical University of Applied Sciences Regensburg/Siemens AG, Germany e-mail: wolfgang.mauerer@othr.de

This is likely because at the current level of abstraction, expressing algorithms resides close to the underlying physical concepts. This necessitates strong inter-domain knowledge for researchers working in the field.

Detailed knowledge of alternative methods of describing the dynamics of quantum systems beyond applying unitary operators on finite-dimensional quantum states might not be universally spread in the software engineering community. This implies that appreciation of the usefulness of such descriptions for many open problems in quantum software engineering could be further fostered. Consequently, we provide an exposition of one particularly important such formalism—linear superoperators acting on density operators—in this chapter especially tailored towards software engineering research. We include a discussion of the possible benefits in various application areas in the domain.

Quantum circuits are the basis of many software engineering considerations, albeit at a low level of abstraction. Any typical introduction to quantum computing for computer scientists includes a discussion of circuits for the foundational set of algorithms like the ones invented by Grover, Shor, or Deutsch. In essence, a *quantum state* $|\psi\rangle$ (we provide precise formal definitions later) propagates through a quantum circuit in three phases—initialisation, application of a sequence of quantum operations, and a measurement delivering stochastic results—that constitute a quantum program. The first two actions are described by so-called unitary operators U that capture possibilities (and limitations) of quantum operations, and exhibit peculiar properties that engineers are not accustomed to from classical programming. Keeping in mind that measurements, which are realised by other means than unitary transformations, are an important ingredient of quantum algorithms, the core part of any such algorithm can nonetheless be expressed as a unitary transformation of an appropriately initialised quantum state $|\psi\rangle$ by $|\psi\rangle \mapsto U|\psi\rangle = |\psi'\rangle$.

However, quantum circuits (or any other equivalent representation of operations on quantum states) do, for most known algorithms, only capture part of the overall computational sequence. Any required classical operations or the implementation of control flow are usually described (and handled) separately from the manipulation of purely quantum mechanical states. Variational algorithms that underlie many considerations of the current era of noisy, intermediate-scale quantum (NISQ) machines rely inherently on interleaved classical and quantum operations, and explicitly operate on quantum and classical data. Consequently, it is helpful to consider a mathematical formalism that can capture all these aspects in a unified description. Likewise, the unavoidable effects of noise and imperfections that exercise probabilistic influence on a quantum state, and thus directly concern any real-world analysis of algorithmic properties, must be taken into account. A good mathematical framework for this purpose is the *density operator* formalism, which generalises quantum states $|\phi\rangle$ into density operators ϱ that can describe both, quantum and classical aspects of a computational state. Instead of unitary operators U that act on states and describe operations, *superoperators* Λ map density operators to density operators (*i.e.*, $\varrho \mapsto \Lambda(\varrho) = \varrho'$), and therefore generalise unitary quantum operations.[1]

In this chapter, we present an exposition of these concepts that targets the needs of software engineers working on the relatively new field of quantum software engineering, which necessitates to gain an understanding of quantum programming languages and possible approaches to equip them with formal semantics, or produce software with correctness by construction approaches. While we aim at using enough mathematical formalism to arrive at a precise and unambiguous presentation, we avoid the use of advanced mathematics, especially category theory, that is commonplace in research work on quantum programming language semantics, yet may act as an impediment to obtaining a higher-level view of the issues from a software engineering perspective.

This chapter nonetheless relies on some amount of formalism, and the particular mathematical topics might not be present in every computer science curriculum (and even if, it might have been a while since the reader had to deal with these topics!). For those who are curious to hear the software engineering essentials short and crisp: If you trust us that

- a density operator ϱ can, compared to ket representations $|\psi\rangle$, describe quantum states that suffer from imperfections and intricacies of the real world;
- superoperators extend the role of quantum gates to this scenario;
- the Kraus representation allows us to describe such operators in a form that is particularly convenient for computer science and software engineering purposes,

then you can skip directly to Sec. 5 that shows some of the most important software-centric applications for superoperators: Formal semantics for and verification of quantum programs, communicating and distributed quantum systems, and dealing with imperfections in real-world NISQ machines.

1.1 Challenges in Quantum Software

Following the recent review Garhwal, Ghorani, Ahmad 2021 and the textbook Ying, Zhou, Li 2019, currently established programming languages follow an either imperative (*e.g.*, QCL Ömer 2002, Silq Bichsel et al. 2020, or Q# K Svore et al. 2018) or functional (*e.g.*, QPL Selinger 2004, Quipper Green et al. 2013, or LIQUiD Wecker, KM Svore 2014). While they differ in their capabilities and degrees of abstraction, their quantum features centre around generating quantum circuits that eventually apply lists of operators

[1] The mathematical formalism of superoperators can handle more general operators than density operators, and we will see later how that can benefit software engineering when trace-decreasing operations come into play.

on quantum states. The same observations can be made for commercial approaches (*e.g.*, Cirq, Ocean, or Qiskit) to the quantum programming problem where instructions on how to generate quantum circuits are embedded into a host language, typically Python.

While software engineering research has established a multitude of methods, techniques and processes aimed at systematically constructing high-quality software artefacts, some of the elementary developmental options like debugging, tracing and some variants of testing are not directly applicable in a meaningful way in the quantum domain. Given the resulting reduction of engineering options, ascertaining the quality (or: correctness) of quantum programs must focus more on other methods like formal verification. This, in turn, requires means of properly formalising quantum programs. Starting with the seminal work Selinger 2004, research on numerous approaches of equipping quantum programming languages with formal semantics, based on which verification efforts can take place, have been conducted. Yet, the current state of the art is still lagging behind the classical level of maturity.

While it should not be required to educate all software engineers working on quantum computing on the bells and whistles of quantum physics, a reasonable awareness of the underlying principles, methods and formalisms is important for determining effective layers of abstraction. The situation is not unlike at the advent of software engineering as a discipline: Then, the need for using structured engineering approaches to construct software became apparent, yet low-level details remained crucial—as embodied, for instance, by research topics like the construction of efficient compilers for expressive high-level languages that nonetheless catered towards the very distinct hardware properties of then-current systems. We believe the superoperator-based view of quantum computing is an apt starting point for deriving such sound, practical and useful abstractions using established methods of computer science for quantum programming and quantum software engineering.

2 Mathematical Foundations

In this and the following section, we introduce the necessary formalities to understand the superoperator-based view of quantum dynamics. We assume knowledge of the standard computer science curriculum of linear algebra, but try to give an otherwise self-contained exposition. All mathematical statements and facts that we refer to without providing an explicit rationale or proof sketch are part of the standard literature on quantum computing and quantum information theory, for instance Nielsen, Chuang 2010; Vedral 2006; Ekert, Hosgold 2022; Preskill 2015, to which we refer readers interested in a more in-depth formal treatment or explicit proofs.

2.1 The Need for Formalisation

A quantum program enacts a transformation of a state (in the sense of computer science) comprised of quantum and classical input data to quantum and classical output data. While it is not possible to perform intermediate measurements on the quantum part of the state without influencing the state itself, this state nonetheless exists uniquely during the whole computation: After preparing required initial states based on classical input data, in each step of performing the computational sequence, and before performing any measurements. The later, finally, reduce (parts of) the quantum data to classical information, usually in a stochastic way. A quantum program can, depending on the input data, lead to many possible intermediate states, and likewise to many possible different outputs, even when perfect machines are assumed for execution. This is similar to stochastic algorithms that find ample use in classical software, and mathematical frameworks that allow us to model such scenarios have been established.

Consequently, we need to deal with three different entities: The state of a system, a quantum program that acts on this state, and a transformation between the quantum program (specified in whatever programming language) and an appropriate collection of formal operations that represent the state transformations, and most importantly allow us to reason about properties of the quantum program with established and new methods. While the scenario in general is very similar to the approaches used in programming language semantics, a crucial difference is that the state is not open to direct inspection by observers, and is described by different mathematical objects than for classical computation. Also, the admissible transformations between states differ radically from classical approaches.

