Classical Foundations

1 Core classical setting

Understanding quantum verification requires a firm grasp of classical program logic. Traditionally, Hoare Logic (HL) ensures correctness by using *over-approximation* to prove the *absence* of bugs. In contrast, the more recent Incorrectness Logic (IL) provides a dual framework: it uses *under-approximation* to prove the *presence* of bugs.

Despite their differing goals, both logics are anchored in the same underlying mathematical meaning of a program. This section fixes the classical programming model that serves as the baseline for both correctness and reachability reasoning. By defining a formal semantics, we ensure that the "meaning" of a program remains consistent, whether we are proving it right or proving it wrong.

Specifically, we develop this foundation through three complementary lenses:

(i) **Syntactic Foundation:** We introduce a standard *while-language*, providing a minimalist yet computationally complete syntax for imperative programs.

(ii) **Relational Semantics:** We define the program's behavior using big-step operational semantics. This allows us to treat a program as a mathematical relation between initial and final states, capturing the transformation of the state space.

(iii) **Predicate Transformers:** We shift to a functional view of program meaning via the *post-image operator* (often termed the *strongest postcondition*). This view is particularly vital for Incorrectness Logic, as it characterizes the set of states that are reachable from a given precondition.

Throughout this section, we restrict our attention to a deterministic setting—where assignments, conditionals, and loops follow a unique execution path. This choice maintains clarity in the presentation of the classical proof rules and mirrors the core structures we will later generalize. As we transition to the quantum domain, these concepts of branching, iteration, and fixed-point semantics will reappear as the building blocks for reasoning about density matrices and super-operators.

1.1 Classical While-Language & State Space

Definition 1.1 (Variables, values, and states)**.** Fix a countable set of program variables Var and a value domain Val (for concreteness, integers $\mathbb{Z}$). A *state* is a total mapping

$$\Sigma \triangleq \mathsf{Var} \to \mathsf{Val}.$$

We write $\sigma \in \Sigma$ for a state and $\sigma(x)$ for the value of variable x in σ. State update is written $\sigma[x \mapsto v]$, meaning the state identical to σ except that it maps x to v.

Definition 1.2 (Expressions)**.** Let AExp and BExp denote the sets of arithmetic and boolean expressions. A typical minimal grammar is:

$$\begin{aligned} a \in \mathsf{AExp} &::= n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \cdot a_2 \\ b \in \mathsf{BExp} &::= \mathtt{true} \mid \mathtt{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2. \end{aligned}$$

(Any standard expression language works; the specific operators are not crucial.)

Definition 1.3 (Commands)**.** Commands of the while-language are generated by:

$$c \in \mathsf{Cmd} ::= \mathtt{skip} \mid x := a \mid c_1; c_2 \mid \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{while}\ b\ \mathtt{do}\ c.$$

Intuitively: `skip` does nothing, assignment updates a variable, sequencing composes programs, conditionals branch on a boolean, and while-loops iterate.

Definition 1.4 (Expression evaluation)**.** We assume standard total evaluation functions

$$[\![a]\!] : \Sigma \to \mathsf{Val}, \qquad [\![b]\!] : \Sigma \to \{\mathsf{tt}, \mathsf{ff}\},$$

defined inductively on the syntax of expressions (e.g. $[\![x]\!](\sigma) = \sigma(x)$, $[\![a_1 + a_2]\!](\sigma) = [\![a_1]\!](\sigma) + [\![a_2]\!](\sigma)$, etc.).

1.2 Operational Semantics and Induced Relational Semantics

Definition 1.5 (Big-step operational semantics)**.** We define a big-step evaluation judgement

$$\langle c, \sigma \rangle \Downarrow \sigma'$$

to mean: running command c from initial state σ terminates normally in final state σ'. Divergence (non-termination) is represented by the absence of any derivation of $\langle c, \sigma \rangle \Downarrow \sigma'$.

The rules below define $\Downarrow$ inductively.

