

# A Semantics for Quantum Loops

NICOLA ASSOLINI, University of Verona, Italy

ALESSANDRA DI PIERRO, University of Verona, Italy

Programming a quantum computer, i.e., implementing quantum algorithms on a quantum processor-based system, is a task that can be addressed (just as for classical computers) at different levels of abstraction. This paper proposes a denotational semantics for high-level quantum programming constructs, focusing on the conceptual meaning of quantum-controlled branching and iteration. We introduce a denotational domain where a mathematical meaning of a quantum control flow with loops can be defined, which reflects the coherent evolution of the quantum system implementing the program.

Additional Key Words and Phrases: Quantum Computing, Quantum Program Semantics, Denotational Semantics

## 1 INTRODUCTION

A crucial part of a computer program is its control flow. In classical computing, control flow refers to the sequencing and branching of instructions within a program, enabling the program to make decisions and alter its behavior based on specific conditions. This also implies the possibility of writing programs with conditional loops.

The control flow of quantum programs cannot be interpreted in the same way as for classical programs due to the properties of the target physical device (where they are intended to be executed), which behaves according to the laws of quantum mechanics. Notably, a quantum processor is able to work on *superpositions* of states (qubits) rather than on single ones and, in a more strikingly different way from a classical computer, it can generate states which are *entangled*, i.e., tied to each other by a strong (non-classical) correlation. Moreover, while the results of the execution of a classical program are immediately available whenever the program reaches the final statement, accessing the results of a quantum program is not so straightforward, due to the so-called measurement problem in the theory of quantum mechanics. In fact, although quantum theory is up to now the most precise description of how the world behaves, the interpretation of such a behaviour is controversial and the debate on which, among the several interpretations that have been proposed, is the right one is still open and represents the main problem for a complete understanding of quantum physics.

When analysing a quantum program and studying its mathematical behaviour, it is inevitable to refer to the interpretation of quantum mechanics. Typically, the quantum programming language literature refers to one of the most accredited interpretations, which goes under the name of the Copenhagen interpretation. According to this interpretation, the results of the execution of a quantum program can only be obtained when the coherent (i.e. in superposition) execution (or evolution of the quantum system) collapses into a classical state, which occurs randomly both in time (at a given average rate), and in space (according to the Born rule). This explanation avoids the measurement problem and leads to modelling the result of a quantum program essentially as a probability distribution on all its possible outcomes.

In this paper, we aim at a description of a quantum program that is as general as possible by avoiding an explicit syntactic construct for measurement, which would effectively lead to a (classical) probabilistic semantics of the program. Moreover, in giving a meaning to a quantum program, we will concentrate on the denotational approach, following Strachey's observation that

fixing the domain within which programs in a given language have their meanings tell us a great deal about the language, and is a sure guide to the design of the language.

At this point, a crucial question arises: *how can we handle termination in quantum loops*? In classical computation, a Turing machine can use a *halting bit* to signal termination. Is it possible to define a halting qubit that can be observed without disrupting coherence, similar to classical termination detection? As analyzed in [10, 12, 13, 20, 21], defining such a halting qubit is not possible without disturbing the computation. Unlike classical loops, where execution can be stopped based on the guard's value, measuring a quantum guard would make a quantum superposition collapse to a classical state, thus altering the computation itself. Quantum languages that rely on measurement-based control flow circumvent this issue but at the cost of introducing non-unitary behavior.

An even deeper problem, also highlighted in the works mentioned above, is that a quantum loop can terminate on some inputs while diverging on others, leading to a superposition of terminating and non-terminating states. Since quantum computation is performed by unitary operators transforming quantum states (or superpositions) into quantum states, to determine from the outside whether a quantum execution has reached a fixed point, one should either arbitrarily interrupt the computation after a finite number of steps, thus making it terminate on all inputs, or let it evolve indefinitely. Restricting only to finite computations would not provide an ideal solution to the problem of defining a formal semantics for quantum computation, as it would prevent us from formally reasoning about termination.

Therefore, just as for the classical case, having a semantics that captures also infinite computation is crucial for correctly modeling quantum programs, although in the quantum case, achieving this result is more problematic for the reasons discussed above. In this paper we show how a model that characterizes which parts of the execution terminate and which do not in a quantum loop can be defined by approximating the concrete unitary behaviour of a quantum program by some linear operators which allows the superposition between an execution that has reached a fixed point and another that is still computing even in the limit, i.e. in the case of an infinite loop. In fact, by relaxing the unitarity constraint, we are able to construct an approximating sequence of linear operators which converges to a mathematically well-defined limit. While for finite computations, this corresponds exactly to the actual (unitary) behaviour of a quantum program, in the presence of infinite computations, it is a linear operator with a norm strictly less than one due to the portion of the superposition on which computation is still going on.

## 2 BACKGROUND

Programs in a quantum programming language are designed to run on quantum computers and are very different from classical computer programs. The design and implementation of such languages requires a sound knowledge of the principles of quantum mechanics and the underlying mathematics. In this section, we briefly recall the main aspects of quantum computation that make this computational model different from the classical one. We will refer to the circuit model of computation and highlight such differences in terms of the meaning of wires and gates in a classical and a quantum circuit.

In a quantum circuit, wires represent quantum bits, or qubits, rather than bits. The classical unit of information (the bit) generates, with its two values 0 and 1, a complex vector space (a quantum system), where each complex vector of norm 1 is the state of a qubit. This is, therefore, a linear combination of the form  $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ , where  $\alpha$  and  $\beta$  are complex numbers from which we can infer the probability of the state resulting (after measurement) in 1 or 0, respectively. Such probabilities are obtained as  $|\alpha|^2$  and  $|\beta|^2$ , which explains why quantum states must be normalized

vectors, i.e.  $|\alpha|^2 + |\beta|^2 = 1$  must hold<sup>1</sup>. Such vectors live in a complex Hilbert space, equipped with the  $\ell^2$  norm<sup>2</sup> [8]. The state of  $n$  qubits corresponds to a unit vector in the  $2^n$ -dimensional Hilbert space ( $\mathcal{H}^{2^n}$ ) obtained by composing by tensor product the normalized states corresponding to each wire (qubit), i.e. a vector in a 2-dimensional complex Hilbert space ( $\mathcal{H}^2$ ) [14, Chapter 2]. For instance, the space of two qubits is  $\mathcal{H}^4 = \mathcal{H}^2 \otimes \mathcal{H}^2$  and a generic state  $|\psi\rangle$  in  $\mathcal{H}^4$  can be written as  $|\psi\rangle = \alpha_0 |00\rangle + \alpha_1 |01\rangle + \alpha_2 |10\rangle + \alpha_3 |11\rangle$ , where all  $\alpha_i$  are complex numbers.

Throughout the paper, we will adopt the following convention. For a variable  $q$ , we will write  $|\psi\rangle_q$  to indicate that  $q$  represents the state  $|\psi\rangle$  of a qubit register. For multi qubits states, such as for example  $1/\sqrt{2}(|01\rangle + |10\rangle)$ , we will write  $1/\sqrt{2}(|01\rangle + |10\rangle)_{p,q}$  to indicate that variable  $p$  represents the first qubit and  $q$  represents the second qubit of the entangled pair.

### 3 QUANTUM WHILE LANGUAGE

To present our semantics, we introduce a generic quantum language that contains quantum loop iterations, sequential composition, and unitary transformation. The syntax of our simple language, which we will refer to as  $SL$ , is given by the following grammar defining a program  $s$  as follows:

$$s ::= U(\bar{q}) \mid s; s \mid \text{skip} \mid \text{while } q \text{ do } \{s\}, \quad (1)$$

where  $q$  is a quantum variable and  $\bar{q}$  denotes a sequence  $q^1, \dots, q^n$  of quantum variables. In  $SL$ , we do not define a command for state's initialization (or assignment) since we assume that all variables are initialized to  $|0 \dots 0\rangle$ . All operations on variables are performed by statement  $U(\bar{q})$  corresponding to applying a unitary transformation  $U$  to  $\bar{q}$ . Every possible quantum transformation, except measurement, is a composition of unitary transformations. The principle of deferred measurement [14, Chapter 4] guarantees that all intermediate measurements in a quantum circuit can be moved to the end of the computation; thus, excluding measurement operation from our languages does not cause a loss of generality. Moreover, as explained in Section 1, this will make our treatment independent of any specific interpretation of quantum mechanics. Since measurement does not occur in  $SL$  programs, their semantics results in a deterministic transition between quantum states according to the physical law (the Shrödinger equation) governing the quantum system on which the program is intended to be executed, without having to establish when quantum mechanics should leave space to classical mechanics (which essentially constitutes the measurement problem in quantum mechanics).

#### 3.1 The State Space

To define the semantics of  $SL$ , we consider a domain similar to the one described in [27, Chapter 3]. Given a program  $s$ , let  $Q_s$  be the set of all variables occurring in  $s$ . Each quantum variable  $q \in Q_s$  has a type  $\mathcal{H}_q$ , and it is interpreted as a vector  $|\varphi\rangle_q$  in its own Hilbert space  $\mathcal{H}_q$ . For example, for a Boolean variable  $b$ , the state  $|\varphi\rangle_b = \alpha |0\rangle + \beta |1\rangle$ ,  $\alpha, \beta \in \mathbb{C}$  corresponds to a vector in the complex Hilbert space of dimension 2 (the state space of 1 qubit), which is the type of  $b$ . We define

$$\mathcal{H}_{Q_s} = \bigotimes_{q \in Q_s} \mathcal{H}_q, \quad (2)$$

as the space of the types of all variables in  $Q$ . We consider an infinitely countable set  $T = \{t_i\}$  of ancillary quantum boolean variables, which are necessary to perform quantum while loops, and their Hilbert space:

$$\mathcal{H}_T = \bigotimes_{t_i \in T} \mathcal{H}_{t_i} \text{ where } \mathcal{H}_{t_i} = \mathcal{H}^2 \quad (3)$$

<sup>1</sup>The *ket* notation  $|\psi\rangle$  is due to Dirac and represents the vector  $(\alpha, \beta)^T$  in linear algebraic notation.