From a physical standpoint, there are different ways of viewing this scenario: A quantum program can be translated (and most of the contemporary compilers follow this approach) into a sequence of gates that are applied to a quantum state (together with a suitable formalisation of the classical state), which can be expressed as the element of a Hilbert space. However, this formalism only applies to *perfect* underlying quantum computers that faithfully execute each gate, and are able to prepare initial quantum states that exactly represent the desired form without any stochastic uncertainties.

When machines are subject to noise and imperfections, gate operations, state preparation, readout etc. are affected by uncontrolled influences that introduce stochasticity into the quantum state; straightforward kets cannot represent the arising statistical mixture of quantum states that requires an additional characterisation of the associated classical distribution to be included in their description. This is however possible in the density operator formulation of quantum mechanics. As we have seen before, superoperators extend the role of perfect gates in the bra-ket picture. We believe this formal representation is well suited to augmenting a quantum program with formal semantics, as it can capture a wider range of phenomena that cannot be ignored at the current state of hardware development, and will with some likelihood also be of interest in the long run. While the picture is solidly established in physics-centric research, this does not universally hold for computer science and software engineering. The aim of this chapter is to provide an introduction to the formalism tailored to the particular needs of software engineering researchers.

First, let us fix some notation conventions: A quantum register $|R\rangle$ is an element of a Hilbert space $\mathcal{H}$. This space is, in the finite-dimensional case relevant for quantum computing, a complex vector space with an inner product, which implies the existence of an orthonormal standard basis $\{|i\rangle\}$ (the infinite-dimensional case requires more care, but is only very rarely relevant for the computer science aspects of quantum computing). Operations on quantum registers are carried out using unitary (linear) operators U (satisfying $UU^\dagger = U^\dagger U = \mathbb{1}$, where the dagger operation $\dagger$ denotes taking the adjoint of a linear operator). $\langle R|$ is the co-vector from the dual space of $\mathcal{H}$ associated with $|R\rangle$. The inner product between two quantum registers is denoted by $\langle R_1 \mid R_2\rangle$; recall that it satisfies, despite the somewhat different notation compared to inner products on vector spaces, (a) conjugate symmetry $\langle x \mid y\rangle = \overline{\langle y \mid x\rangle}$, (b) linearity in the second argument $\langle x| (\alpha |y_1\rangle + \beta |y_2\rangle) = \alpha \langle x \mid y_1\rangle + \beta \langle y_2\rangle$[2], and (c) positive definiteness $\langle x \mid x\rangle > 0$. Since quantum states are normalised, the latter condition effectively reads $\langle x \mid x\rangle = 1$.

2.2 Linear and Hilbert-Schmid Operators

The notion of linearity is well established in physics and computer science, and linear maps find use in many domains. The concept of linearity can of course be easily applied to operators; for the sake of completeness, let us recall the exact definition of a *linear* operator on normed spaces, as it is the formal backbone of our considerations:

Definition 1 (Linear operator) A *linear operator* T from a normed space X to another normed space Y is a linear map from $D(T) \subseteq X$ (the *domain* of T) to Y with the following property for $x, y \in D(T)$, $\alpha, \beta \in \mathbb{K}$, where $\mathbb{K}$ is an unspecified field:

$$T(\alpha x + \beta y) = \alpha T(x) + \beta T(y). \tag{1}$$

A particularly important class of operators in quantum computing (including, most importantly, density and unitary operators) is *bounded* as by the following definition:

[2] The standard scalar product on vector spaces requires linearity in the first argument.

Definition 2 (Bounded operator) An operator is called *bounded* if $\exists C \geq 0, C \in \mathbb{R}$ such that

$$||Tx|| \leq C \cdot ||x|| \tag{2}$$

for all $x \in D(T)$.

We write $\mathcal{B}(\cdot)$ to denote the set of all bounded operators acting on an underlying space. Operators that map Hilbert spaces to Hilbert spaces are crucial for our considerations:

Definition 3 (Hilbert-Schmidt operator) Let X, Y be Hilbert spaces. An operator $K \in \mathcal{B}(X, Y)$ is called *Hilbert-Schmidt operator* if there exists an orthonormal basis $\{e_\alpha : \alpha \in A\}$ (where A is some index set) with $\sum_{\alpha \in A} ||Ke_\alpha||^2 < \infty$.

Using the trace of an operator M given by $\operatorname{tr} M = \sum_i \langle i| M |i\rangle$ for an orthonormal basis $\{|i\rangle\}$ of the Hilbert space $\mathcal{H}$, we can also express the latter condition in the above definition by $\operatorname{tr} K^\dagger K < \infty$, which is obviously fulfilled if $K \in \mathcal{B}(\mathcal{H})$ and $\dim(\mathcal{H}) < \infty$, and therefore for the finite-dimensional Hilbert spaces relevant for quantum computing.

Theorem 1 (Hilbert space of Hilbert-Schmidt operators) *For Hilbert-Schmidt operators K, L of a Hilbert space X to a Hilbert space Y, $||\cdot||_{HS}$ is a norm on this space induced by the scalar product*

$$\langle K, L\rangle_{HS} := \sum_\alpha \langle Ke_\alpha, Le_\alpha\rangle . \tag{3}$$

In the quantum computing literature (and more general expositions from quantum physics), this is usually expressed by using the trace operation:

$$\langle K, L\rangle_{HS} = \operatorname{tr} K^\dagger L. \tag{4}$$

Proof If K is a Hilbert-Schmidt operator, aK is a Hilbert-Schmidt operator as well for every $a \in \mathbb{K}$. If K, L are HS operators, then for every orthonormal basis $\{e_\alpha\}$, it holds that

$$\sum_\alpha ||(K+L)e_\alpha||^2 \leq 2 \cdot \sum_\alpha \left(||Ke_\alpha||^2 + ||Le_\alpha||^2\right) < \infty, \tag{5}$$

which makes $K+L$ a Hilbert-Schmidt operator. By $\langle\cdot,\cdot\rangle$, we denote the scalar product in the space of Hilbert-Schmidt operators, and $||K||_{\text{HS}} = \langle K, K\rangle_{\text{HS}}^{1/2}$ (of course, the scalar product induces a metric). $\square$

A comparison of Hilbert spaces for quantum states and Hilbert spaces with a Hilbert-Schmidt operator basis that extend and generalise this concept is given in Table 1. The similarities between the two constructions that may seem very different at a first glance are particularly obvious when viewed in direct comparison.

Table 1: A comparison between standard Hilbert spaces used for quantum states, and Hilbert spaces based on Hilbert-Schmidt operators.

Entity	Hilbert space	Hilbert space of Hilbert-Schmidt operators
State	$\lvert f\rangle \in \mathcal{H}$	$\hat{D} : \mathcal{H} \to \mathcal{H}$
Operator	$\mathcal{H}\to\mathcal{H}$: $\hat{D}\lvert x\rangle = \lvert x'\rangle$	$\Lambda : \hat{D}\to\hat{D} \equiv (\mathcal{H}\to\mathcal{H}) \to (\mathcal{H}\to\mathcal{H})$
Norm	$\lvert\lvert f\rangle\rvert = \sqrt{\langle f\vert f\rangle}$	$\lVert\hat{D}\rVert_{\mathrm{HS}} = \sqrt{\operatorname{tr} D^{\dagger} D}$
Operator norm[a]	$\lVert\hat{D}\rVert = \sup\limits_{\substack{\lvert f\rangle\in\mathcal{H}\\ \lVert\lvert f\rangle\rVert\leq 1}} \lvert\hat{D}\lvert f\rangle\rvert$	$\lvert\lvert\Lambda\rvert\rvert = \sup\limits_{\substack{\hat{D}\in\mathcal{H}\\ \lVert\hat{D}\rVert\leq 1}} \Lambda(\hat{D}) = \sup\limits_{\substack{\hat{D}\in\mathcal{H}\\ \lVert\hat{D}\rVert\leq 1}} \operatorname{tr}\Lambda(\hat{D})^{\dagger}\Lambda(\hat{D})$

[a] Other choices for the Hilbert space norm that fulfil the required properties are possible.