$$\frac{}{\langle \mathtt{skip}, \sigma \rangle \Downarrow \sigma} \qquad \frac{}{\langle x := a, \sigma \rangle \Downarrow \sigma[x \mapsto [\![a]\!](\sigma)]}$$

$$\frac{\langle c_1, \sigma \rangle \Downarrow \sigma_1 \qquad \langle c_2, \sigma_1 \rangle \Downarrow \sigma_2}{\langle c_1; c_2, \sigma \rangle \Downarrow \sigma_2}.$$

$$\frac{[\![b]\!](\sigma) = \mathtt{tt} \qquad \langle c_1, \sigma \rangle \Downarrow \sigma'}{\langle \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2, \sigma \rangle \Downarrow \sigma'} \qquad \frac{[\![b]\!](\sigma) = \mathsf{ff} \qquad \langle c_2, \sigma \rangle \Downarrow \sigma'}{\langle \mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2, \sigma \rangle \Downarrow \sigma'}.$$

$$\frac{[\![b]\!](\sigma) = \mathsf{ff}}{\langle \mathtt{while}\ b\ \mathtt{do}\ c, \sigma \rangle \Downarrow \sigma}$$

$$\frac{[\![b]\!](\sigma) = \mathtt{tt} \qquad \langle c, \sigma \rangle \Downarrow \sigma_1 \qquad \langle \mathtt{while}\ b\ \mathtt{do}\ c, \sigma_1 \rangle \Downarrow \sigma_2}{\langle \mathtt{while}\ b\ \mathtt{do}\ c, \sigma \rangle \Downarrow \sigma_2}.$$

These are the standard big-step rules: if the guard is false, the loop terminates immediately; if the guard is true, execute the body once and then iterate.

In this simple language, evaluation is deterministic: for each c and σ, there is at most one σ' such that $\langle c, \sigma \rangle \Downarrow \sigma'$. This can be proved by a straightforward induction on derivations. This deterministic model will be extended to incorporate non-determinism when we establish the foundations of Incorrectness Logic.

1.3 Relational Semantics

The big-step operational semantics defined in the previous section induces a *relational* meaning for commands. Rather than focusing on the step-by-step derivation, we view a program as a mathematical object that relates input states to output states.

Definition 1.6. The relational semantics of a command c, denoted by $[\![c]\!]$, is a binary relation on states $[\![c]\!] \subseteq \Sigma \times \Sigma$ defined by:

$$[\![c]\!] \triangleq \{(\sigma, \sigma') \mid \langle c, \sigma \rangle \Downarrow \sigma'\}.$$

We write $\sigma[\![c]\!]\sigma'$ as shorthand for $(\sigma, \sigma') \in [\![c]\!]$.

In this view, the semantics of individual commands can be expressed directly as sets of state pairs:

$$[\![\texttt{skip}]\!] = \{(\sigma, \sigma) \mid \sigma \in \Sigma\} = \mathrm{Id}_\Sigma.$$

$$[\![x := a]\!] = \{(\sigma, \sigma[x \mapsto [\![a]\!](\sigma)]) \mid \sigma \in \Sigma\}.$$

Let $\Sigma_b = \{\sigma \mid [\![b]\!](\sigma) = \texttt{tt}\}$ and $\Sigma_{\neg b} = \{\sigma \mid [\![b]\!](\sigma) = \texttt{ff}\}$. Then:

$$[\![\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2]\!] = (\mathrm{Id}_{\Sigma_b}; [\![c_1]\!]) \cup (\mathrm{Id}_{\Sigma_{\neg b}}; [\![c_2]\!])$$

where $\mathrm{Id}_S \triangleq \{(\sigma, \sigma) \mid \sigma \in S\}$ is the identity relation restricted to the set S, and ; denotes relational composition (see below).

Because the underlying language is deterministic, the relation $[\![c]\!]$ is *functional* (or a partial function): for every σ, there is at most one σ' such that $\sigma[\![c]\!]\sigma'$. This ensures that for any initial state, the program either diverges (no related σ') or terminates in a unique state.

1.4 Compositional Properties

A critical advantage of the relational view is its *compositionality.* The meaning of a complex program can be constructed from the meanings of its components using standard relational algebra.

Relational Composition. Recall that for two relations $R_1, R_2 \subseteq \Sigma \times \Sigma$, their composition $R_1; R_2$ is defined as:

$$R_1; R_2 \triangleq \{(\sigma, \sigma'') \mid \exists \sigma'. (\sigma, \sigma') \in R_1 \wedge (\sigma', \sigma'') \in R_2\}.$$

The semantics of sequential execution coincides exactly with this algebraic composition:

$$[\![c_1; c_2]\!] = [\![c_1]\!]; [\![c_2]\!]. \tag{1}$$

Sometimes, we might write $R_2 \circ R_1$ instead. Relational composition is monotonic with respect to set inclusion. If $[\![c_1]\!] \subseteq [\![c_1']\!]$, then $([\![c_1]\!]; [\![c_2]\!]) \subseteq ([\![c_1']\!]; [\![c_2]\!])$. This property is foundational for Incorrectness Logic, which relies on the fact that an *under-approximation* of a command's relation results in a valid under-approximation of the composed program.

