# 5 QWhile Language

This section introduces the core programming language we will use for quantum program logics. The language follows the *classical control + quantum data* paradigm: the program's control flow (sequencing, branching, looping, and the program counter itself) is entirely classical, while the data manipulated by commands are quantum states (density operators) on a fixed composite Hilbert space. Concretely:

*Control is classical.* The program counter is never put into superposition. The only source of probabilistic branching is *measurement*: a measurement produces a classical outcome, and the next command is chosen based on that outcome.

*Data is quantum.* At every point, the program carries a (possibly subnormalized) density operator $\rho$ on a fixed global Hilbert space $H$. Primitive commands act *locally* on specified subsystems, leaving the rest untouched (up to the standard identity extension).

**Registers and subsystem structure.** Fix a finite set of register labels $\mathsf{Reg}$ and a family of finite-dimensional Hilbert spaces $\{H_x\}_{x\in\mathsf{Reg}}$. The *global* state space is the tensor product

$$H \;\cong\; \bigotimes_{x\in\mathsf{Reg}} H_x.$$

A *subsystem* is a subset of labels $s \subseteq \mathsf{Reg}$. We write

$$H_s \;\cong\; \bigotimes_{x\in s} H_x, \qquad \overline{s} := \mathsf{Reg}\setminus s, \qquad H \cong H_s \otimes H_{\overline{s}}.$$

The purpose is to make precise the standard programming intuition that a program acts on *named registers* and typically touches only *finitely many* of them. A concrete quantum program is written in terms of a finite collection of variables/registers/qubits, such as $q_1, q_2, \ldots, q_n$, and every primitive command (unitary gates, measurements, initialization) targets a specified subset of these registers. Thus it is natural to model the program's data state as a single density operator on a tensor product space that factors according to these register boundaries. Moreover, working with explicit subsystems is essential for two semantic reasons:

*Locality of commands.* A command that is declared to act on subsystem $s$ should leave the rest of the machine state unchanged; this is expressed later by cylindrical extension $(A^{(s)} = A \otimes \mathbf{1}_{H_{\overline{s}}})$.

*Discarding and re-initialization.* Initialization/reset operations need to "forget" whatever was previously stored in a register (including any entanglement with other registers). This is naturally described using the decomposition $H \cong H_s \otimes H_{\overline{s}}$ together with partial trace.

We tacitly fix a tensor-factor ordering/parenthesization convention so that expressions such as $H \cong H_s \otimes H_{\overline{s}}$ are well-defined (up to canonical unitary isomorphism). This lets us speak cleanly about commands that act only on $s$ while leaving $\overline{s}$ unchanged.

**Program states (subnormalized density operators).** To represent probabilistic branching without constantly renormalizing, we use *subnormalized* (partial) density operators:

$$\mathcal{D}^-(H) \;:=\; \{\rho \in L(H) \;:\; \rho \succeq 0 \;\wedge\; \mathrm{tr}(\rho) \le 1\}.$$

Intuitively, $\mathrm{tr}(\rho)$ is the *probability mass* of reaching $\rho$ along a particular execution path. Thus the same symbolic $\rho$ can simultaneously encode the post-state *and* the probability of being in that post-state. The zero operator 0 is allowed and represents an impossible (unreachable) state.

**Local operators and cylindrical extension.** A key design principle is *locality*: commands target a subsystem $s \subseteq \mathsf{Reg}$ and act as the identity on the complement $\bar{s}$. If $A \in L(H_s)$ is a linear operator acting on subsystem $s$, its *cylindrical extension* (identity extension) to the global space is

$$A^{(s)} := A \otimes \mathbf{1}_{H_{\bar{s}}} \quad \text{(under the fixed identification } H \cong H_s \otimes H_{\bar{s}}).$$

Likewise, if $U_s$ is unitary on $H_s$, then $U_s^{(s)}$ is unitary on $H$ and represents "apply $U_s$ on subsystem $s$ and do nothing elsewhere."

We also need a primitive way to *discard* part of a composite system. For $\rho \in L(H)$ and $s \subseteq \mathsf{Reg}$, write $\mathrm{tr}_s(\rho)$ for the partial trace over $H_s$. Then $\mathrm{tr}_s(\rho) \in L(H_{\bar{s}})$ and preserves subnormalization:

$$\mathrm{tr}(\mathrm{tr}_s(\rho)) = \mathrm{tr}(\rho).$$

Operationally, $\mathrm{tr}_s$ means "throw away subsystem $s$ and keep only the reduced state of the rest." A key operation in qwhile is to *discard* the old content of subsystem $s$ and replace it by a fresh local state $\rho_s$ on $H_s$. Intuitively, we first "throw away" subsystem $s$ by taking the partial trace $\mathrm{tr}_s(\rho)$, and then prepare a new state $\rho_s$ on $s$. As a result, the register $s$ is set to $\rho_s$ regardless of its previous contents, and any entanglement between $s$ and $\bar{s}$ is removed. This is the subsystem-level generalization of the familiar qubit reset command $q := |0\rangle$.