3 Modelling Hybrid Quantum-Classical Systems

Having laid out the mathematical preliminaries, we commence with discussing how to apply the formalism to model hybrid quantum-classical systems, as they form the basis of essentially all known quantum algorithms.

3.1 States and effects

Ideally, an experiment resulting in a probability distribution can be carried out by repeating the following two processes until a a meaningful level of statistical significance is reached?

- *Preparation* of a (quantum mechanical) state according to some fixed procedure that can be repeated a sufficient number of times.
- *Measurement* of some *observable* quantity (*e.g.*, spin, energy, . . .). *Effects* are a special class of measurements that can result in either the answer 'yes' or 'no' according to some probability distribution.

It is important to note that quantum measurements do not correspond to a passive acquisition of information that is common in classical computing. While it is a physical process, it is described by a different set of mathematical tools in the standard formalism of quantum computing based on states and operators. This unsatisfactory difference can be mostly mended by the use of superoperators.

Since quantum computing does not only deal with pure quantum states (and, at least in the NISQ era, statistical mixtures), but needs to handle classical and quantum data, the formalism must be able to account for such settings. Resulting systems are usually termed *hybrid* systems). It is obvious that any measurement results obtained from quantum systems fall into the classical category since measurement gauges that materialise in the macroscopic world are used to infer them from the quantum system, whatever their exact mechanism of performing the measurement is; this requires to provide mechanisms that reduce quantum to classical data.[3]

Every quantum system can be completely characterised by its observable quantities which in turn are characterised by self-adjoint operators. These operators form an algebra $\mathcal{A}$; since we do only deal with finite-dimensional Hilbert spaces here, we can restrict ourselves to sub-algebras of $\mathcal{B}(\mathcal{H})$ (*i.e.*, $\mathcal{A} \subset \mathcal{B}(\mathcal{H})$). $\mathcal{A}$ is called the *observable algebra* of the system and is often identified with the system itself because it is possible to deduce all properties of the system from its observable algebra. The *dual algebra* of $\mathcal{A}$ is denoted by $\mathcal{A}^*$ and is the algebra defined on the dual space.

To capture the notions of *state* and *effect* mathematically, two sets ($\mathcal{S}$ representing all states, and $\mathcal{E}$ containing all effects) are defined as follows:

$$\mathcal{S}(\mathcal{A}) = \{\, \varrho \in \mathcal{A}^* \mid \varrho \geq 0 \wedge \varrho(\mathbb{1}) = 1 \,\}\,, \tag{6}$$

$$\mathcal{E}(\mathcal{A}) = \{\, A \in \mathcal{A} \mid A \geq 0 \wedge A \leq \mathbb{1} \,\}\,. \tag{7}$$

For every tuple $(\varrho, A) \in \mathcal{S} \times \mathcal{E}$, there exists a map $(\varrho, A) \rightarrow \varrho(A) \in [0, 1]$ which gives the probability $p = \varrho(A)$ that measuring an effect A on a (system prepared in the) state ϱ results in the answer 'yes'. Accordingly, the probability for the answer 'no' is given by $1 - p$. $\varrho(A)$ is called the *expectation value* of an effect A; states are thus defined as expectation value functionals from an abstract point of view. These expectation value functionals can be uniquely connected with a normalised trace-class operator (for which the the value of the trace operation is independent of the basis chosen to evaluate the trace) ϱ such that $\varrho(A) = \mathrm{tr}(\varrho A)$. In principle, it would be necessary to introduce two different symbols for the expectation value functional and the operator, but for simplicity, we omit this complication.

We need to distinguish between two different kinds of states: *Pure* and *mixed* ones. This is a consequence of both $\mathcal{S}$ and $\mathcal{E}$ being convex spaces: For two states $\varrho_1, \varrho_2 \in \mathcal{S}(\mathcal{A})$ and $\lambda \in \mathbb{R}, 0 \leq \lambda \leq 1$, the convex combination $\lambda\varrho_1 + (1 - \lambda)\varrho_2$ is also an element of $\mathcal{S}(\mathcal{A})$. The same statement holds for the elements of $\mathcal{E}(\mathcal{A})$. This decomposition provides a nice insight into the structure of both spaces: Extremal points cannot be written as a proper

[3] The problem of how measurements of a quantum system are to be interpreted (or even how the whole process can be described consistently) has been and still is one of the fundamental philosophical problems of quantum mechanics Auletta, Fortunato, Parisi 2009. Fortunately, choosing an interpretation (or answering the question if an interpretation is necessary at all) is not relevant for any of the formalisms discussed in this chapter.

convex decomposition, that is, for $x = \lambda y + (1-\lambda)z$ it follows that either $\lambda = 1$, or $\lambda = 0$, or $x = y = z$. They can be interpreted as follows:

- For $\mathcal{S}(\mathcal{A})$, extremal points are *pure states* with no associated classical uncertainty.
- For $\mathcal{E}(\mathcal{A})$, extremal points describe measurements which do not allow any fuzziness as is, for instance, introduced by a detector which detects some property not with certainty, but only up to some finite error (alas, this applied to all real-world detectors used in NISQ machines to read the result of a computation).

It can be shown that the density matrix $\varrho = |\phi\rangle\langle\phi|$ of pure states fulfils the property $\text{tr}(\varrho^2) = 1$, whereas for mixed states, $\text{tr}(\varrho^2) < 1$. Consequently, it is possible to distinguish between pure and mixed states when a physical tomography of the resulting state (or any intermediate state of a computation) is available. While this is not within the usual functionalities offered by NISQ machines, it can be implemented with some effort, and it is important to know from a software point of view (especially in terms of result reliability and quality) that the approach is available.

3.2 Observables

Until now, we have only considered effects, that is, measurements resulting in a binary answer that is either 'yes' or 'no'. We also need to cover measurements with a more complicated result range; this is necessary to describe general *observables*. Although we would have to consider an infinite (even uncountable) number of possible outcomes for a general description of quantum mechanics, it is sufficient to consider only observables with a finite range for the purposes of quantum computing.[4] Such observables are represented by maps which connect elements x of a finite set R to some effect $E_x \in \mathcal{E}(\mathcal{A})$; this in turn gives rise to a probability distribution $p_x = \varrho(E_x)$. More formally, we can put it as in the following:

Definition 4 (Positive Operator-Valued Measurement) A family $E = \{E_x\}, x \in R$ of effects $E_x \in \mathcal{A}$ is called a *positive operator valued measurement* (POVM) on R if $\sum_{x \in R} E_x = \mathbb{1}$.

Note that the E_x need *not* necessarily be projectors, that is, they must *not* necessarily satisfy the identity $E_x^2 = E_x$. Should this be the case for all

[4] This is justified because quantum computers process states of the type $(|0\rangle, |1\rangle)^{\otimes n}$. Although quantum computers can possess an arbitrary number of qubits, it is still a fixed and (which is most important) finite number; additionally, we do not care for any continuous quantum properties of these objects. Notice that special types of computations like analogue quantum simulation of molecules of chemical compounds that are seen as possible use-cases for quantum computers are not included in the framework discussed here.

x, the measurement is called a *projective measurement*, which is the type measurement used in most canonical quantum algorithms and variational approaches when a projection onto the binary basis is performed.