Iteration and Fixed Points. The semantics of the `while` loop is more complex as it involves recursion. Relationally, $[\![\texttt{while } b \texttt{ do } c]\!]$ is the smallest relation R satisfying the following recursive equation:

$$R = (\mathrm{Id}_{\Sigma_{\neg b}}) \cup (\mathrm{Id}_{\Sigma_b}; [\![c]\!]; R).$$

Formally, we define the loop semantics as the limit of a sequence of relations representing finite unrollings:

(1) $R_0 = \emptyset$ (no executions).

(2) $R_{n+1} = (\mathrm{Id}_{\Sigma_{\neg b}}) \cup (\mathrm{Id}_{\Sigma_b}; [\![c]\!]; R_n)$.

Then $[\![\texttt{while } b \texttt{ do } c]\!] = \bigcup_{n=0}^{\infty} R_n$. This construction establishes that a state pair is in the loop relation if and only if there exists some finite number of iterations n that leads from the input to the output, terminating when the guard b becomes false.

1.5 Predicates, post-images, and strongest postconditions

This subsection introduces the semantic object that will play the central role in both correctness and incorrectness reasoning: the *post-image* (also known as the strongest postcondition).

Definition 1.7. A predicate is a set of states $P \subseteq \Sigma$. Equivalently, one may write predicates as boolean-valued maps $P : \Sigma \to \mathsf{Prop}$; these views are interchangeable. We write

$$\sigma \models P \quad \text{to mean} \quad \sigma \in P \quad \text{(i.e. } \sigma \text{ satisfies predicate } P\text{)}.$$

Logical connectives correspond to set operations:

$$P \wedge Q \equiv P \cap Q, \qquad P \vee Q \equiv P \cup Q, \qquad \neg P \equiv \Sigma \setminus P.$$

Definition 1.8. Given a command c and a set of initial states P, the *post-image* of P under c is the set of all states that can be reached by running c from some initial state in P:

$$\mathrm{post}(c)(P) \;\triangleq\; \{\sigma' \in \Sigma \mid \exists \sigma \in P.\, (\sigma, \sigma') \in [\![c]\!]\}.$$

In words, $\mathrm{post}(c)(P)$ collects exactly the *reachable final states* from P (upon termination). When the command is clear from context, we write $\mathrm{post}(P)$. Note that this function is not computable, as it can solve the Halting problem. We also may use post as a function:

$$\mathrm{post} : \mathsf{Cmd} \times \mathscr{P}(\Sigma) \to \mathscr{P}(\Sigma)$$

In a deterministic, partial-correctness setting, the strongest postcondition coincides with the post-image:

$$\mathrm{sp}_c(P) \;\triangleq\; \mathrm{post}(c)(P).$$

The terminology "strongest" means that $\mathrm{sp}_c(P)$ is the most precise postcondition that can follow from P through c. A predicate R is a *postcondition* for c from precondition P if every terminating run from a P-state ends in an R-state, i.e. if $\mathrm{post}(c)(P) \subseteq R$. Then $\mathrm{sp}_c(P)$ is the smallest such R (smallest by $\subseteq$).
Formally, for any $R \subseteq \Sigma$: $\mathrm{post}(c)(P) \subseteq R \iff \forall \sigma \in P.\, \forall \sigma'.\, (\sigma, \sigma') \in [\![c]\!] \Rightarrow \sigma' \in R.$
The operator $\mathrm{post}(c) : \mathcal{P}(\Sigma) \to \mathcal{P}(\Sigma)$ is monotone:

$$P \subseteq P' \;\Rightarrow\; \mathrm{post}(c)(P) \subseteq \mathrm{post}(c)(P').$$

It also respects unions:

$$\mathrm{post}(c)\left(\bigcup_{i \in I} P_i\right) = \bigcup_{i \in I} \mathrm{post}(c)(P_i).$$

These facts are immediate from the existential definition of $\mathrm{post}(c)$.

Compositional equations for $\mathrm{post}(c)$**.** From the definition of relational semantics and relation composition one obtains:

$$\mathrm{post}(\mathtt{skip})(P) = P, \qquad \mathrm{post}(c_1; c_2)(P) = \mathrm{post}(c_2)(\mathrm{post}(c_1)(P)).$$

For conditionals, define the guard-truth set

$$\mathrm{Guard}(b) \triangleq \{\sigma \in \Sigma \mid [\![b]\!](\sigma) = \mathsf{tt}\} \quad \mathrm{Guard}(\neg b) \triangleq \{\sigma \in \Sigma \mid [\![b]\!](\sigma) = \mathsf{ff}\}$$

Then

$$\mathrm{post}(\mathtt{if}\ b\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2)(P) = \mathrm{post}(c_1)(P \cap \mathrm{Guard}(b)) \;\cup\; \mathrm{post}(c_2)(P \cap \mathrm{Guard}(\neg b)).$$

This explicitely displays how branching partitions the initial states.