<sup>2</sup>The  $\ell^2$  norm of a vector  $|\psi\rangle = [\alpha_1, \alpha_2, \dots, \alpha_n]^T$  is defined as  $\| |\psi\rangle \| = \sum_{i=1}^n \sqrt{|\alpha_i|^2}$ .



Fig. 1. The circuits corresponding to  $U(\bar{q})$  (a) and  $s_1; s_2$  (b).

Finally, we call  $\mathcal{H}_p = \mathcal{H}_T \otimes \mathcal{H}_{Q_s}$  the Hilbert space of the program  $s$ .

#### 4 UNITARY SEMANTICS

A quantum language with no classical operations can be completely described by using unitary operators that can be visually represented by quantum circuits. A mathematical description of these circuits is by means of linear algebra and, in particular, by the group of unitary operators on a Hilbert space.

We start by defining the semantics for each statement in  $SL$  except the while statement.

*Definition 4.1.* Let  $\mathcal{U}(\mathcal{H}_p)$  be the group of unitary operators from  $\mathcal{H}_p$  to  $\mathcal{H}_p$ . The unitary semantics is the function  $[\cdot] : s \rightarrow \mathcal{U}(\mathcal{H}_p)$ , defined by:

- (1)  $[\text{skip}] = \mathbf{I}_{\mathcal{H}_p}$ ;
- (2)  $[U(\bar{q})] = \bar{U}$ , where  $\bar{U} = \mathbf{I} \otimes U \otimes \mathbf{I}$ , i.e. the extension of  $U$  on  $\mathcal{H}_p$ ;
- (3)  $[s_1; s_2] = [s_2] \cdot [s_1]$ .

The rule (2) corresponds to computing the operator  $U$  on  $\bar{q}$ , and the identity on the other components of the program space, and rule (3) corresponds to the sequential composition of operators. Since  $\bar{U}$  and  $\mathbf{I}$  are unitary operators, the semantics of each statement is a unitary operator. Therefore, every statement can be represented by a circuit as shown in Figure 1. The case for the while is slightly more involved, and we treat it in the next section.

##### 4.1 While Loop Semantics

A well-formed while statement should modify its guard within the loop body. To define a quantum while instruction, we need a representation where the guard qubit can be updated while preserving unitarity. Ideally, given a program  $p = \text{if } q \text{ do } \{U(q)\}$ , i.e. the equivalent of one iteration of a while statement, we would like to have a unitary operator  $C$  such that  $C(\alpha|0\rangle_q + \beta|1\rangle_q) = \alpha|0\rangle_q + \beta U|1\rangle_q$ . However, in quantum computing, controlled operations cannot have their controller affected by the target, as this would break unitarity. For example, consider  $p = \text{if } q \text{ do } \{X(q)\}$ . The corresponding operator  $C$  would map  $|0\rangle_q$  to  $|0\rangle_q$  and  $|1\rangle_q$  to  $|0\rangle_q$ , which is clearly non-injective and thus non-unitary. We must introduce an auxiliary qubit to model a self-controlled operation within a unitary framework on which to copy information via a CNOT gate. Figure 2a illustrates a hypothetical quantum if-statement where the guard is included in the body. The semantics can be represented by a unitary operator  $CU$  such that  $CU(|0\rangle_t \otimes (\alpha|0\rangle_q + \beta|1\rangle_q)) = \alpha|0\rangle_t|0\rangle_q + \beta|1\rangle_t U|1\rangle_q$ . Extending this approach to a while loop introduces an additional challenge: each iteration requires a fresh temporary qubit, which implies the need for an infinite set of auxiliary qubits to model non-terminating loops. Figure 2b shows the circuit representation of a quantum while loop, which is recursively defined and corresponds to an infinite composition of unitary operations on an infinite-dimensional Hilbert space.

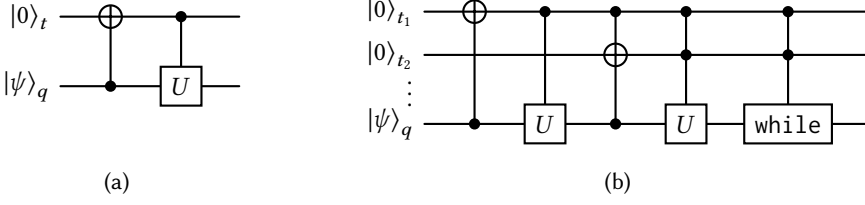


Fig. 2. if  $q$  do  $\{U(q)\}$  circuit **(a)** and while  $q$  do  $\{U(q)\}$  circuit **(b)**

The first step in defining the semantics of a while loop is to start with the controlled operation. In general, consider a Hilbert space  $\mathcal{H} = \mathcal{H}_g \otimes \mathcal{H}_s$ , where  $\mathcal{H}_g$  is a 2-dimensional Hilbert space representing the control (guard) qubit, and  $\mathcal{H}_s$  is the space of target qubits. Given a unitary operator  $U_s$  in  $\mathcal{H}_s$ , the unitary operator corresponding to  $U_s$  controlled by  $g$  can be represented by  $|0\rangle\langle 0|_g \otimes \mathbf{I}_s + |1\rangle\langle 1|_g \otimes U_s$ , where  $\mathbf{I}_s$  is the identity in  $\mathcal{H}_s$  and  $|0\rangle\langle 0|_g$  and  $|1\rangle\langle 1|_g$  are the projector on  $|0\rangle_g$  and  $|1\rangle_g$  respectively in  $\mathcal{H}_g$ .

**PROPOSITION 4.2.** *Let  $P_{i_g} = (|i\rangle\langle i|_g \otimes \mathbf{I}_s)$  be the projector  $|i\rangle\langle i|_g$  in the whole  $\mathcal{H}$  and  $\bar{U} = (\mathbf{I}_g \otimes U_s)$  the extension of  $U_s$  in  $\mathcal{H}$ . The controlled operation  $|0\rangle\langle 0|_g \otimes \mathbf{I}_s + |1\rangle\langle 1|_g \otimes U_s$  can be written as  $P_{0_g} + P_{1_g} \cdot \bar{U}$ .*

**PROOF.** By the property  $AC \otimes BD = (A \otimes B)(C \otimes D)$  [16, Lemma 4.2.10] we have  $|1\rangle\langle 1|_g \otimes U_s = (|1\rangle\langle 1|_g \cdot \mathbf{I}_g) \otimes (\mathbf{I}_s \cdot U_s) = (|1\rangle\langle 1|_g \otimes \mathbf{I}_s)(\mathbf{I}_g \otimes U_s) = P_{1_g} \cdot \bar{U}$ . Thus  $|0\rangle\langle 0|_g \otimes \mathbf{I}_s + |1\rangle\langle 1|_g \otimes U_s = P_{0_g} + P_{1_g} \cdot \bar{U}$ .  $\square$

As shown in Figure 2b, to perform a quantum loop, we need to make a quantum ‘copy’ of the guard variable in a new fresh ancillary variable for every iteration. This is realized by a CNOT gate, which we will denote by  $\mathbf{G}(q, n)$  in our semantics, where  $q$  is a variable in  $Q_s$  and  $n \in \mathbb{N}$ ,  $n \geq 1$  indicates the target  $t_n \in T$ .

Given a qubit  $q \in Q_s$ , and a unitary operator  $S$  that does not act on  $t_1$ , we recursively define an operator  $w_n(q, S)$ , using the control operation in the format introduced in Proposition 4.2, as follows:

$$\begin{aligned} w_0(q, S) &= \mathbf{I} \\ w_n(q, S) &= (P_{0_{t_1}} + P_{1_{t_1}} \cdot S(w_{n-1}(q, S)) \cdot S) \mathbf{G}(q, 1). \end{aligned} \quad (4)$$

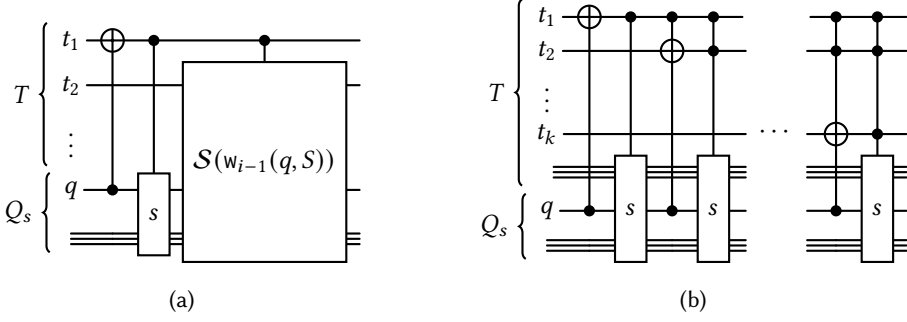
where  $S$  shifts the controls to keep the first ancilla qubit free. In particular,  $S(\mathbf{G}(q, n)) = \mathbf{G}(q, n+1)$  and  $S(P_{j_{t_n}}) = P_{j_{t_{n+1}}}$ . Let us look closer to  $w_i(q, S)$ . The operator  $w_i(q, S)$  corresponds to the circuit depicted in Figure 3a. Specifically, it includes the component  $\mathbf{G}(q, 1)$ , which represents the first controlled-not operation, and it incorporates the operator defined by  $(P_{0_{t_1}} + P_{1_{t_1}} \cdot S(w_{n-1}(q, S))S)$ . This expression represents the operation  $S$  followed by  $S(w_{n-1}(q, S))$  controlled by  $t_1$ . Here,  $S(w_{n-1}(q, S))S$  denotes the composition of the unitary  $S$ , which encodes the semantics of the body of the while loop, and  $S(w_{n-1}(q, S))$ , which captures the semantics of the remaining part of the while loop.