Observables of this kind can be described by self-adjoint operators of the underlying Hilbert space $\mathcal{H}$ which can be seen as follows: Every self-adjoint operator A on a Hilbert space $\mathcal{H}$ of finite dimension can (owing to the spectral theorem for normal matrices) be decomposed as $A = \sum_{\lambda \in \sigma(A)} \lambda P_\lambda$. Here $\sigma(A)$ denotes the spectrum of A, while P_λ provide projectors onto the corresponding eigenspace. The expectation value $\sum_\lambda \lambda \varrho(P_\lambda)$ of A for a given state ϱ can equivalently be calculated by $\varrho(A) = \mathrm{tr}(\varrho A)$. Since this is the standard way of formulating the expectation value of an operator, both points of view coincide.

3.3 Classical components

Systems consisting solely of quantum components are generally not to be found: At the latest after a measurement has been performed, classical probabilities need to be accounted for. Therefore, we need to pay attention to hybrid systems composed from quantum and classical parts as well. Obviously, we have to orient ourselves along the lines of Section 3.1 to provide proper grounding for both possibilities. Consider a finite set X of elementary events, that is, all possible outcomes of an experiment. Again, $\mathcal{S}(\mathcal{A})$ and $\mathcal{E}(\mathcal{A})$ define the set of states and effects, respectively, but this time, the observable algebra is given by all complex valued functions from the set X to $\mathbb{C}$ as defined by

$$\mathcal{A} = \mathcal{C}(X) = \{ f : X \to \mathbb{C} \} . \tag{8}$$

By identifying the function f with the operator $\hat{f}$ given by

$$\hat{f} = \sum_{x \in X} f_x \left| x \right\rangle \left\langle x \right| \tag{9}$$

where $|x\rangle$ denotes a fixed orthonormal basis, the probability distribution can be interpreted as an operator algebra similar to the quantum mechanical case because $\hat{f}$ is an element of $\mathcal{B}(\mathcal{H})$. Thus, $\mathcal{C}(X)$ can be used as an observable algebra $\mathcal{A}$ along any other quantum mechanical or classical constituent of a multi-partite composite system.

3.4 Composite and hybrid systems

Since quantum mechanical and classical systems can be described with very similar structures, the presented formalism is well suited for the presentation of composite systems, as becomes necessary when quantum computations are subjected to a classical control flow, or when hybrid quantum-classical calculations are performed, as is the case for variational algorithms. Let $\mathcal{A} \subset \mathcal{B}(\mathcal{H})$ and $\mathcal{A}' \subset \mathcal{B}(\mathcal{K})$ be systems given in terms of their observable algebras; the composite system is then given by

$$\mathcal{A} \otimes \mathcal{A}' \equiv \operatorname{span}\{\, A \otimes B \mid A \in \mathcal{A}, B \in \mathcal{A}' \,\}\,. \tag{10}$$

Three cases for the choice of $\mathcal{H}, \mathcal{K}$ can be distinguished:

- If both systems are quantum, then $\mathcal{A} \otimes \mathcal{A}' = \mathcal{B}(\mathcal{H} \otimes \mathcal{K})$.
- If both systems are classical, then $\mathcal{A} \otimes \mathcal{A}' = \mathcal{C}(X \times Y)$ with $\mathcal{C}$ as defined by Eq. (8)
- If $\mathcal{A}$ is classical and $\mathcal{A}'$ is quantum mechanical, we have a *hybrid* system; the composite observable algebra is then given by $\mathcal{C}(X) \otimes \mathcal{B}(\mathcal{H})$ which cannot be simplified any further. Observables are operator-valued functions in this case, as expected.

4 Completely Positive Maps and their Representation

In quantum mechanics, time evolution is described by transformations of density matrices with an operator Λ that is called a *superoperator*.[5] Before we can proceed to formally define superoperators, let us fix some terminology: An operator O acting on a Hilbert space is positive definite if $\langle\psi| O |\psi\rangle > 0$ for all elements $|\psi\rangle$ of the Hilbert space, and positive semi-definite if $\langle\psi| O |\psi\rangle$ is non-negative. Physical density operators are Hermitian and positive semidefinite, which implies they have real non-negative eigenvalues. A positive map Λ transforms positive operators into positive operators. If $\Lambda \otimes \mathbb{1}$ is semidefinite positive ($\forall n \in \mathbb{N} : \Lambda \otimes \mathbb{1}_n \geq 0$), then Λ is called a completely positive map.

Definition 5 (Superoperator) A superoperator $\Lambda : \mathcal{B}(\mathcal{H}) \to \mathcal{B}(\mathcal{H})$ has the following properties for all density operators ϱ with $\varrho' = \Lambda(\varrho)$:

1. Λ is linear.
2. If $\varrho^\dagger = \varrho$, then $\varrho'^\dagger = \varrho'$ (Hermiticity preservation).
3. If $\operatorname{tr} \varrho = 1$, then $\operatorname{tr} \varrho' = 1$ (trace preservation).
4. Λ is a completely positive map.

[5] The Schrödinger equation $i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle = H|\psi(t)\rangle$ governs, given a Hamilton operator H (whose meaning is extensively discussed in the quantum software engineering chapter of this book) the time evolution of a closed quantum system. The Liouville–von Neumann equation $i\hbar\frac{\partial\varrho}{\partial t} = [H, \varrho] = H\varrho - \varrho H$ generalises the Schrödinger equation to density operators. For a time-independent system (which we take as a simple illustration, albeit the consideration would also apply to time-dependent interactions), the density operator at time t, $\varrho(t)$, can be obtained as $\varrho(t) = \exp(-iHt/\hbar)\varrho(t=0)\exp(iHt/\hbar)$, which is nothing else than a mapping $\varrho(t=0) \mapsto \varrho(t) = \Lambda(\varrho)$ using a superoperator Λ.

Superoperators share the convenient property of linearity with many other objects in computer science. Since physical density operators are Hermitian, the preservation of Hermiticity in property (2) means this important characteristic of a density operator is not changed by any superoperator. Especially, it implies that eigenvalues of the operator remain real-valued after transformations. Property (3) means that statistical mixtures of quantum states are mapped to other valid statistical mixtures of quantum states, and we cannot produce 'invalid' objects by executing transformations described by superoperators. Note that if dissipative processes are considered, the second condition must be loosened to $\mathrm{tr}(\varrho') \leq 1$; we will see later that relaxing this physically motivated condition is reasonable for the computer science domain. Finally, property (4) is of physical importance: Λ is not only positive semidefinite (*i.e.*, ϱ' is non-negative if ϱ is non-negative) on $\mathcal{H}_A$, but also on any possible extension $\mathcal{H}_A \otimes \mathcal{H}_B$. This ensures that Λ maps a density operator to another valid density operator even when the system under consideration is entangled with some outside entity.

4.1 Operator-Sum Representation

Kraus 1983 provides a seminal result about the decomposability of completely positive maps that allows us to specify concrete, operational representations for superoperators:

Theorem 2 (Kraus representation theorem) *A superoperator Λ as defined in Def. 5 can be written as a partition of $\mathbb{1} = \sum_{k=1}^{N} A_k^\dagger A_k$ where A_k are linear operators acting on the Hilbert space of the system such that*

$$\varrho' = \Lambda(\varrho) = \sum_{k=1}^{N} A_k \varrho A_k^\dagger \qquad (11)$$

for any density matrix ϱ that represents a mixed or a pure state.[6]

This representation is also known under the illustrative name *operator-sum representation.*

A unitary operator U that is applied to a (possibly mixed) density operator ϱ is a Kraus representation with a single element ($k = 1$) for the underlying transformation, as $\varrho' = U\varrho U^\dagger$, and $U^\dagger U = \mathbb{1}$. Superoperators, in that sense,

[6] Notice that while Kraus published his representation theorem relatively late compared to the advent of quantum mechanics, and coincidentally around the time when Feynman first considered the computational power of quantum mechanics, the concept of density operators goes back much further in history. Both, concept and representation, find widespread use outside quantum computing in the dynamical description of general dissipative systems.

generalise unitary transformations as they allow for expressing more complex transformations than can be provided by $\varrho \mapsto U\varrho U^\dagger$.