## 5.1 QWhile Language

We now define a core language, *qwhile*, that is expressive enough to support quantum Hoare-style correctness reasoning and under-approximate incorrectness reasoning. The distinctive features are: (i) commands act on designated subsystems; (ii) branching/looping is controlled by measurement outcomes; (iii) the statement error models an explicit abnormal termination.

**Definition 5.1.** Fix a global register set $\mathsf{Reg}$ and global space $H \cong \bigotimes_{x \in \mathsf{Reg}} H_x$. qwhile commands are generated by the following grammar:

$$
\begin{aligned}
C \in \mathsf{Cmd} \quad ::= \quad &\mathsf{error} \mid \mathsf{skip} \mid C_1; C_2 \mid \mathsf{init}\ \rho_s \mid \mathsf{apply}\ U_s \\
&\mid \mathsf{if}\ (\square m.\ M_s = m \to C_m)\ \mathsf{fi} \mid \mathsf{while}\ M'_s = 1\ \mathsf{do}\ C\ \mathsf{od}.
\end{aligned}
$$

Here: $s \subseteq \mathsf{Reg}$ ranges over subsystems. $\rho_s$ ranges over density operators on $H_s$ (i.e. $\rho_s \succeq 0$, $\mathrm{tr}(\rho_s) = 1$). $U_s$ ranges over unitary operators on $H_s$. $M_s = \{(m, M_m)\}_{m \in \mathsf{Out}(M_s)}$ is a measurement on $H_s$ in Kraus form, i.e. $\sum_m M_m^\dagger M_m = \mathbf{1}_{H_s}$. $M'_s = \{M_0, M_1\}$ is a two-outcome measurement on $H_s$ (a special case of the above).

skip terminates normally and leaves the state unchanged.

error terminates *abnormally*. It is intended to model bug signals such as failed runtime checks. Crucially, once error occurs, the program stops immediately: any remaining code is discarded rather than executed.

$C_1; C_2$ is sequential composition: execute $C_1$ first; if $C_1$ terminates normally then continue with $C_2$; if $C_1$ terminates abnormally then the whole composition terminates abnormally.

init $\rho_s$ *resets* subsystem $s$ to $\rho_s$, discarding any prior content of $s$ (including entanglement with $\bar{s}$). Operationally, it applies $\rho \mapsto \rho_s \otimes \mathrm{tr}_s(\rho)$.

apply $U_s$ applies the unitary $U_s$ to subsystem $s$ (lifted cylindrically to $H$), i.e. $\rho \mapsto U_s^{(s)} \rho \left(U_s^{(s)}\right)^\dagger$.

if$(\square m.\ M_s = m \to C_m)$fi first measures subsystem $s$ using the measurement $M_s$. The (classical) outcome $m$ selects the branch $C_m$, and the quantum state is updated by the corresponding Kraus operator $M_m$.

while $M'_s = 1$ do $C$ od repeatedly measures subsystem $s$ using $M'_s = \{M_0, M_1\}$. Outcome 0 terminates the loop; outcome 1 executes the body $C$ and repeats.

## 5.2 Operational semantics

The semantics is given as a labelled transition system on configurations. Exit conditions are

$$\epsilon ::= \mathsf{ok} \mid \mathsf{er},$$

where $\mathsf{ok}$ denotes normal steps/termination and $\mathsf{er}$ denotes abnormal termination caused by error. We call outputs of $\mathsf{ok}$-terminations *normal states* and outputs of $\mathsf{er}$-terminations *abnormal states*.

A *configuration* is a pair $\langle C, \rho \rangle$ where $C$ is the remaining code to be executed (or $\downarrow$ to denote termination by convention) and $\rho \in \mathcal{D}^-(H)$ is the current program state. The one-step transition relation is written

$$\langle C, \rho \rangle \xrightarrow{\epsilon} \langle C', \rho' \rangle.$$

Because measurements branch on outcomes, a configuration may have multiple $\mathsf{ok}$-successors, each carrying a different (subnormalized) post-measurement state.

**Conventions.** Whenever an operator is defined only on a subsystem $s$, it is understood to act on the whole space via cylindrical extension (e.g. $U_s^{(s)}$ and $M_m^{(s)}$). All rules below are to be understood under the fixed identification $H \cong H_s \otimes H_{\bar{s}}$ whenever subsystem $s$ is involved, and with cylindrical extensions $A^{(s)} := A \otimes \mathbf{1}_{H_{\bar{s}}}$.

$$\frac{}{\langle \mathsf{skip}, \rho \rangle \xrightarrow{\mathsf{ok}} \langle \downarrow, \rho \rangle} \text{(Skip)} \qquad \frac{}{\langle \mathsf{error}, \rho \rangle \xrightarrow{\mathsf{er}} \langle \downarrow, \rho \rangle} \text{(Error)}$$