Post semantics for while as a least fixed point. Fix b and c. For $P \subseteq \Sigma$, define the monotone functional

$$H_P(X) \triangleq P \cup \operatorname{post}(c)\big(X \cap \operatorname{Guard}(b)\big).$$

Let

$$X^* \triangleq \operatorname{lfp}(H_P).$$

Intuitively, X^* is the set of states reachable at the loop head (just before testing the guard), starting from P. Then the set of terminating final states of the loop from P is

$$\operatorname{post}(\texttt{while } b \texttt{ do } c)(P) = X^* \cap \big(\operatorname{Guard}(\neg b)\big).$$

Moreover, by Kleene iteration,

$$X_0 \triangleq \emptyset, \qquad X_{n+1} \triangleq H_P(X_n) = P \cup \operatorname{post}(c)\big(X_n \cap \operatorname{Guard}(b)\big), \qquad X^* = \bigcup_{n \geq 0} X_n,$$

and therefore

$$\operatorname{post}(\texttt{while } b \texttt{ do } c)(P) = \left(\bigcup_{n \geq 0} X_n\right) \cap \big(\operatorname{Guard}(\neg b)\big).$$

Example: Kleene iteration for a countdown loop

Consider the loop

$$L \triangleq \texttt{while } b \texttt{ do } c, \qquad b \triangleq \neg(x \leq 0), \qquad c \triangleq (x := x - 1;\ y := y + 1).$$

Thus $\operatorname{Guard}(b) = \{\sigma \mid \sigma(x) > 0\}$ and $\operatorname{Guard}(\neg b) = \{\sigma \mid \sigma(x) \leq 0\}$.

Let the initial predicate be the *single* starting state

$$P \triangleq \{\sigma \in \Sigma \mid \sigma(x) = 2 \ \wedge\ \sigma(y) = 0\}.$$

Because c deterministically decrements x and increments y, we have:

$$\operatorname{post}(c)(P) = \{\sigma \in \Sigma \mid \sigma(x) = 1 \ \wedge\ \sigma(y) = 1\},$$

and applying the body once more,

$$\operatorname{post}(c)\big(\operatorname{post}(c)(P)\big) = \{\sigma \in \Sigma \mid \sigma(x) = 0 \ \wedge\ \sigma(y) = 2\}.$$

Recall the functional

$$H_P(X) \triangleq P \cup \operatorname{post}(c)\big(X \cap \operatorname{Guard}(b)\big), \qquad X_0 \triangleq \emptyset, \qquad X_{n+1} \triangleq H_P(X_n).$$

Now iterate:

$$X_0 = \emptyset.$$

$$X_1 = H_P(X_0) = P \cup \operatorname{post}(c)(\emptyset) = P = \{\sigma \mid \sigma(x) = 2 \wedge \sigma(y) = 0\}.$$

Since every $\sigma \in X_1$ satisfies $\sigma(x) = 2 > 0$, we have $X_1 \cap \operatorname{Guard}(b) = X_1$ and hence

$$\begin{aligned}
X_2 &= H_P(X_1) \\
&= P \cup \operatorname{post}(c)(X_1 \cap \operatorname{Guard}(b)) \\
&= P \cup \operatorname{post}(c)(P) \\
&= \{\sigma \mid \sigma(x) = 2 \wedge \sigma(y) = 0\} \ \cup\ \{\sigma \mid \sigma(x) = 1 \wedge \sigma(y) = 1\}.
\end{aligned}$$

Again, all states in X_2 have $x > 0$, so $X_2 \cap \mathrm{Guard}(b) = X_2$, and

$$\begin{aligned} X_3 &= H_P(X_2) \\ &= P \cup \mathrm{post}(c)(X_2 \cap \mathrm{Guard}(b)) \\ &= P \cup \mathrm{post}(c)(X_2) \\ &= P \cup \mathrm{post}(c)(P) \cup \mathrm{post}(c)(\mathrm{post}(c)(P)) \\ &= \{\sigma \mid \sigma(x) = 2 \land \sigma(y) = 0\} \ \cup \ \{\sigma \mid \sigma(x) = 1 \land \sigma(y) = 1\} \\ &\quad \cup \ \{\sigma \mid \sigma(x) = 0 \land \sigma(y) = 2\}. \end{aligned}$$