To further illustrate the Kraus representation, consider the situation that the system is in contact with a (larger) environment, which is a common situation not only for more general physical experiments, but especially for quantum computers: The processing unit (QPU) where quantum effects take place is surrounded by multiple levels of cooling, the laboratory room in an experimental facility (or a data centre), and ultimately, the rest of the universe. All of these can perturb and therefore influence the QPU, which must be shielded from the influence of this environment. Even setting aside engineering issues, a perfect shielding that eliminates the influence of the environment is impossible to achieve, as this would make it impossible to prepare initial states, apply transformations on them, and read out the result.

If the environment is modelled sufficiently large, both systems form a closed quantum system. Transformations in the combined system can be described by a unitary transformation $U \in U(\dim(\mathcal{H}) \cdot \dim(\mathcal{H}_{\text{env}}))$ where $\mathcal{H}$ denotes the Hilbert space of the system under consideration and $\mathcal{H}_{\text{env}}$ the Hilbert space of the environment. Assume that the environment is in a pure state $|e_0\rangle\langle e_0|$.[7] The density operator of the system under consideration *after* the unitary operation was applied to the total system can be recovered by tracing out the environment:

$$\begin{aligned}
\varrho' = \Lambda(\varrho) &= \text{tr}_{\text{env}}(U\varrho \otimes |e_0\rangle\langle e_0| U^\dagger) && (12)\\
&= \sum_k \langle e_k| U(\varrho \otimes |e_0\rangle\langle e_0|)U^\dagger |e_k\rangle && (13)\\
&= \sum_k \langle e_k| U |e_0\rangle \varrho \langle e_0| U^\dagger |e_k\rangle && (14)\\
&= \sum_k A_k \varrho A_k^\dagger. && (15)
\end{aligned}$$

In the last step, we define A_k by $A_k \equiv \langle e_k| U |e_0\rangle$. A set of Kraus operators $\{A_k\}$ *implements* a completely positive Λ if $\forall \varrho \in \mathcal{D} : \sum_k A_k \varrho A_k^\dagger = \Lambda(\varrho)$.

Theorem 3 *The operation elements of a given superoperator Λ are not unique: If $\{E_j\}$ is a set of Kraus operators, then a different set of Kraus operators $\{F_k\}$ describes the same operation if and only if there exists a unitary matrix $U \in U(n)$ with $n = \text{card}(\{E_k\})$ (where $\text{card}(X)$ is the cardinality of the set X) such that*

$$F_k = \sum_j U_{kj} E_j. \tag{16}$$

[7] This assumption holds without loss of generality because it can be shown that a system can be purified by introducing extra dimensions which do not have any physical consequences.

Note that the shorter set may be padded with zero elements until the cardinality of both matches.

Let $\{A_k\}$ be a set of Kraus operators that represents the cp-map Λ. Note that if any number of elements A_i is taken from $\{A_k\}$, the set still remains a completely positive map, but is not trace preserving any more.

Superoperators are elements of $\mathcal{B}(\mathcal{H})$, which makes it possible to apply many theorems of linear operator algebra to superoperators. As we have seen above superoperators can themselves be used as elements of a Hilbert space, which implies that from a structural point of view, any distinction between operators and superoperators is mathematically irrelevant. However, we believe this is an argument in favour of using superoperators to describe quantum programming languages, as insights and techniques from linear operator theory can be immediately applied. Finally, notice that the number of Kraus elements needed to express any arbitrary completely positive map $T : \mathcal{B}(\mathcal{H}_1) \to \mathcal{B}(\mathcal{H}_2)$ is bounded by $\dim(\mathcal{H}_1) \cdot \dim(\mathcal{H}_2)$.

5 Applications in Quantum Software and Systems Engineering

In this section, we provide concrete examples for the use of superoperators in problems related to software engineering, embedding a brief discussion of seminal and recent results.

5.1 Formal Semantics and Verification

Several semantic domains based on various mathematical formalisms of the underlying quantum physics have been used to provide semantics for quantum programs: Unitary operations or probabilistic functions on pure quantum states, admissible transformations Perdrix 2008, or completely positive maps on density operators, for which Selinger 2004 initiated a series of follow-up results that established connexions between the physical framework outlined in this chapter, and established approaches to (denotational) semantics in computer science, in particular based on category theory.

Following Moggi 1991, it is known that modelling classical computational effects like assignments or exceptions are possible using the category theoretical concept of monads that have received considerable attention in computer science as abstract data types in functional programming languages. Likewise, quantum computing based on states and linear operators is known to be almost a monad Mu, Bird 2001; by extending the physical model to density operators and superoperators Vizotto, Altenkirch, Sabry 2006, it is possible to formalise the computational semantics by the category theoretical construct of arrows, which generalise monads. Importantly, such approaches do not need to distinguish between computation and measurement, as the underlying superoperator formalism unifies both aspects. As stated before, establishing connexions between quantum computations (in terms of the superoperator formalism) and monads and arrows enables embeddings in current classical languages, and exposes connections to well-understood concepts from the semantics of (classical) programming languages.

Earlier seminal work Selinger 2004 defines a functional programming language that can establish various compile-time guarantees, and is equipped with a denotational semantics based on complete partial orders of superoperators: The established Löwner partial order, in which $A \sqsubseteq B$ holds if and only if $B-A$ is positive semidefinite, is slightly extended to apply on matrix tuples. The approach also offers a formal category theoretic treatment based on so-called complete partial order-enriched traced monadial categories, for which categorical operations like composition and tensor are Scott-continuous, (*i.e.*, they preserve least upper bounds of increasing sequences), which allows for using the guaranteed existence of fix-points of Scott-continuous endofunctions on pointed (*i.e.*, equipped with a least element) complete partial orders to deal with loops and recursion.

An additional recent approach, QUnity Voichick et al. 2023 provides a type system based on algebraic data types, and allows for combining (and nesting) the use of unitary transformations and superoperators. Denotational semantics are provided in the form of pure and mixed semantics, building upon unitary transformation of quantum states and superoperators applied to density operators, respectively. Finally, let us mention the review of formal verification Lewis, Soudjani, Zuliani 2023 that summarises further approaches to quantum semantics, not limited to superoperator-based constructions.

Given the multitude of existing approaches, it is interesting to observe that actual software containing quantum code Schönberger et al. 2022 and patterns for quantum software Leymann 2019 are almost exclusively expressed in languages that are not equipped with advanced formal semantics, while languages that enjoy this quality find popularity restricted to within academic circles. This leaves important gaps to be filled, given that it is textbook knowledge in software engineering how software quality and reliability of systems can be considerably improved by formal verification, static analysis or correctness by construction. First results along this line for the quantum domain have appeared recently (albeit also based, as is customary in this line of research, on toy languages that expose only the most salient features without syntactic sugar) Peduri, Schaefer, Walter 2023; Zhou et al. 2023; interestingly, both approaches revolve around denotational semantics Scott, Strachey 1971 for a 'while' language that inductively constructs a superoperator $[\![C]\!] : \mathcal{B}(\mathcal{H}) \to \mathcal{B}(\mathcal{H})$ as denotation of program (fragment) C. To convey the flavour of how the approaches relate to superoperators, consider the following (incomplete) fragment of a language similarly to what is used by Peduri *et al.* and Zhou *et al.*:

$$\begin{aligned} S \coloneqq\ & \textbf{skip} \mid \textbf{abort} \mid \vec{q} := \hat{U}(\vec{q}) \mid S_1; S_2 \mid \textbf{repeat}\ N\ \textbf{do}\ S\ \textbf{end} \mid \\ & \textbf{while meas}\ \vec{q}\ \textbf{with}\ B\ \textbf{do}\ S\ \textbf{end} \end{aligned}$$