**PROPOSITION 4.3.** *The closed formula of Equation 4, is:*

$$w_n(q, S) = \sum_{h=1}^n \left( \prod_{i=1}^{h-1} (P_{1_{t_i}}) \cdot P_{0_{t_h}} \cdot \prod_{i=n-h}^{n-2} (\mathbf{G}(q, n-i) S) \cdot \mathbf{G}(q, 1) \right) + \prod_{i=1}^n P_{1_{t_i}} \cdot \prod_{i=0}^{n-1} S \mathbf{G}(q, n-i). \quad (5)$$

The proof is given in Appendix A.

Given a while  $q$  do  $\{s\}$  statement, we can build the chain of finite unitary approximation  $\{W(q, [s])\}_n$ . To define the semantics of the general construct, we must now consider the case of an infinite loop. However, we can show that there is no limit to this sequence. From [24], we

Fig. 3. Finite  $k$ -while loop

recall the notion of *strong convergence* and an important theorem about the convergence of an infinite sequence of operators.

**Definition 4.4.** Let  $T_n$  and  $T$  be linear operators from  $\mathcal{H}_P$  to itself. If  $\|T_n |\psi\rangle - T |\psi\rangle\| \rightarrow 0$  as  $n \rightarrow \infty, \forall |\psi\rangle \in \mathcal{H}_P$ , then the sequence of operators  $\{T_n\}$  is **strongly convergent** to  $T$  (denoted as  $T_n \rightarrow T$ ).

**THEOREM 4.5.** Let  $\{T_n\}_n$  be a sequence of bounded linear operators from  $X \rightarrow Y$ , where  $X$  and  $Y$  are Banach spaces.  $T_n \rightarrow T$  if and only if the sequence  $\{\|T_n\|\}_n$  is bounded and the sequence  $\{T_n x\}_n$  is a Cauchy sequence in  $Y$  for all  $x \in M \subset X$ , where the span of  $M$  is dense in  $X$ .

A sequence  $\{x_n\}_n$  in a Hilbert space is said to be a Cauchy sequence if, for any positive real number  $\epsilon > 0$ , there exists a positive integer  $N$  such that for all positive integers  $m$  and  $n$  greater than  $N$ , the distance  $\|x_m - x_n\| < \epsilon$ .

Let us consider as an example the quantum while  $q$  do {skip}, evaluated on  $|\psi\rangle_P = |0 \dots\rangle_T |1\rangle_q$ . This loop corresponds to the following chain:

$$\begin{aligned} W_0(q, X_q) |0 \dots\rangle_T |1\rangle_q &= |0 \dots\rangle_T |1\rangle_q, \\ W_1(q, X_q) |0 \dots\rangle_T |1\rangle_q &= |10 \dots\rangle_T |1\rangle_q, \\ W_n(q, X_q) |0 \dots\rangle_T |1\rangle_q &= |1^{\otimes n} 0 \dots\rangle_T |1\rangle_q. \end{aligned} \tag{6}$$

We see that for all  $n$ ,  $\|W_{n+1}(q, X_q) |\psi\rangle_P - W_n(q, X_q) |\psi\rangle_P\| = 2$ . More in general, if

$$|\psi\rangle_P = |0 \dots\rangle_T (\alpha |0\rangle + \beta |1\rangle)_q,$$

we have

$$W_n(q, X_q) |\psi\rangle_P = \alpha |0 \dots\rangle_T |0\rangle_q + \beta |1^{\otimes n} 0 \dots\rangle_T |1\rangle_q,$$

and for all  $n$ ,  $\|W_{n+1}(q, X_q) |\psi\rangle_P - W_n(q, X_q) |\psi\rangle_P\| = 2\beta^2$ . Since for infinite vectors the distance between the element of  $\{W_n(q, X_q) |\psi\rangle_P\}_n$  is constant, the sequence  $\{W_n(q, X_q) |\psi\rangle_P\}_n$  is not Cauchy, and for [Theorem 4.5](#) the sequence  $\{W_n(q, X_q)\}_n$  is not strongly convergent. The unitary semantics is therefore only able to model a while statement limited to a certain finite number of iterations  $k$ , which we call called  $\text{while}^k q$  do  $\{s\}$ . The semantics for this statement is defined as follows:

$$[\text{while}^k q \text{ do } \{s\}] = W_k(q, [s]).$$

This unitary operator can be represented by the circuit in [Figure 3b](#).

## 4.2 Example

Let us see how the operator  $W_n$  works with some examples. First we consider  $\text{while}^k q \text{ do } \{X(q)\}$  with the state  $|\psi\rangle = \frac{1}{\sqrt{2}}|0 \dots\rangle_T (|0\rangle_q + |1\rangle_q)$  as input:

$$\begin{aligned} W_0(q, X_q) |\psi\rangle &= \frac{1}{\sqrt{2}}|0 \dots\rangle_T (|0\rangle_q + |1\rangle_q) \\ W_1(q, X_q) |\psi\rangle &= \frac{1}{\sqrt{2}}(|0 \dots\rangle_T |0\rangle_q + |10 \dots\rangle_T |0\rangle_q) \\ W_2(q, X_q) |\psi\rangle &= \frac{1}{\sqrt{2}}(|0 \dots\rangle_T |0\rangle_q + |10 \dots\rangle_T |0\rangle_q) \\ &\dots \end{aligned} \tag{7}$$

Note that with this input, the while loop fully terminates, and in fact,  $W_n(q, X_q)$  reaches a fix-point. Things are different when not-termination is involved.

Let's now consider the loop  $\text{while } q \text{ do } \{H(q)\}$  with the same input  $|\psi\rangle$ :

$$\begin{aligned} W_0(q, H_q) |\psi\rangle &= \frac{1}{\sqrt{2}}|0 \dots\rangle_T (|0\rangle_q + |1\rangle_q) \\ W_1(q, H_q) |\psi\rangle &= \frac{1}{\sqrt{2}}|0 \dots\rangle_T |0\rangle_q + \frac{1}{2}(|10 \dots\rangle_T |0\rangle_q - |10 \dots\rangle_T |1\rangle_q) \\ W_2(q, H_q) |\psi\rangle &= \frac{1}{\sqrt{2}}|0 \dots\rangle_T |0\rangle_q + \frac{1}{2}|10 \dots\rangle_T |0\rangle_q \\ &\quad + \frac{1}{\sqrt{8}}(|110 \dots\rangle_T |0\rangle_q - |110 \dots\rangle_T |1\rangle_q) \\ W_3(q, H_q) |\psi\rangle &= \frac{1}{\sqrt{2}}|0 \dots\rangle_T |0\rangle_q + \frac{1}{2}|10 \dots\rangle_T |0\rangle_q \\ &\quad + \frac{1}{\sqrt{8}}|110 \dots\rangle_T |0\rangle_q + \frac{1}{4}(|1110 \dots\rangle_T |0\rangle_q - |1110 \dots\rangle_T |1\rangle_q) \\ W_n(q, H_q) |\psi\rangle &= \sum_{i=0}^n \frac{1}{\sqrt{2^{i+1}}} |1^{\otimes i} 0 \dots\rangle_T |0\rangle_q - \frac{1}{\sqrt{2^{n+1}}} |1^{\otimes n} 0 \dots\rangle_T |1\rangle_q. \end{aligned} \tag{8}$$

Since each  $t_j$  controls the execution of the  $j$ -th iteration,  $t_j = 1$  indicates that the  $j$ -th iteration has been executed (see [Figure 3b](#) and [Figure 2b](#)). During each iteration of  $w_n$ , the terminating part of the state—where  $q$  equals  $|0\rangle$ —is gradually increased. In contrast, the portion corresponding to non-termination, where  $q$  equals  $|1\rangle$ , decreases but never fully reaches zero.

Finally, consider again the while loop  $\text{while } q \text{ do } \{\text{skip}\}$ , evaluated on  $|0 \dots\rangle_T |1\rangle_q$  in [Equation 6](#). Here for a divergent loop, each  $w_n$  differs from the previous ones, reflecting that the unitary semantics corresponds to the partial results of the divergent loop computation.

## 5 LINEAR SEMANTICS

To model infinite computation in quantum computing, we need to enlarge the domain of denotations so as to include an appropriate limit. To this purpose, we observe that a unitary operator is also bounded and, therefore, can be seen as an element of the Banach Space of linear bounded operators on  $\mathcal{H}_P$ .

We start by defining the linear semantics for each statement in  $SL$ , except the while statement.

*Definition 5.1.* Let  $\mathcal{B}(\mathcal{H}_P)$  be the space of linear bounded operators from  $\mathcal{H}_P$  to  $\mathcal{H}_P$  equipped with the operator norm  $\|A\| = \sup_{\|\psi\|=1} \|A|\psi\rangle\|$ . The linear semantics is a function  $\llbracket \cdot \rrbracket : s \rightarrow \mathcal{B}(\mathcal{H}_P)$ , defined by:

- (1)  $\llbracket \text{skip} \rrbracket = I_{\mathcal{H}_P}$ ;
- (2)  $\llbracket U(\bar{q}) \rrbracket = \bar{U}$ , where  $\bar{U} = I \otimes U \otimes I$ , i.e. the extension of  $U$  on  $\mathcal{H}_P$ ;
- (3)  $\llbracket s_1; s_2 \rrbracket = \llbracket s_2 \rrbracket \cdot \llbracket s_1 \rrbracket$ ;

Notice that the linear semantics of these statements are the same as the unitary semantics defined in [Definition 4.1](#). For the while statement, we introduce a new linear bounded operator. Let  $S$  be a bounded linear operator and  $q \in Q_s$ . We define the recursive operator  $l_n(q, S)$  as follows:

$$\begin{aligned} l_0(q, S) &= 0 \\ l_n(q, S) &= (P_{0_{t_1}} + P_{1_{t_1}} \cdot S(l_{n-1}(q, S)) \cdot S)G(q, 1), \end{aligned} \tag{9}$$

where  $0$  is the zero operator in  $\mathcal{H}_P$ , and  $S$  produces the shift  $t_n \rightarrow t_{n+1}$ . We remark that for defining  $l_n(q, S)$ , we use the definition of the controlled operation presented in [Proposition 4.2](#). Note that, for  $n > 0$ , the operator  $l_n(q, S)$  is equivalent to the operator  $w_n(q, S)$  as defined in [Equation 4](#). Indeed, in this case,  $l_n(q, S)$  is defined as the composition of the controlled-not gate, which evaluates the guard, and the operator that corresponds to the composition of  $S$ , representing the semantics of the loop body, and the operator representing the semantics of  $l_{n-1}(q, S)$  shifted by  $S$ .