The Error rule is the essence of abnormal termination: it stops execution immediately, raises label $\mathsf{er}$, and returns the current quantum state unchanged.

$$\frac{\langle C_1; C_2, \rho \rangle \xrightarrow{\mathsf{ok}} \langle C_2, \rho' \rangle}{\langle C_1, \rho \rangle \xrightarrow{\mathsf{ok}} \langle \downarrow, \rho' \rangle} \text{(Seq-Done)} \qquad \frac{\langle C_1; C_2, \rho \rangle \xrightarrow{\mathsf{ok}} \langle C_1'; C_2, \rho' \rangle}{\langle C_1, \rho \rangle \xrightarrow{\mathsf{ok}} \langle C_1', \rho' \rangle} \text{(Seq-Step)}$$

$$\frac{\langle C_1; C_2, \rho \rangle \xrightarrow{\mathsf{er}} \langle \downarrow, \rho' \rangle}{\langle C_1, \rho \rangle \xrightarrow{\mathsf{er}} \langle \downarrow, \rho' \rangle} \text{(Seq-Err)}$$

These sequencing rules express two behaviors: (i) normal sequencing proceeds by stepping $C_1$ until it finishes, then continues with $C_2$; (ii) $\mathsf{er}$ *short-circuits* sequencing: if $C_1$ terminates abnormally, then the whole program terminates abnormally and $C_2$ is discarded.

$$\frac{}{\langle \mathsf{init}\ \rho_s, \rho \rangle \xrightarrow{\mathsf{ok}} \langle \downarrow, \rho_s \otimes \mathrm{tr}_s(\rho) \rangle} \text{(Init)} \qquad \frac{}{\langle \mathsf{apply}\ U_s, \rho \rangle \xrightarrow{\mathsf{ok}} \langle \downarrow, U_s^{(s)} \rho \left(U_s^{(s)}\right)^\dagger \rangle} \text{(Apply)}$$

The Init rule makes explicit that initialization *forgets* whatever was stored in $s$ by taking $\mathrm{tr}_s(\rho)$ and then prepares $\rho_s$ afresh. The Apply rule is local unitary evolution: it preserves trace, so $\mathrm{tr}(\rho)$ (the path weight) is unchanged.

$$\frac{\langle \mathsf{if}\ (\square m.\ M_s = m \to C_m)\ \mathsf{fi}, \rho \rangle \xrightarrow{\mathsf{ok}} \langle C_m, M_m^{(s)} \rho \left(M_m^{(s)}\right)^\dagger \rangle}{m \in \mathsf{Out}(M_s)} \text{(If)}$$

This rule formalizes *measurement-controlled branching*. For each possible outcome $m$, there is a transition to the corresponding branch $C_m$. The post-measurement state is subnormalized: its trace equals the probability mass of seeing outcome $m$ on input $\rho$.

$$\langle \text{while } M'_s = 1 \text{ do } C \text{ od}, \rho \rangle \xrightarrow{\text{ok}} \langle \downarrow, M_0^{(s)} \rho \left(M_0^{(s)}\right)^\dagger \rangle \quad (\text{WHILE-0})$$

$$\langle \text{while } M'_s = 1 \text{ do } C \text{ od}, \rho \rangle \xrightarrow{\text{ok}} \langle C; \text{ while } M'_s = 1 \text{ do } C \text{ od}, M_1^{(s)} \rho \left(M_1^{(s)}\right)^\dagger \rangle \quad (\text{WHILE-1})$$

The loop is driven by a two-outcome measurement. Outcome 0 terminates the loop immediately; outcome 1 unrolls one iteration: execute $C$ and then repeat the loop.

**Comments on normalization and probability.** The rules intentionally avoid renormalization. For example, $M_m^{(s)} \rho (M_m^{(s)})^\dagger$ is not divided by its trace. Instead, $\text{tr}\left(M_m^{(s)} \rho (M_m^{(s)})^\dagger\right)$ *is* the probability mass of taking outcome $m$ from state $\rho$. This is why subnormalized states are convenient: the branching structure is classical (one successor per outcome), while the quantitative weights are carried by the traces of the resulting partial states.

Write $\xrightarrow{\epsilon}^*$ for the reflexive-transitive closure of $\xrightarrow{\epsilon}$, where the label records whether an abnormal termination occurred:

$\langle C, \rho \rangle \xrightarrow{\text{ok}}^* \langle C', \rho' \rangle$ means: execute zero or more steps, and no step is labelled er.

$\langle C, \rho \rangle \xrightarrow{\text{er}}^* \langle \downarrow, \rho' \rangle$ means: along the execution, an er-labelled termination occurs. By SEQ-ERR, once er happens, the remaining code is discarded and the computation ends immediately.

This separation of ok vs. er executions is essential for later incorrectness-style specifications: it allows us to talk about *normal outcomes* and *error outcomes* of the same program, and to treat the existence of an er-path as evidence of a bug.