Here, S_i denote quantum program fragments obtained from the production S, $\vec{q}$ allows us to select quantum bits from the overall quantum register, and $\hat{U}$ is a unitary operator, as usual. An application of the operator on a subset of the available quantum bits is given by $\hat{U}(\vec{q})$. A denotational semantics, again similar to the variants used in the cited approaches, can be defined as follows:

$$\begin{aligned}
&[\![\textbf{skip}]\!] = \mathbb{1}\text{; that is, } [\![\textbf{skip}]\!](\varrho) = \mathbb{1}\varrho\mathbb{1}^\dagger \\
&[\![\textbf{abort}]\!] = 0\text{; that is, } [\![\textbf{abort}]\!](\varrho) = 0 \\
&[\![\vec{q} := \hat{U}(\vec{q})]\!] = \hat{U}_{\vec{q}}\text{; that is, } [\![\vec{q} := \hat{U}(\vec{q})]\!](\varrho) = \hat{U}_{\vec{q}}\rho\hat{U}^\dagger_{\vec{q}} \\
&[\![S_1; S_2]\!] = [\![S_2]\!] \circ [\![S_1]\!] \\
&[\![\textbf{repeat}\ N\ \textbf{do}\ S\ \textbf{end}]\!] = \underbrace{[\![S]\!] \circ [\![S]\!] \circ \cdots \circ [\![S]\!]}_{N \text{ times}} \\
&[\![\textbf{while meas}\ \vec{q}\ \textbf{with}\ B\ \textbf{do}\ S\ \textbf{end}]\!] = \textstyle\sum_{k=0}^{\infty} \left(\mathcal{B}_{0,\vec{q}} \circ ([\![S]\!] \circ \mathcal{B}_{1,\vec{q}})^k\right)
\end{aligned}$$

The definitions for the skip statement and unitary operator application on (a list of) quantum states $\vec{q}$ defined in the first two lines make it clear that the denotations for the fragments are superoperators that can be applied on concrete density matrices ϱ; however, a density matrix is not required to define the actual denotation. The while statement uses two binary projective measurement operations that can also be represented by superoperators that define the transformation: $\mathcal{B}_i(\varrho) = B_i\varrho B_i^\dagger$ for $B_i = |i\rangle\langle i|$ in the computational standard basis.

Multiple approaches can establish that the while statement is well-defined; Peduri *et al.* base their consideration on an increasing sequence (in terms of the Löwner partial order as defined above) of density operators obtained for termination within an increasing number of iterations. To allow for modelling non-termination, their considerations are based on sub-normalised density operators, that is, positive semi-definite operators with trace *at most* one instead of exactly one, which is satisfied for physical density operators. It is immediately clear that the **abort** is non-trace preserving, and the **while** loop can be non-trace preserving if it does not terminate after a finite number of iterations.

However, despite recent progress and compared to the substantial body of literature on classical programming language semantics, the field is still very much in its early stages. Apt mathematical models to describe foundational semantics that attract researchers from both fields, together with a common understanding of necessities, can hopefully lead to fruitful progress in the future, possibly also eventually benefiting classical software engineering. As many of the approaches are implicitly or explicitly based on superoperators, it is not unlikely that this formalisation of quantum computing will play an important role in the further development of quantum software semantics.

5.2 Communicating and Distributed Systems

Quantum communicating systems Khatri, Wilde 2020 can be seen as an example of restricted, distributed quantum computers; while they are not intended for general-purpose computing, they share some characteristics with quantum computers in that they prepare, manipulate and measure quantum states. In contrast to NISQ machines and future fault-tolerant quantum computers, quantum communication systems have reached commercial maturity. In view of future distributed quantum computers that will also face how to distribute quantum states over spatial distances, insights into quantum communication systems can therefore benefit future quantum software engineering. Again, superoperators play a pronounced role in this domain.

Let, for example, ϱ_{AB} denote the density matrix of the state shared by Alice and Bob, the two customary virtual representative parties of distributed (communication) systems. The information available for each of them can be inferred by calculating the partial trace: $\varrho_{\mathrm{A}} = \mathrm{tr}_{\mathrm{B}}\, \varrho_{\mathrm{AB}}$ and $\varrho_{\mathrm{B}} = \mathrm{tr}_{\mathrm{A}}(\varrho_{\mathrm{AB}})$. The bipartite density matrix can never be recovered from these partial density matrices because as it is known that in general, many bipartite density matrices that give rise to the same partial density matrices. It is also obvious that a density operator representing an entangled state cannot be represented by a direct tensor product of unrelated partial density operators by the very definition of entanglement. One of the goals of denotational semantics as exemplified for the quantum case in the previous section is, essentially, to assign sufficient information to every edge of a flow graph such that the complete semantics of a program can be reconstructed by combining only the information given by the edges constituting the program. The denotation of a statement composed of several sub-statements must be completely determined only by a function of the denotations of the sub-statements.

This is impossible when transformations between explicit density matrices are considered. Since a combination of the partial density matrices ϱ_{A}, ϱ_{B} which were manipulated by Alice and Bob does not restore the total bipartite state ϱ_{AB}, a description that relies on a single density operator would obviously not comply with the physical state afterwards.

A possible solution would be to annotate the complete flow graph, that is, of both paths representing the control flow for Alice and Bob. In this case, the operations performed by Alice and Bob would be written as tensor products of the type $A \otimes \mathbb{1}_{\mathrm{B}}$ and $\mathbb{1}_{\mathrm{A}} \otimes B$ that act on the combined density matrix ϱ_{AB}. This way, we could assign semantics to the program as a whole, but would loose the ability to construct the denotation of a phrase from the denotations of its sub-phrases. This means that the semantics of the complete program could not be constructed from the denotation of Alice's and Bob's programs (each running on a separate computational entity) alone, which contrasts the key idea of denotational semantics.

Therefore, we need to seek a solution that does not characterise quantum operations by showing transformations of explicit density matrices (or provides superoperators that operate on the overall density operator), but instead captures the notion of a transformation in a more abstract sense. Completely positive maps (as explicitly represented by a set of Kraus operators) obviously fulfil this need, and we consequently deem them a good choice to describe quantum communication processes, and more general quantum computations that involve communication Mauerer 2005, or distributed computations Cirac et al. 1999. While this requires some additional care in making sure that the definition of denotations does not carry an implicit dependence on an actual density operator, this is possible with the Kraus representation of superoperators as introduced above.

5.3 Noise and Imperfection Modelling

At the time of writing, all physically available quantum computers fall into the class of noisy, intermediate-scale quantum (NISQ) machines that are not fully fault-tolerant. This means that there is a difference between the *intended* transformation of quantum states (and measurements) described by a quantum program, and the *actual* transformation performed by the hardware. While no physical obstacles prevent building perfect machines reduce error rates to arbitrary low levels by using error-correcting codes, this comes at the expense of substantial overhead in the amount of required qubits and other resources that is currently much beyond experimental reach. Therefore, any formal and semantic considerations that implicitly assume perfect quantum computers will be in disagreement with experimental and practical reality, which is counter to their crucial point as they are supposed to improve software quality and correctness, which is a strongly practical desideratum.

However, even if this problem will disappear with the advent of perfect quantum computers, there are reasons to believe that NISQ machines of sufficient quality will be able to perform advantageous computations, and it cannot be ruled out this class of machines—given that it will likely be possible to manufacture them at substantially reduced cost and effort in comparison to fault-tolerant machines with application-specific co-design techniques Wintersperger, Safi, Mauerer 2022; Safi, Wintersperger, Mauerer 2023—will be of long-term or even permanent relevance. Considering the effects of imperfections is, consequently, not only worthwhile in the NISQ area, but might also display benefits beyond, and software engineering research should consider the respective implications: We believe it is important to understand scalability, performance, quality and reliability of quantum software on NISQ machines beyond empirical measurements, as these are also core considerations for classical software executing on classical hardware.