**PROPOSITION 5.2.** *For all  $n$ , if  $S$  is bounded then  $l_n(q, S)$  is bounded.*

**PROOF.** Let's consider  $l_n(q, S)$  and a vector  $|\psi\rangle$  such that  $\|\psi\| = 1$ :

$$\begin{aligned}
 \|(P_{0_{t_1}} + P_{1_{t_1}} \cdot S(l_{n-1}(q, S)) \cdot S)G(q, 1) |\psi\rangle\|^2 &= \\
 &\quad (\text{since } G(q, 1) \text{ is unitary}) \\
 &= \|(P_{0_{t_1}} + P_{1_{t_1}} \cdot S(l_{n-1}(q, S)) \cdot S) |\psi\rangle\|^2 = \\
 &= \|P_{0_{t_1}} |\psi\rangle + P_{1_{t_1}} \cdot S(l_{n-1}(q, S)) \cdot S |\psi\rangle\|^2 = \\
 &\quad (P_{0_{t_1}} |\psi\rangle \text{ and } P_{1_{t_1}} S(l_{n-1}(q, S)) S |\psi\rangle \text{ are orthogonal [8]}) \\
 &= \|P_{0_{t_1}} |\psi\rangle\|^2 + \|P_{1_{t_1}} S(l_{n-1}(q, S)) S |\psi\rangle\|^2 \leq \\
 &\quad (S(l_{n-1}(q, S)) S \text{ is bounded}) \\
 &\leq \|P_{0_{t_1}} |\psi\rangle\|^2 + \|P_{1_{t_1}} |\psi\rangle\|^2 = \\
 &= \|(P_{0_{t_1}} + P_{1_{t_1}}) |\psi\rangle\|^2 = \|\psi\|^2 = 1.
 \end{aligned}$$

Since  $\|l_n(q, S) |\psi\rangle\|^2 \leq 1$  also  $\|l_n(q, S) |\psi\rangle\| \leq 1$ . □

**PROPOSITION 5.3.** *The closed formula of [Equation 9](#) is:*

$$L_n(q, S) = \sum_{k=1}^n \left( \prod_{i=1}^{k-1} (P_{1_{t_i}}) \cdot P_{0_{t_k}} \cdot \prod_{i=n-k}^{n-2} (G(q, n-i) \cdot S) \cdot G(q, 1) \right). \quad (10)$$

The proof by induction can be found in the [Appendix A](#).

Now, given the while  $q$  do  $\{s\}$  statement, we can define a chain of linear operator  $\{L_n(q, \llbracket s \rrbracket)\}_n$  that represent the linear semantics of all finite execution of the while loop.

**PROPOSITION 5.4.** *For all  $n$ ,  $L_n(q, \llbracket s \rrbracket)$  is bounded.*

**PROOF.** The proof follows by means of a structural induction on  $s$  and [Proposition 5.2](#) □

To define the semantics of the whole construct now, we must consider the case of an infinite loop. Thus, we need to consider the limit of the sequence  $\{L_n(q, \llbracket s \rrbracket)\}_n$ . From [Theorem 4.5](#), we know that if we prove that the sequence  $\{\|L_n(q, \llbracket s \rrbracket)\|\}$  is bounded and that, for all  $|\psi\rangle \in \mathcal{H}_P$ , the sequence  $\{L_n(q, \llbracket s \rrbracket) |\psi\rangle\}$  is a Cauchy sequence in  $\mathcal{H}_P$ , then the sequence  $\{L_n(q, \llbracket s \rrbracket)\}_n$  has a limit. First, we prove that  $\{\|L_n(q, \llbracket s \rrbracket)\|\}_n$  is bounded.

**PROPOSITION 5.5.** *The sequence  $\{\|L_n(q, \llbracket s \rrbracket)\|\}$  is bounded.*

**PROOF.** From [Proposition 5.4](#), we know that for all  $n$ ,  $\|L_n(q, \llbracket s \rrbracket)\| \leq 1$ , thus the sequence  $\{\|L_n(q, \llbracket s \rrbracket)\|\}$  is bounded above [\[11\]](#). □

Now we have to prove that, for all  $|\psi\rangle \in \mathcal{H}_P$ , the sequence  $\{L_n(q, \llbracket s \rrbracket) |\psi\rangle\}$  is Cauchy. We start with some support lemmas.

**LEMMA 5.6.** *For all  $n > 0$ ,  $L_n(q, \llbracket s \rrbracket) = \sum_{i=1}^n (L_i(q, \llbracket s \rrbracket) - L_{i-1}(q, \llbracket s \rrbracket))$ .*



PROOF. In general, given a sequence  $A_n$ , by induction on  $n$  and by straightforward arithmetic simplifications, we can always prove that  $A_n = \sum_{i=1}^n (A_i - A_{i-1}) + A_0$ . In our case,  $L_0(q, \llbracket s \rrbracket) = 0$ , thus  $L_n(q, \llbracket s \rrbracket) = \sum_{i=1}^n (L_i(q, \llbracket s \rrbracket) - L_{i-1}(q, \llbracket s \rrbracket))$ .  $\square$

LEMMA 5.7. *For every  $n \neq m$ ,  $L_n(q, \llbracket s \rrbracket) - L_{n-1}(q, \llbracket s \rrbracket)$  and  $L_m(q, \llbracket s \rrbracket) - L_{m-1}(q, \llbracket s \rrbracket)$  are orthogonal.*

PROOF. From Equation 10, we can compute:

$$\begin{aligned} L_n(q, \llbracket s \rrbracket) - L_{n-1}(q, \llbracket s \rrbracket) &= \prod_{i=1}^{n-1} (P_{1_{t_i}}) \cdot P_{0_{t_n}} \cdot \prod_{i=0}^{n-2} (G(q, n-i) \cdot S) \cdot G(q, 1), \\ L_m(q, \llbracket s \rrbracket) - L_{m-1}(q, \llbracket s \rrbracket) &= \prod_{i=1}^{m-1} (P_{1_{t_i}}) \cdot P_{0_{t_m}} \cdot \prod_{i=0}^{m-2} (G(q, m-i) \cdot S) \cdot G(q, 1). \end{aligned}$$

Clearly, if  $n \neq m$ ,  $\prod_{i=1}^{n-1} (P_{1_{t_i}}) P_{0_{t_n}}$  and  $\prod_{i=1}^{m-1} (P_{1_{t_i}}) P_{0_{t_m}}$  are orthogonal, and thus  $L_n(q, \llbracket s \rrbracket) - L_{n-1}(q, \llbracket s \rrbracket)$  and  $L_m(q, \llbracket s \rrbracket) - L_{m-1}(q, \llbracket s \rrbracket)$  are orthogonal.  $\square$

LEMMA 5.8. *For every  $n, m$ , if  $n \leq m$ , then  $\|L_n(q, \llbracket s \rrbracket) |\psi\rangle\| \leq \|L_m(q, \llbracket s \rrbracket) |\psi\rangle\|$ .*

PROOF. From Equation 10, we observe that if  $n \leq m$ , the summation corresponding to  $L_m(q, \llbracket s \rrbracket)$  contains all the terms of  $L_n(q, \llbracket s \rrbracket)$  plus additional terms. Recall that each term in the summation consists of projectors that are mutually orthogonal, and the sum of all projectors is different from the identity. Thus, if  $L_n(q, \llbracket s \rrbracket) |\psi\rangle = \sum \alpha_i |e_i\rangle$  and  $L_m(q, \llbracket s \rrbracket) |\psi\rangle = \sum \beta_i |e_i\rangle$ , where  $|e_i\rangle$  is a standard basis of  $\mathcal{H}_P$ , then  $\forall i, \alpha_i \neq 0 \Rightarrow \beta_i = \alpha_i$ . In other words,  $L_n(q, \llbracket s \rrbracket) |\psi\rangle$  is a substate of  $L_m(q, \llbracket s \rrbracket) |\psi\rangle$  since  $L_m(q, \llbracket s \rrbracket)$  projects the state on ‘more basis vectors’ than  $L_n(q, \llbracket s \rrbracket)$ .  $\square$

THEOREM 5.9. *For all vectors  $|\psi\rangle \in \mathcal{H}_P$ , the sequence  $\{L_n(q, \llbracket s \rrbracket) |\psi\rangle\}_n$  is Cauchy.*

PROOF. By Lemma 5.6,  $L_n(q, \llbracket s \rrbracket) |\psi\rangle = \sum_{i=1}^n (L_i(q, \llbracket s \rrbracket) - L_{i-1}(q, \llbracket s \rrbracket)) |\psi\rangle$ . By Lemma 5.7 and [8, Exercise 2] it holds that:

$$\|L_n(q, \llbracket s \rrbracket) |\psi\rangle\|^2 = \sum_{i=1}^n \|(L_i(q, \llbracket s \rrbracket) - L_{i-1}(q, \llbracket s \rrbracket)) |\psi\rangle\|^2.$$