Now note that the last component has $x = 0$, so it *does not* satisfy the guard:

$$X_3 \cap \mathrm{Guard}(b) = \{\sigma \mid \sigma(x) = 2 \land \sigma(y) = 0\} \ \cup \ \{\sigma \mid \sigma(x) = 1 \land \sigma(y) = 1\}.$$

Therefore $\mathrm{post}(c)(X_3 \cap \mathrm{Guard}(b))$ produces only the $x = 1, y = 1$ and $x = 0, y = 2$ states again, so no new states appear:

$$X_4 = H_P(X_3) = X_3.$$

Hence the iteration has stabilized, and the least fixed point is

$$X^* = \mathrm{lfp}(H_P) = X_3.$$

By the post-semantics formula,

$$\mathrm{post}(L)(P) = X^* \cap \mathrm{Guard}(\neg b) = X^* \cap \{\sigma \mid \sigma(x) \leq 0\}.$$

Since X^* contains exactly one state with $x \leq 0$, we get

$$\mathrm{post}(\texttt{while}\ b\ \texttt{do}\ c)(P) = \{\sigma \in \Sigma \mid \sigma(x) = 0 \ \land \ \sigma(y) = 2\}.$$

Definition 1.9 (Weakest liberal precondition (for correctness reasoning))**.** Given a desired postcondition $Q \subseteq \Sigma$, the *weakest liberal precondition* of c with respect to Q is:

$$\mathrm{wlp}(c, Q) \ \triangleq \ \{\sigma \in \Sigma \mid \forall \sigma'.\, (\sigma, \sigma') \in [\![c]\!] \Rightarrow \sigma' \in Q\}.$$

This characterizes partial correctness: if c terminates from σ, then the result satisfies Q. The term "liberal" emphasizes that non-termination is allowed: if c does not terminate from σ, then $\sigma \in \mathrm{wlp}(c, Q)$ holds vacuously.

$$\langle c, \sigma \rangle \not\Downarrow \Longrightarrow \ \neg\exists\sigma'.\, (\sigma, \sigma') \in [\![c]\!] \ \Longrightarrow \ \big(\forall \sigma'.\, (\sigma, \sigma') \in [\![c]\!] \Rightarrow \sigma' \in Q\big) \text{ (vacuously true).}$$

Adjoint-style connection between post **and** wlp**.** The post-image and weakest liberal precondition are linked by the following equivalence:

$$\mathrm{post}(c)(P) \subseteq Q \quad \Longleftrightarrow \quad P \subseteq \mathrm{wlp}(c, Q).$$

Proof. ($\Rightarrow$) Take $\sigma \in P$. For any σ' with $(\sigma, \sigma') \in [\![c]\!]$, we have $\sigma' \in \mathrm{post}(c)(P) \subseteq Q$, hence $\sigma \in \mathrm{wlp}(c, Q)$. ($\Leftarrow$) Take $\sigma' \in \mathrm{post}(c)(P)$, so $\exists \sigma \in P$ with $(\sigma, \sigma') \in [\![c]\!]$. Since $\sigma \in \mathrm{wlp}(c, Q)$, we obtain $\sigma' \in Q$.

Both Hoare Logic and Incorrectness Logic can be phrased using the same post-image operator.

A (partial) Hoare triple $\{P\}\ c\ \{Q\}$ is semantically valid exactly when every terminating execution from P ends in Q:

$$\models \{P\}\, c\, \{Q\} \quad \Longleftrightarrow \quad \mathrm{post}(c)(P) \subseteq Q \quad \Longleftrightarrow \quad P \subseteq \mathrm{wlp}(c, Q).$$

This is an *over-approximation* statement about reachable states.

An incorrectness-style triple $[P]\ c\ [Q]$ (in its simplest reachability form) is semantically valid when every state in Q is reachable as a terminating final state from some initial state in P:

$$\models [P]\, c\, [Q] \quad \Longleftrightarrow \quad Q \subseteq \mathrm{post}(c)(P).$$

This is an *under-approximation* statement: proving $Q \subseteq \mathrm{post}(c)(P)$ certifies genuine reachability (hence genuine bugs) with no false positives.

It is useful to keep the contrast visible:

$$\text{Correctness: } \mathrm{post}(c)(P) \subseteq Q \qquad \text{vs.} \qquad \text{Incorrectness: } Q \subseteq \mathrm{post}(c)(P).$$

This single semantic object $\mathrm{post}(c)(P)$ (reachable final states from P) will therefore be the conceptual bridge between correctness reasoning and incorrectness reasoning, and it will later generalize to the quantum setting where post becomes a transformer on quantum states.