As NISQ systems can be seen as open quantum systems, superoperators are again well suited to modelling their properties and behaviour (other techniques like Lindbladian dynamics Preskill 2015 could model such scenarios, but are outside the scope of our considerations).

Current software-centric research deals, for instance, with effectively adapting noise models to real machines Georgopoulos, Emary, Zuliani 2021 or the efficient learning of quantum noise Harper, Flammia, Wallman 2020, as characterising noise from first physical principles or even measuring the actual characteristics on machines can be computationally prohibitive. Likewise, the (non-)resilience against noise of variational algorithms like the quantum alternating operator approach (QAOA), an optimisation algorithm targeted at NISQ machines, has been studied Marshall et al. 2020; Xue et al. 2021. From the software engineering point of view, Greiwe, Krüger, Mauerer 2023 provide a didactic exposition to modelling imperfections of quantum computers with a focus on consequences for non-functional properties; we partly follow their presentation below.

To model such imperfections, consider that while the evolution of a closed quantum system is described by unitary operations, NISQ machines do not enjoy a complete isolation against their environment. Until fully error-corrected systems that mitigate this deficiency are available, the arising consequences will penetrate into the quantum software and programming language layers. A noisy system is subject to the influence of an external, uncontrolled environment that must be included in any model of the system, eventually ending up with a larger, but closed quantum system.

Similar to the illustration of the Kraus representation theorem before, ϱ denotes the open quantum system under consideration. It is combined with an uncontrolled environment ϱ_{env}, equating to a larger, closed system $\varrho \otimes \varrho_{\text{env}}$ subject to evolution $U(\varrho \otimes \varrho_{\text{env}})U^\dagger$. This *overall* evolution is described by a unitary operator U. By eliminating the uncontrolled environment using a partial trace operation, the effective (and usually non-unitary) evolution of ϱ under noise is given by $\mathcal{E}(\varrho) = \text{tr}_{\text{env}}(U(\varrho \otimes \varrho_{\text{env}})U^\dagger)$.

Let $\mathcal{B}_e = \{|e_k\rangle\}_k$ be a basis of the environment. If the environment is measured in $\mathcal{B}_e$ after the time evolution, then the outcome determines the state of the principal system. We end up with a random distribution of states for the principal system depending on the measurement. The effect the environment had on ϱ when the outcome k occurred can be described by an operator E_k, leading to a mixed state description

$$\varrho \mapsto \sum_k E_k \varrho E_k^\dagger. \tag{17}$$

This Kraus representation can be used to describe effects that occur in *imperfect*, NISQ-era quantum systems. One canonical example is a probabilistic qubit flip that randomly with probability p (*i.e.*, by the influence of external factors like energy dissipation, or the imperfect operation of quantum gates) negates a quantum bit can be described by

$$\varrho \mapsto (1-p)\mathbb{1}\varrho\mathbb{1}^\dagger + pX\varrho X^\dagger. \tag{18}$$

The Pauli X gate is a unitary operator that, in state-based notation, flips a quantum bit: $X|0\rangle = |1\rangle$, and $X|1\rangle = |0\rangle$. The operator can be applied as usual in a quantum circuit to *deterministically* negate a quantum bit. In the above formulation of Eq. (18), however, the gate is applied to the one qubit system ϱ with probability p, and else leaves the state as is. When the source of corresponding errors is unclear in a NISQ system, probability p is an effective (classical) parameter of the system whose magnitude can be determined by testing the system. By comparing the structure of Eq. (17) with the above equation, it can be seen that this delivers operators E_k. Similarly, randomly occurring phase flip errors (described in the state formulation by a Pauli Z gate), or a combination of bit and phase flip error (described in the state formulation by a Pauli Y gate) can be constructed by replacing X by Z or Y in Eq. (18). In each case, the interpretation of the the operation is that the quantum state is left intact with probability $1-p$ by applying an identity transformation, and affected by the error with probability p, resulting in a convex combination of density operators that includes *classical, stochastic* uncertainty: While the actual quantum system is in each of possibly multiple computational runs either in state ϱ (when no error occurred) *or* state $X\varrho X$ (in case an error occurred), an observer does not know *if* a stochastic error occurred, and must therefore include this lack of knowledge in the description of the quantum state. Note that while the initial state may be a pure state that does not contain any lack of knowledge before the operation induced by Eq. (18) is performed, it is also possible that a density operator already featuring classical lack of knowledge enters the quantum operation, which then in turn (usually) increases the lack of knowledge even further.

Generalising from the binary error model use for bit, phase and phase-bit flips, the formalism also allows us to model more complex imperfections as they occur in realistic systems, for instance with the commonly employed completely depolarising operator: One qubit is randomly subjected to one of the Pauli operators X, Y, Z by

$$\begin{aligned}\varrho \;\mapsto\; & (1-p)\mathbb{1}\varrho\mathbb{1}^\dagger + \\ & p\frac{1}{4}\left(\mathbb{1}\varrho\mathbb{1}^\dagger + X\varrho X^\dagger + Y\varrho Y^\dagger + Z\varrho Z^\dagger\right),\end{aligned} \tag{19}$$

with a certain probability, and else leaves the qubit as is. A quick calculation reveals that (19) equals $\varrho \mapsto (1-p)\varrho + p\frac{1}{2}\mathbb{1}$, where $\frac{1}{2}\mathbb{1}$ is the density representing the state of a system being in every basis state with equal probability. Hence, the system either stays intact or all information gets destroyed with probability p. For an n qubit system, we obtain

$$\varrho \mapsto (1-p)\varrho + p\frac{1}{2^n}\mathbb{1} \tag{20}$$

following a textbook calculation.

6 Summary and Conclusion

Superoperators provide a rigorous mathematical representation of quantum operations that go beyond unitary transformations, as they allow us to model measurements and imperfections. In this chapter, we have provided an introductory exposition to the concept tailored towards the domain of software engineering, and have elaborated on existing and possible use-cases for the concept, including to equip quantum programs with formal semantics, and how to handle communication and imperfection in current and future quantum computers.

While constructing practical software and algorithms for NISQ machines is likely to differ substantially from approaches geared towards scaleable and fault-tolerant quantum computing, the superoperator formalism may provide a unified and consistent representation that caters well to both scenarios. We expect that with an increasing interest for quantum computing in software engineering, more uses of the concept will appear in future literature, which makes it important for software engineering researchers to be aware of the necessary structures and methods.

Acknowledgements This work is partly supported by the German Federal Ministry of Education and Research within the funding program quantum technologies—from basic research to market—contract numbers 13N15647 and 13NI6092, and by the high-tech agenda of the free state of Bavaria. The author would like to thank Felix Greiwe, Tom Krüger and four semi-anonymous reviewers for helpful comments on a draft of this chapter that helped to substantially improve presentation.