Let us consider the sequence of partial sums  $\{\sum_{j=1}^n \|(L_j(q, \llbracket s \rrbracket) - L_{j-1}(q, \llbracket s \rrbracket)) |\psi\rangle\|^2\}_n$ . By Proposition 5.4 and Lemma 5.8, we know that all partial sums are bounded and that the sequence of partial sums is increasing. Since the sequence of partial sums is both increasing and bounded above, it must converge [11]. This implies that the infinite sum  $\sum_{j=1}^{\infty} \|(L_j(q, \llbracket s \rrbracket) - L_{j-1}(q, \llbracket s \rrbracket)) |\psi\rangle\|^2$  also converges [22]. We have that  $\sum_{i=1}^{\infty} \|(L_i(q, \llbracket s \rrbracket) - L_{i-1}(q, \llbracket s \rrbracket)) |\psi\rangle\|^2$  converges, and for all  $i$ ,  $\|(L_i(q, \llbracket s \rrbracket) - L_{i-1}(q, \llbracket s \rrbracket)) |\psi\rangle\|^2 > 0$ . Thus,  $\lim_{i \rightarrow \infty} \|(L_i(q, \llbracket s \rrbracket) - L_{i-1}(q, \llbracket s \rrbracket)) |\psi\rangle\|^2 = 0$ , which implies that  $\lim_{i \rightarrow \infty} \|(L_i(q, \llbracket s \rrbracket) |\psi\rangle - L_{i-1}(q, \llbracket s \rrbracket) |\psi\rangle)\| = 0$ . This ensures that the sequence  $\{L_i(q, \llbracket s \rrbracket) |\psi\rangle\}_i$  is a Cauchy sequence in  $\mathcal{H}_P$ .  $\square$

We have proved that if  $\{\|L_n(q, \llbracket s \rrbracket) |\psi\rangle\|_n$  is bounded and for all  $|\psi\rangle \in \mathcal{H}_P$ , then  $\{L_n(q, \llbracket s \rrbracket) |\psi\rangle\}_n$  is a Cauchy sequence. Thus by Theorem 4.5 we know that the sequence  $\{L_n(q, \llbracket s \rrbracket)\}$  has a limit. Thus, we use that limit to define the semantics of the quantum loop:

$$\llbracket \text{while } q \text{ do } \{s\} \rrbracket = \lim_{n \rightarrow \infty} L_n(q, \llbracket s \rrbracket).$$

After having defined the semantics as the limit of the sequence, we define the linear operator that corresponds to the limit.

THEOREM 5.10. *Let  $\{|e_i\rangle\}_i$  be the set of standard basis of  $\mathcal{H}_P$ . We call  $\mathcal{L}(q, \llbracket s \rrbracket)$  the linear operator such that,  $\forall |e_i\rangle$ ,  $\mathcal{L}(q, \llbracket s \rrbracket) |e_i\rangle = \lim_{n \rightarrow \infty} L_n(q, \llbracket s \rrbracket) |e_i\rangle$ . The operator  $\mathcal{L}(q, \llbracket s \rrbracket)$  is the limit of the sequence  $\{L_n(q, \llbracket s \rrbracket)\}_n$ .*

PROOF. By definition, for all basis  $|e_i\rangle$  of  $\mathcal{H}_P$ ,  $\mathcal{L}(q, \llbracket s \rrbracket) |e_i\rangle = \lim_{n \rightarrow \infty} \mathcal{L}_n(q, \llbracket s \rrbracket) |e_i\rangle$ , thus  $\|\mathcal{L}_n(q, \llbracket s \rrbracket) |e_i\rangle - \mathcal{L}(q, \llbracket s \rrbracket) |e_i\rangle\| \rightarrow 0$  as  $n \rightarrow \infty$ . Therefore, for all  $|\psi\rangle \in \mathcal{H}_P$ ,  $\|\mathcal{L}_n(q, \llbracket s \rrbracket) |\psi\rangle - \mathcal{L}(q, \llbracket s \rrbracket) |\psi\rangle\| \rightarrow 0$  as  $n \rightarrow \infty$ .  $\square$

## 6 RELATION WITH THE UNITARY SEMANTICS

So far, we have introduced two semantics. The unitary semantics models exactly the behavior of quantum computation, and the linear semantics allows us to define a fix-point and define the semantics of the infinite loops. In this section, we investigate the relationship between these two semantics.

Let us consider the examples introduced in [subsection 4.2](#). First we consider the semantics of the program `while`  $q$  `do`  $\{X(q)\}$ , starting from the state  $|\psi\rangle = 1/\sqrt{2} |0 \dots\rangle_T (|0\rangle_q + |1\rangle_q)$ :

$$\begin{aligned} \mathcal{L}_0(q, X_q) |\psi\rangle &= 0 \\ \mathcal{L}_1(q, X_q) |\psi\rangle &= 1/\sqrt{2} |0 \dots\rangle_T |0\rangle_q \\ \mathcal{L}_2(q, X_q) |\psi\rangle &= 1/\sqrt{2} (|0 \dots\rangle_T |0\rangle_q + |10 \dots\rangle_T |0\rangle_q) \\ \mathcal{L}_3(q, X_q) |\psi\rangle &= 1/\sqrt{2} (|0 \dots\rangle_T |0\rangle_q + |10 \dots\rangle_T |0\rangle_q) \end{aligned} \quad (11)$$

We see that we have ‘collected’ the sub-state that corresponds to the terminating execution, and since the program is fully terminating, the fixpoint is a state with a norm equal to 1. Comparing the examples in the previous sections ([Equation 7](#)), we find clear evidence of this difference. We see that when the loop terminates, the two semantics coincide, specifically  $\mathcal{L}_{n+1} = \mathcal{W}_n$ . In this case, the linear semantics is ‘behind’ the unitary semantics because the latter also includes the component that corresponds to executions that ‘keeps going’.

Then we consider the program `while`  $q$  `do`  $\{H(q)\}$ :

$$\begin{aligned} \mathcal{L}_0(q, H_q) |\psi\rangle &= 0 \\ \mathcal{L}_1(q, H_q) |\psi\rangle &= 1/\sqrt{2} |0 \dots\rangle_T |0\rangle_q \\ \mathcal{L}_2(q, H_q) |\psi\rangle &= 1/\sqrt{2} |0 \dots\rangle_T |0\rangle_q + 1/2 |10 \dots\rangle_T |0\rangle_q \\ \mathcal{L}_3(q, H_q) |\psi\rangle &= 1/\sqrt{2} |0 \dots\rangle_T |0\rangle_q + 1/2 |10 \dots\rangle_T |0\rangle_q + 1/\sqrt{8} |110 \dots\rangle_T |0\rangle_q \\ \mathcal{L}_n(q, H_q) |\psi\rangle &= \sum_{i=1}^n 1/\sqrt{2^i} |1^{\otimes i} 0 \dots\rangle_T |0\rangle_q. \end{aligned} \quad (12)$$

Here, we see that in each approximation  $\mathcal{L}_n$ , we obtain a state with a norm less than 1, which corresponds to the part of the execution that terminates in at most  $n - 1$  iterations. If we compare the linear semantics in [Equation 12](#) with the corresponding unitary semantics in [Equation 8](#), we observe that the linear semantics has a ‘missing’ part in the output states—specifically, the portion of the state representing the execution that ‘keeps going’.

Finally, we consider the while loop `while`  $q$  `do`  $\{skip\}$ , evaluated on  $|0 \dots\rangle_T |1\rangle_q$ ,

$$\begin{aligned} \mathcal{L}_0(q, \mathbf{I}) |0 \dots\rangle_T |1\rangle_q &= 0 \\ \mathcal{L}_1(q, \mathbf{I}) |0 \dots\rangle_T |1\rangle_q &= P_{0_{t_1}} |10 \dots\rangle_T |1\rangle_q = 0 \\ \mathcal{L}_2(q, \mathbf{I}) |0 \dots\rangle_T |1\rangle_q &= P_{0_{t_1}} |10 \dots\rangle_T |1\rangle_q + P_{1_{t_1} 0_{t_2}} |110 \dots\rangle_T |1\rangle_q = 0. \end{aligned} \quad (13)$$

When dealing with a loop that diverges for the whole quantum state, the linear semantics in [Equation 13](#) evaluates to 0, as the linear semantics of a non-terminating loop is the zero operator. On the other hand, the unitary semantics ([Equation 6](#)), compute all partial executions of the divergent loop.

More generally, in all examples, the unitary operator  $\mathcal{W}_n$  returns the portion of the state corresponding to executions that terminate after less than  $n$  iterations, along with the partial results of

computations that are still ongoing. In contrast, the linear semantics only captures the terminating portion of the computation. In fact, by examining the definition of  $\mathbb{W}_n$  (Equation 5), we can identify two key components:

$$P_{0_{t_h}} \cdot \prod_{i=n-h}^{n-2} (\mathbf{G}(q, n-i)S) \cdot \mathbf{G}(q, 1),$$

which is also present in the linear semantics (Equation 10), and the final term is given by:

$$\prod_{i=1}^n P_{1_{t_i}} \cdot \prod_{i=0}^{n-1} S\mathbf{G}(q, n-i).$$

The first part corresponds to computations where the guard becomes 0 after the  $(n-1)$ -th execution, indicating that the loop has finished. The second part represents the non-terminating portion, where the guard remains true, meaning the loop is still in progress. The linear semantics, therefore, returns a substate of the result of the unitary semantics. For this reason, we can consider the linear semantics as an *under-approximation* of the unitary semantics. By discarding part of the result, the linear semantics allows the definition of a limit and, therefore, gives meaning to infinite behaviors.

### 6.1 Non-termination

These examples underlined that the operator  $L_n |\psi\rangle$  returns the state that is the superposition of all executions starting from  $|\psi\rangle$  and terminating in less than  $n$  steps.

Thus the norm  $\|\llbracket \text{while } q \text{ do } \{s\} \rrbracket |\psi\rangle\|$  gives a measure of the portion of the state that diverges. Specifically, if the loop diverges, the semantics will be equal to the zero operator (as in Equation 13). If both terminating and non-terminating executions are in a superposition, the linear semantics will return a state with a norm less than one, and the missing norm value provides a measure of non-termination. For instance, in Equation 12 we see that for all  $n$ ,  $\|L(q, H_q) |\psi\rangle\| < 1$ , and also the norm of the limit, applied to  $|\psi\rangle$  will be less than one. On the other hand, if the program terminates fully, the semantics are defined for all inputs, and the norm  $\|\llbracket \text{while } q \text{ do } \{s\} \rrbracket |\psi\rangle\|$  will be equal to 1 (as we see in Equation 11). In general, the norm  $\|\llbracket \text{while } q \text{ do } \{s\} \rrbracket\|$  will be equal to 1 if there exists at least one input for which the loop terminates while  $\min_{\|\psi\rangle=1} \|\llbracket \text{while } q \text{ do } \{s\} \rrbracket |\psi\rangle\| = 1$  if and only if the loop terminates for all inputs.

## 7 RELATED WORK

So far, various forms of control work have been introduced in quantum programming languages. Here, we present a short review of the different approaches.

*Probabilistic control flow.* The initial works on quantum program semantics focused on languages with quantum data and classical probabilistic control, primarily based on measurement operators. In [18], Selinger introduced the basic notations, theories, and conventions for a quantum programming language with measurement-based probabilistic control flow, called QPL. He provided a denotational semantics for QPL by associating each program with a superoperator (a completely positive map that is not necessarily trace-preserving) in a finite-dimensional Hilbert space, represented as a morphism in a CPO-enriched traced monoidal category.

In [15], Perdrix extended this work by introducing a complete partial order (CPO) over admissible transformations, i.e., multisets of linear operators, and defining a denotational semantics for a simple quantum imperative language similar to QPL. He demonstrated that his semantics is an exact abstraction of Selinger's semantics.

Ying et al., in a series of works [7, 25, 28, 29], explored a quantum while-language with measurement-based probabilistic control flow. They defined a denotational semantics in terms of maps between

density operators, generalizing Selinger’s results to infinite-dimensional Hilbert spaces. In these works, the denotational semantics is constructed using the CPO of superoperators acting on partial density operators. Finally, in [27, Chapter 3], Ying further developed this domain to define the semantics of a quantum language with recursion and measurement-based probabilistic control flow, providing a more comprehensive framework for reasoning about quantum programs.

*Quantum Control.* All the previous works focused on language with probabilistic control flow, thus their semantics is probabilistic and no superposition is introduced between possible executions of the programs. To overcome this limit, we need to introduce quantum control flow.

A first form of quantum control was introduced in Altenkirch and Grattage’s functional language QML [1]. QML is a first-order functional language on finite types that provides a high-level syntax to represent quantum computation. Here, the semantics is defined in terms of morphisms in the category of Finite Quantum Computation, which provides a constructive semantics realisable as quantum gates. In [9], Lampis et al. introduced nQML, a simplified version of QML, where the control constructs of nQML are designed to be simpler. They provide a denotational semantics based on density matrices and unitary transformations.

A formal definition of quantum imperative language with quantum control flow was introduced by Ying et al. in [29, 30], where they define the QuGCL language, i.e. a language with both quantum control flow and measurement-based control flow but no recursion. They provide a semantics for it, on the basis of a new mathematical tool called the guarded composition of operator-valued functions, where operator-valued functions are defined using the Kraus operator-sum representation [14, Chapter 8]. The semantics is then defined using the previous defined CPO of superoperators. However, this semantics is not compositional (only semi-compositional).

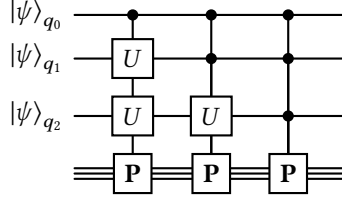
More recently, Valiron has reviewed in [23] quantum control with quantum  $\lambda$ -calculus explaining the difference between the superposition of terms and the superposition of data in the various formulation.

In [31], Yuan et al. studied the problem of automatically compiling self-controlled quantum operations minimizing the use of extra temporary variables and avoiding that these extra temporary variables are entangled with the rest of the variables. In particular, they formalize the conditions under which this compilation is possible. Their compilation technique can be seen as a particular case of our unitary semantics. Inspired by Yuan et al., Zhang and Ying [32] propose a quantum architecture that supports quantum control flows and finite quantum recursion.

*Quantum recursion.* Introducing a construct for quantum control flow in a programming language naturally leads to the concepts of quantum recursion or quantum loops. An initial idea of quantum recursion, based on using an infinite set of external coins, was informally discussed in [29].

The issue of quantum recursion was formally addressed for the first time by Badescu and Panangaden in [4]. They extended the QPL programming language [18] by introducing a quantum if-statement and provided a denotational semantics for this extension based on Kraus decomposition. However, they observed that the semantics of quantum case statements is not monotone with respect to Selinger’s order, concluding that the existing framework is inadequate for modeling quantum recursion. In our paper, we faced a similar problem. However, our work focuses on a language without measurement, demonstrating that even in such a setting, it is impossible to define a limit for the sequence of unitary operators. This highlights the inherent challenges in modeling quantum recursion within a purely unitary framework.

In [17], Sabry et al. start from a classical, typed, reversible language that includes lists and fixpoints and expand the language into a quantum domain. The expanded language is purely quantum, without measurement, allows for linear combinations of terms, and restricts fixpoints to structurally recursive fixpoints. This restriction allows for the representation of quantum programs

Fig. 4.  $\mathbf{P} := \text{if } q \text{ do } \{U(q); \mathbf{P}\}$  circuit

as unitary operators in infinite-dimensional Hilbert spaces. An operational semantics for the quantum language is also provided, following the algebraic  $\lambda$ -calculi principles. This work proves that it is possible to represent with a unitary operator a quantum program with recursion, only we know the fixpoint of the recursive call. In fact, the papers exploit a sort of unfolding of the recursive calls. Thus, our work is more general, since no restriction is imposed on the quantum loop.

Andrés-Martínez, M. et al. [3] model quantum loop using weak measurements. A weak measure is a measurement that gives us partial information about the system on average, but also disturbs the state as little as possible [6]. In this way, they are able to decide when to stop a while loop, limiting the perturbation caused by a measurement at each iteration. In their PhD thesis [2], Andrés-Martínez provides a quantum while language, similar to ours, equipped with a categorical semantics. The thesis extends Haghverdi’s unique decomposition categories—originally introduced to model iteration in classical computation—by addressing their incompatibility with quantum settings. This generalisation establishes connections to topological groups and leads to a hierarchy of categories enriched with infinitary addition and convergence criteria. Building on this foundation, the execution formula is shown to define a valid categorical trace even over categories of quantum processes on infinite-dimensional Hilbert spaces. Intuitively, in the thesis, they assume that input and output are interpreted in terms of plane waves, instead of in the form of particles, which are, in some sense, everywhere at once. In this way, when we execute a quantum loop, it is like all the executions reach the fixed point instantly. **PBS work** These approaches are indeed interesting. On the other hand in our paper, we choose a different approach. We did not change the computation model since our unitary semantics directly correspond to the semantics implementable in a quantum computer, and the linear ones are an approximation of the unitary, which is suitable for studying infinite computations.

In [26, 27], Ying explores a problem similar to ours, considering a recursive quantum language. This work has a stronger similarity with our approach than the other works mentioned above, although the implementation of the self-controlled operation is in a sense ‘dual’ with respect to our model. In fact, instead of making a quantum copy of the guard using a CNOT gate, Ying proposes to prepare and carry along the program an infinite number of copies of identical qubits to represent the guard. However, since we are trying to represent in a unitary way an operation that inherently cannot be unitary, this difference is merely a design choice to address the unitarity constraint.

In this setting, recursion is achieved with programs of the form  $\mathbf{P} := \text{if } q \text{ do } \{U(q); \mathbf{P}\}$ , which can be visually represented in Figure 4. It can be seen that each recursive call consumes a copy of the guard variable, and to achieve infinite recursion, we need an infinite number of copies of the same guard variables. For defining their semantics, Ying et al. employ a formalism from quantum physics related to multi-particle systems, specifically Fock spaces [19] and second quantization [5]. In particular, they consider free Fock spaces, i.e., Hilbert spaces that describe quantum states with a variable number of indistinguishable particles, constructed as  $\oplus_{n=1}^{\infty} \mathcal{H}^{\otimes n}$ , i.e., the direct sum of tensor

powers of the single-particle Hilbert space  $\mathcal{H}$ . Using this formulation, Ying defines a Complete Partial Order (CPO), which orders the operators within the Fock space based on the number of guard copies that these operators act on. As a result, the semantics of a recursive program demonstrates a monotonically continuous order, which, in turn, allows for the existence of a fixed point.

The difference between our approach and Ying's is twofold. Firstly, we adopt a different method for performing self-controlled operations with a unitary operator. The fundamental distinction lies in the formalism that Ying uses for the semantics of mathematical formulations. His approach is capable of handling quantum systems that involve varying quantities of identical particles; moreover, he defines a denotational semantics through second quantization. However, this formalism is based on Fock spaces, which are more abstract compared to the typical semantics used to model circuits. In contrast, we choose to work exclusively within the Hilbert space framework for the program variables. To manage infinite computations, we rely on linear operators, which, as we have demonstrated, are closely related to their corresponding unitary operations. Additionally, our semantics specifically addresses the portion of execution that leads to termination.

In conclusion, both semantics share the key feature of utilising an infinite number of qubits to perform recursion or while loops and the corresponding operator can be defined as a recursive unitary operator, as is intuitively illustrated in Figure 2b and Figure 4. In fact, we can choose to formulate our approach within Ying's Fock space semantics or, conversely, describe Ying's semantics using our Unitary/Linear framework. In the latter case, we can specifically adopt the formulation introduced by Ying in [26, Section 6.4], mapping the Fock space operator to a unitary operator on the program variables space as in Figure 4, and subsequently to a linear operator, in the same way as in section 5.

## 8 CONCLUSION

We have introduced a denotational semantics for quantum programs that approximates their unitary behaviour by separating the finite results from the possibly infinite ones. In the following, we discuss some of the works that are related to ours.