References

Auletta G, Fortunato M, Parisi G (2009) *Quantum Mechanics.* Cambridge University Press DOI: 10.1017/CBO9780511813955

Bichsel B et al. (June 2020) ‘Silq: A High-Level Quantum Language with Safe Uncomputation and Intuitive Semantics’ *Proc. 41st ACM SIGPLAN* ACM: 286–300 ISBN: 978-1-4503-7613-6 DOI: 10.1145/3385412.3386007 (visited on 24/04/2023)

Cirac JI et al. (1999) Distributed quantum computation over noisy channels. *Phys. Rev. A* 59 (6): 4249–4254 DOI: 10.1103/PhysRevA.59.4249 URL: https://link.aps.org/doi/10.1103/PhysRevA.59.4249

Ekert A, Hosgold T (2022) *Introduction to Quantum Information Science.* URL: https://qubit.guide/qubit_guide.pdf

Garhwal S, Ghorani M, Ahmad A (Mar. 2021) Quantum Programming Language: A Systematic Review of Research Topic and Top Cited Languages. *Archives of Computational Methods in Engineering* 28(2): 289–310 ISSN: 1134-3060, 1886-1784 DOI: 10.1007/s11831-019-09372-6

Georgopoulos K, Emary C, Zuliani P (Dec. 2021) Modeling and simulating the noisy behavior of near-term quantum computers. *Phys. Rev. A* 104 (6): 062432 DOI: 10.1103/PhysRevA.104.062432 URL: https://link.aps.org/doi/10.1103/PhysRevA.104.062432

Green AS et al. (2013) Quipper: A Scalable Quantum Programming Language. *SIGPLAN Not.* 48(6): 333–342 ISSN: 0362-1340 DOI: 10.1145/2499370.2462177 URL: https://doi.org/10.1145/2499370.2462177

Greiwe F, Krüger T, Mauerer W (2023) 'Effects of Imperfections on Quantum Algorithms: A Software Engineering Perspective' en *2023 IEEE International Conference on Quantum Software (QSW)*: 31–42 DOI: 10.1109/QSW59989.2023.00014 URL: https://doi.org/10.1109/QSW59989.2023.00014

Harper R, Flammia ST, Wallman JJ (2020) Efficient learning of quantum noise. *Nature Physics* 16(12): 1184–1188 DOI: 10.1038/s41567-020-0992-8 URL: https://doi.org/10.1038/s41567-020-0992-8

Khatri S, Wilde MM (2020) Principles of quantum communication theory: A modern approach. *arXiv preprint arXiv:2011.04672*: DOI: https://doi.org/10.48550/arXiv.2011.04672

Kraus K (1983) *States, Effects, and Operations Fundamental Notions of Quantum Theory.* en vol. 190 Lecture Notes in Physics Springer, Berlin, Heidelberg ISBN: 9783540127321 DOI: 10.1007/3-540-12732-1 URL: http://link.springer.com/10.1007/3-540-12732-1

Lewis M, Soudjani S, Zuliani P (2023) Formal Verification of Quantum Programs: Theory, Tools and Challenges. *ACM Transactions on Quantum Computing*: DOI: 10.1145/3624483 URL: https://doi.org/10.1145/3624483

Leymann F (2019) 'Towards a Pattern Language for Quantum Algorithms' *Quantum Technology and Optimization Problems* vol. 11413 Lecture Notes in Computer Science (LNCS) Springer International Publishing, Cham: 218–230 DOI: 10.1007/978-3-030-14082-3_19

Marshall J et al. (2020) Characterizing local noise in QAOA circuits. *IOP SciNotes* 1(2): 025208 DOI: 10.1088/2633-1357/abb0d7 URL: https://dx.doi.org/10.1088/2633-1357/abb0d7

Mauerer W (2005) *Semantics and simulation of communication in quantum programming.* DOI: 10.48550/ARXIV.QUANT-PH/0511145 URL: https://arxiv.org/abs/quant-ph/0511145

Moggi E (1991) Notions of computation and monads. *Information and Computation* 93(1) Selections from 1989 IEEE Symposium on Logic in Computer Science: 55–92 ISSN: 0890-5401 DOI: https://doi.org/10.1016/

0890-5401(91)90052-4 URL: https://www.sciencedirect.com/science/article/pii/0890540191900524

Mu S-C, Bird R (Dec. 2001) 'Functional Quantum Programming' *Asian Workshop on Programming Languages and Systems* KAIST, Dajeaon, Korea URL: http://www.cs.ox.ac.uk/people/richard.bird/online/MuBird2001Functional.pdf

Nielsen MA, Chuang IL (2010) *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press DOI: 10.1017/CBO9780511976667 URL: https://doi.org/10.1017/CBO9780511976667

Ömer B (Sept. 2002) Procedural Quantum Programming. *AIP Conference Proceedings* 627(1): 276–285 ISSN: 0094-243X DOI: 10.1063/1.1503695 eprint: https://pubs.aip.org/aip/acp/article-pdf/627/1/276/11571870/276_1_online.pdf URL: https://doi.org/10.1063/1.1503695

Peduri A, Schaefer I, Walter M (2023) *QbC: Quantum Correctness by Construction.* arXiv: 2307.15641 [quant-ph]

Perdrix S (Nov. 2008) A Hierarchy of Quantum Semantics. *Electron. Notes Theor. Comput. Sci.* 192(3): 71–83 ISSN: 1571-0661 DOI: 10.1016/j.entcs.2008.10.028 URL: https://doi.org/10.1016/j.entcs.2008.10.028

Preskill J (2015) *Lecture Notes for Physics 229:Quantum Information and Computation.* CreateSpace Independent Publishing Platform ISBN: 9781506189918 URL: https://books.google.de/books?id=MIv8rQEACAAJ

Safi H, Wintersperger K, Mauerer W (2023) 'Influence of HW-SW-Co-Design on Quantum Computing Scalability' en *2023 IEEE International Conference on Quantum Software (QSW)*: 104–115 DOI: 10.1109/QSW59989.2023.00022

Schönberger M et al. (2022) 'Peel — Pile? Cross-Framework Portability of Quantum Software' *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*: 164–169 DOI: 10.1109/ICSA-C54293.2022.00039

Scott D, Strachey C (1971) *Toward a Mathematical Semantics for Computer Languages.* Programming Research Group Technical Monograph PRG-6 Oxford Univ. Computing Lab.

Selinger P (Aug. 2004) Towards a Quantum Programming Language. *Math. Struct. in Computer Science* 14(4): 527–586 ISSN: 0960-1295, 1469-8072 DOI: 10.1017/S0960129504004256

Svore K et al. (Feb. 2018) 'Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL' *Proceedings of the Real World Domain Specific Languages Workshop 2018* ACM: 1–10 ISBN: 978-1-4503-6355-6 DOI: 10.1145/3183895.3183901 (visited on 24/08/2023)

Vedral V (2006) *Introduction to Quantum Information Science (Oxford Graduate Texts).* Oxford University Press, Inc., USA ISBN: 0199215707

Vizotto J, Altenkirch T, Sabry A (2006) Structuring quantum effects: superoperators as arrows. *Mathematical Structures in Computer Science* 16(3): 453–468 DOI: 10.1017/S0960129506005287

Voichick F et al. (2023) Qunity: A Unified Language for Quantum and Classical Computing. *Proc. ACM Program. Lang.* 7(POPL): DOI: 10.1145/3571225 URL: https://doi.org/10.1145/3571225

Wecker D, Svore KM (Feb. 2014) LIQUi|>: A Software Design Architecture and Domain-Specific Language for Quantum Computing. *CoRR* abs/1402.4467: DOI: 10.48550/arXiv.1402.4467

Wintersperger K, Safi H, Mauerer W (Aug. 2022) 'QPU-System Co-Design for Quantum HPC Accelerators' *Proceedings of the 35th GI/ITG International Conference on the Architecture of Computing Systems* ed. by M Schulz et al. Gesellschaft für Informatik: 100–114 ISBN: 978-3-031-21867-5 DOI: https://doi.org/10.1007/978-3-031-21867-5_7

Xue C et al. (2021) Effects of Quantum Noise on Quantum Approximate Optimization Algorithm. *Chinese Physics Letters* 38(3): 030302 DOI: 10.1088/0256-307X/38/3/030302 URL: https://dx.doi.org/10.1088/0256-307X/38/3/030302

Ying M, Zhou L, Li Y (Aug. 2019) Reasoning about Parallel Quantum Programs.: DOI: 10.48550/arXiv.1810.11334 URL: http://arxiv.org/abs/1810.11334

Zhou L et al. (Jan. 2023) CoqQ: Foundational Verification of Quantum Programs. *Proc. ACM Program. Lang.* 7(POPL): DOI: 10.1145/3571222 URL: https://doi.org/10.1145/3571222