## REFERENCES

- [1] T. Altenkirch and J. Grattage. 2005. A functional quantum programming language. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*. 249–258. <https://doi.org/10.1109/LICS.2005.1>
- [2] Pablo Andrés-Martínez. 2022. Unbounded loops in quantum programs: categories and weak while loops. *arXiv preprint arXiv:2212.05371* (2022).
- [3] Pablo Andrés-Martínez and Chris Heunen. 2022. Weakly measured while loops: peeking at quantum states. *Quantum Science and Technology* 7, 2 (feb 2022), 025007. <https://doi.org/10.1088/2058-9565/ac47f1>
- [4] Costin Badescu and Prakash Panangaden. 2015. Quantum Alternation: Prospects and Problems. *Proceedings 12th International Workshop on Quantum Physics and Logic, QPL 2015, Oxford, UK, July 15-17, 2015* 195 (2015), 33–42. <https://doi.org/10.4204/EPTCS.195.3>
- [5] Michel Baranger. 2003. Many-Body Problems and Quantum Field Theory: An Introduction. *Physics Today* 56, 4 (04 2003), 69–70. <https://doi.org/10.1063/1.1580057> arXiv:[https://pubs.aip.org/physicstoday/article-pdf/56/4/69/11139124/69\\_1\\_online.pdf](https://pubs.aip.org/physicstoday/article-pdf/56/4/69/11139124/69_1_online.pdf)
- [6] Todd A. Brun. 2002. A simple model of quantum trajectories. *American Journal of Physics* 70, 7 (07 2002), 719–737. <https://doi.org/10.1119/1.1475328> arXiv:[https://pubs.aip.org/aapt/ajp/article-pdf/70/7/719/7530993/719\\_1\\_online.pdf](https://pubs.aip.org/aapt/ajp/article-pdf/70/7/719/7530993/719_1_online.pdf)
- [7] Yuan Feng, Runyao Duan, Zhengfeng Ji, and Mingsheng Ying. 2007. Proof rules for the correctness of quantum programs. *Theoretical Computer Science* 386, 1 (2007), 151–166. <https://doi.org/10.1016/j.tcs.2007.06.011>
- [8] Teiko Heinosaari and Mario Ziman. 2008. Guide to mathematical concepts of quantum theory. *Acta Physica Slovaca* 58, 4 (Aug. 2008), 487–674. <https://doi.org/10.2478/v10155-010-0091-y> arXiv:0810.3536 [quant-ph]
- [9] Michael Lampis, Kyriakos G. Ginis, Michalis A. Papakyriakou, and Nikolaos S. Papaspyrou. 2008. Quantum Data and Control Made Easier. *Electron. Notes Theor. Comput. Sci.* 210 (July 2008), 85–105. <https://doi.org/10.1016/j.entcs.2008.04.020>



- [10] Noah Linden and Sandu Popescu. 1998. The halting problem for quantum computers. *arXiv preprint quant-ph/9806054* (1998).
- [11] A. Mattuck. 2013. *Introduction to Analysis*. CreateSpace Independent Publishing Platform. <https://books.google.it/books?id=mYppngEACAAJ>
- [12] Takayuki Miyadera and Masanori Ohya. 2005. On Halting Process of Quantum Turing Machine. *Open Systems & Information Dynamics* 12, 3 (2005), 261–264. <https://doi.org/10.1007/s11080-005-0923-2>
- [13] John M. Myers. 1997. Can a Universal Quantum Computer Be Fully Quantum? *Phys. Rev. Lett.* 78 (Mar 1997), 1823–1824. Issue 9. <https://doi.org/10.1103/PhysRevLett.78.1823>
- [14] Michael A. Nielsen and Isaac L. Chuang. 2010. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press.
- [15] S Perdrix. 2008. A hierarchy of quantum semantics. *Electronic Notes in Theoretical Computer Science* 192, 3 (2008), 71–83.
- [16] Horn Roger and R Johnson Charles. 1991. Topics in matrix analysis.
- [17] Amr Sabry, Benoît Valiron, and Juliana Kaizer Vizzotto. 2018. From Symmetric Pattern-Matching to Quantum Control. In *Foundations of Software Science and Computation Structures*. Springer International Publishing, 348–364. [https://doi.org/10.1007/978-3-319-89366-2\\_19](https://doi.org/10.1007/978-3-319-89366-2_19)
- [18] Peter Selinger. 2004. Towards a quantum programming language. *Mathematical Structures in Computer Science* 14, 4 (2004), 527–586. <https://doi.org/10.1017/S0960129504004256>
- [19] V. S. Shchesnovich. 2013. The second quantization method for indistinguishable particles (Lecture Notes in Physics, UFABC 2010). *arXiv:1308.3275* [cond-mat.quant-gas] <https://arxiv.org/abs/1308.3275>
- [20] Yu Shi. 2002. Remarks on universal quantum computer. *Physics Letters A* 293, 5 (2002), 277–282. [https://doi.org/10.1016/S0375-9601\(02\)00015-4](https://doi.org/10.1016/S0375-9601(02)00015-4)
- [21] Daegene Song. 2008. Unsolvability of the halting problem in quantum dynamics. *International Journal of Theoretical Physics* 47 (2008), 1785–1791.
- [22] Michael Spivak. 2008. *Calculus* (fourth ed.). Publish or Perish.
- [23] Benoît Valiron. 2022. Semantics of quantum programming languages: Classical control, quantum control. *Journal of Logical and Algebraic Methods in Programming* 128 (2022), 100790. <https://doi.org/10.1016/j.jlamp.2022.100790>
- [24] Paolo Vanini. 2017. *Functional Analysis*. Chapter XII Convergence in Infinite Dimensional Spaces. [https://www.researchgate.net/publication/319618010\\_Functional\\_Analysis\\_XII\\_Convergence\\_in\\_Infinite\\_Dimensional\\_Spaces](https://www.researchgate.net/publication/319618010_Functional_Analysis_XII_Convergence_in_Infinite_Dimensional_Spaces)
- [25] Mingsheng Ying. 2012. Floyd–hoare logic for quantum programs. *ACM Trans. Program. Lang. Syst.* 33, 6, Article 19 (January 2012), 49 pages. <https://doi.org/10.1145/2049706.2049708>
- [26] Mingsheng Ying. 2014. Quantum recursion and second quantisation. *arXiv preprint arXiv:1405.4443* (2014).
- [27] Mingsheng Ying. 2016. *Foundations of quantum programming*. Morgan Kaufmann.
- [28] Mingsheng Ying and Yuan Feng. 2010. Quantum loop programs. *Acta Informatica* 47, 4 (2010), 221–250.
- [29] Mingsheng Ying, Nengkun Yu, and Yuan Feng. 2012. Defining Quantum Control Flow. *CoRR* abs/1209.4379 (2012). *arXiv:1209.4379* <http://arxiv.org/abs/1209.4379>
- [30] Mingsheng Ying, Nengkun Yu, and Yuan Feng. 2014. Alternation in Quantum Programming: From Superposition of Data to Superposition of Programs. *CoRR* abs/1402.5172 (2014). *arXiv:1402.5172* <http://arxiv.org/abs/1402.5172>
- [31] Charles Yuan, Agnes Villanyi, and Michael Carbin. 2024. Quantum Control Machine: The Limits of Control Flow in Quantum Programming. *Proc. ACM Program. Lang.* 8, OOPSLA1, Article 94 (April 2024), 28 pages. <https://doi.org/10.1145/3649811>
- [32] Zhicheng Zhang and Mingsheng Ying. 2024. Quantum Register Machine: Efficient Implementation of Quantum Recursive Programs. *arXiv:2408.10054* [quant-ph] <https://arxiv.org/abs/2408.10054>

## A OMITTED PROOFS

PROPOSITION 4.3. *The closed formula of Equation 4, is:*

$$W_n(q, S) = \sum_{h=1}^n \left( \prod_{i=1}^{h-1} (P_{1_{t_i}}) \cdot P_{0_{t_h}} \cdot \prod_{i=n-h}^{n-2} (G(q, n-i) S) \cdot G(q, 1) \right) + \prod_{i=1}^n P_{1_{t_i}} \cdot \prod_{i=0}^{n-1} S G(q, n-i). \quad (5)$$

PROOF. We proceed by induction. If  $n = 0$ ,  $w_0 = I$  and  $w_0 = \sum_{k=1}^0 (\dots) + \prod_{i=1}^0 P_{1_{t_i}} \cdot \prod_{i=0}^{-1} [s] G(q, n-i) = I$ . Let's consider  $w_{n+1} = (P_{0_{t_1}} + P_{1_{t_1}} \cdot S(W_n) \cdot [s]) G(q, 1)$ . By inductive hypothesis  $w_n = W_n$ , thus

we need to compute  $\mathcal{S}(W_n)$ :

$$\mathcal{S}(W_n) = \sum_{k=1}^n \left( \prod_{i=1}^{k-1} (P_{1_{t_{i+1}}}) \cdot P_{0_{t_{k+1}}} \cdot \prod_{i=n-k}^{n-2} (\mathbf{G}(q, n-i+1)[s]) \cdot \mathbf{G}(q, 2) \right) + \prod_{i=1}^n P_{1_{t_{i+1}}} \cdot \prod_{i=0}^{n-1} [s] \mathbf{G}(q, n-i+1).$$

The initial and final values of the first and third products can be updated to remove the +1 from the indices, resulting in:

$$\mathcal{S}(W_n) = \sum_{k=1}^n \left( \prod_{i=2}^{(k+1)-1} (P_{1_{t_i}}) \cdot P_{0_{t_{k+1}}} \cdot \prod_{i=n-k}^{n-2} (\mathbf{G}(q, (n+1)-i)[s]) \cdot \mathbf{G}(q, 2) \right) + \prod_{i=2}^{n+1} P_{1_{t_i}} \cdot \prod_{i=0}^{n-1} [s] \mathbf{G}(q, (n+1)-i).$$

Finally, the  $k+1$  can be collected, and the summation indices can be updated, resulting in:

$$\mathcal{S}(W_n) = \sum_{k=2}^{n+1} \left( \prod_{i=2}^{k-1} (P_{1_{t_i}}) \cdot P_{0_{t_k}} \cdot \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1)-i)[s]) \cdot \mathbf{G}(q, 2) \right) + \prod_{i=2}^{n+1} P_{1_{t_i}} \cdot \prod_{i=0}^{n-1} [s] \mathbf{G}(q, (n+1)-i).$$

Now we substitute this equation in  $w_{n+1} = P_{0_{t_1}} \cdot \mathbf{G}(q, 1) + P_{1_{t_1}} \cdot \mathcal{S}(W_n) \cdot [s] \cdot \mathbf{G}(q, 1)$ . Initially, we compute  $w' = P_{1_{t_1}} \cdot \mathcal{S}(W_n)$ , in particular:

$$\begin{aligned} w' &= P_{1_{t_1}} \left( \sum_{k=2}^{n+1} \left( \prod_{i=2}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \cdot \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1)-i)[s]) \mathbf{G}(q, 2) \right) + \prod_{i=2}^{n+1} P_{1_{t_i}} \prod_{i=0}^{n-1} [s] \mathbf{G}(q, (n+1)-i) \right) \\ &= \sum_{k=2}^{n+1} \left( P_{1_{t_1}} \prod_{i=2}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \cdot \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1)-i)[s]) \mathbf{G}(q, 2) \right) + P_{1_{t_1}} \prod_{i=2}^{n+1} P_{1_{t_i}} \prod_{i=0}^{n-1} [s] \mathbf{G}(q, (n+1)-i) \\ &= \sum_{k=2}^{n+1} \left( \prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \cdot \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1)-i)[s]) \mathbf{G}(q, 2) \right) + \prod_{i=1}^{n+1} P_{1_{t_i}} \prod_{i=0}^{n-1} [s] \mathbf{G}(q, (n+1)-i). \end{aligned}$$

Secondly, let's consider  $w'' = w' \cdot [s] \cdot \mathbf{G}(q, 1)$ , it result in:

$$\begin{aligned} w'' &= \left( \sum_{k=2}^{n+1} \left( \prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \cdot \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1)-i)[s]) \mathbf{G}(q, 2) \right) \right. \\ &\quad \left. + \prod_{i=1}^{n+1} P_{1_{t_i}} \prod_{i=0}^{n-1} [s] \mathbf{G}(q, (n+1)-i) \right) [s] \mathbf{G}(q, 1) \\ &= \sum_{k=2}^{n+1} \left( \prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \cdot \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1)-i)[s]) \mathbf{G}(q, 2) [s] \mathbf{G}(q, 1) \right) \\ &\quad + \prod_{i=1}^{n+1} P_{1_{t_i}} \prod_{i=0}^{n-1} [s] \mathbf{G}(q, (n+1)-i) [s] \mathbf{G}(q, 1) \end{aligned}$$

If we consider the second and fourth products, we can include  $\mathbf{G}(q, 2)[s]$  by updating the indices. Specifically, in the first product, when  $i = n-2$ , we have  $n+1-n+2 = 3$ . Therefore, by setting  $n-1$  as the upper limit of the product, we can include  $\mathbf{G}(q, 2)[s]$  in the product. Similarly, in the fourth product, when  $i = n-1$ ,  $[s] \mathbf{G}(q, 2)$  is included. By varying the product from 0 to  $n$ , we also include  $[s] \mathbf{G}(q, 1)$ . So finally, we can write  $w''$  as:

$$w'' = \sum_{k=2}^{n+1} \left( \prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \cdot \prod_{i=n-(k-1)}^{n-1} (\mathbf{G}(q, (n+1)-i)[s]) \mathbf{G}(q, 1) \right) + \prod_{i=1}^{n+1} P_{1_{t_i}} \prod_{i=0}^n [s] \mathbf{G}(q, (n+1)-i).$$



Finally, since  $W_{n+1} = P_{0_{t_1}} \mathbf{G}(q, 1) + W''$  and  $(\prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \prod_{i=n-(k-1)}^{n-1} (\mathbf{G}(q, (n+1)-i) [s]) \mathbf{G}(q, 1))_{k=1} = P_{0_{t_1}} \cdot \mathbf{G}(q, 1)$  we can write:

$$W_{n+1} = \sum_{k=1}^{n+1} (\prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \prod_{i=(n+1)-k}^{n-1} (\mathbf{G}(q, (n+1)-i) [s]) \mathbf{G}(q, 1)) + \prod_{i=1}^{n+1} P_{1_{t_i}} \prod_{i=0}^n [s] \mathbf{G}(q, (n+1)-i),$$

i.e., we write  $W_{n+1}$  in the form of [Equation 5](#).

□

PROPOSITION 5.3. *The closed formula of [Equation 9](#) is:*

$$L_n(q, S) = \sum_{k=1}^n (\prod_{i=1}^{k-1} (P_{1_{t_i}}) \cdot P_{0_{t_k}} \cdot \prod_{i=n-k}^{n-2} (\mathbf{G}(q, n-i) \cdot S) \cdot \mathbf{G}(q, 1)). \quad (10)$$

PROOF. If  $n = 0$ ,  $l_0 = 0$  and  $L = \sum_{k=1}^0 (\dots) = 0$ . By inductive hypothesis,

$$l_n = \sum_{k=1}^n (\prod_{i=1}^{k-1} (P_{1_{t_i}}) \cdot P_{0_{t_k}} \cdot \prod_{i=n-k}^{n-2} (\mathbf{G}(q, n-i) [s]) \cdot \mathbf{G}(q, 1)).$$

Let's consider  $l_{n+1} = (P_{0_{t_1}} + P_{1_{t_1}} \cdot S(l_n) \cdot [s]) \mathbf{G}(q, 1)$ . By inductive hypothesis  $l_n = L_n$ . First we compute  $S(L_n)$ , in particular,

$$S(L_n) = \sum_{k=1}^n (\prod_{i=1}^{k-1} (P_{1_{t_{i+1}}}) \cdot P_{0_{t_{k+1}}} \cdot \prod_{i=n-k}^{n-2} (\mathbf{G}(q, n-i+1) [s]) \cdot \mathbf{G}(q, 2)).$$

The initial and final values of the first product can be updated to remove the +1 from the indices, resulting in:

$$S(L_n) = \sum_{k=1}^n (\prod_{i=2}^{(k+1)-1} (P_{1_{t_i}}) \cdot P_{0_{t_{k+1}}} \cdot \prod_{i=n-k}^{n-2} (\mathbf{G}(q, (n+1)-i) [s]) \cdot \mathbf{G}(q, 2)).$$

Then, the  $k+1$  can be collected, and the summation indices can be updated, resulting in the following:

$$S(L_n) = \sum_{k=2}^{n+1} (\prod_{i=2}^{k-1} (P_{1_{t_i}}) \cdot P_{0_{t_k}} \cdot \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1)-i) [s]) \cdot \mathbf{G}(q, 2)).$$

Now we substitute this equation in  $l_{n+1} = P_{0_{t_1}} \cdot \mathbf{G}(q, 1) + P_{1_{t_1}} \cdot S(L_n) \cdot [s] \cdot \mathbf{G}(q, 1)$ . Initially, we compute  $l' = P_{1_{t_1}} \cdot S(L_n)$ , in particular:

$$\begin{aligned} l' &= P_{1_{t_1}} \left( \sum_{k=2}^{n+1} \prod_{i=2}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1)-i) [s]) \mathbf{G}(q, 2) \right) \\ &= \sum_{k=2}^{n+1} (P_{1_{t_1}} \prod_{i=2}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1)-i) [s]) \mathbf{G}(q, 2)) \\ &= \sum_{k=2}^{n+1} (\prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1)-i) [s]) \mathbf{G}(q, 2)). \end{aligned}$$

Secondly, let's consider  $1'' = 1' \cdot \llbracket s \rrbracket \cdot \mathbf{G}(q, 1)$ , it result in:

$$\begin{aligned} 1'' &= \left( \sum_{k=2}^{n+1} \prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1) - i) \llbracket s \rrbracket) \mathbf{G}(q, 2) \right) \llbracket s \rrbracket \mathbf{G}(q, 1) = \\ &= \sum_{k=2}^{n+1} \prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \prod_{i=n-(k-1)}^{n-2} (\mathbf{G}(q, (n+1) - i) \llbracket s \rrbracket) \mathbf{G}(q, 2) \llbracket s \rrbracket \mathbf{G}(q, 1). \end{aligned}$$

If we consider the second product, we can include  $\mathbf{G}(q, 2) \llbracket s \rrbracket$  by updating the indices. Specifically, when  $i = n - 2$ ,  $n + 1 - n + 2 = 3$ , therefore, by setting  $n - 1$  as the upper limit of the product, we can include  $\mathbf{G}(q, 2) \llbracket s \rrbracket$  in the product. So finally, we can write  $1''$  as:

$$1'' = \sum_{k=2}^{n+1} \prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \prod_{i=n-(k-1)}^{n-1} (\mathbf{G}(q, (n+1) - i) \llbracket s \rrbracket) \mathbf{G}(q, 1).$$

Finally, since  $1_{n+1} = P_{0_{t_1}} \mathbf{G}(q, 1) + 1''$  and  $(\prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \prod_{i=n-(k-1)}^{n-1} (\mathbf{G}(q, (n+1) - i) \llbracket s \rrbracket) \mathbf{G}(q, 1))_{k=1} = P_{0_{t_1}} \cdot \mathbf{G}(q, 1)$  we can write:

$$1_{n+1} = \sum_{k=1}^{n+1} \prod_{i=1}^{k-1} (P_{1_{t_i}}) P_{0_{t_k}} \prod_{i=(n+1)-k}^{n-1} (\mathbf{G}(q, (n+1) - i) \llbracket s \rrbracket) \mathbf{G}(q, 1),$$

i.e., we have written  $1_{n+1}$  in the form of [Equation 10](#).

□