Property-based and metamorphic quantum tests

Find defects using invariants and related inputs when a complete answer is expensive.

Undergraduate 22 min lesson Dr. Charith Jayasekara

By the end, you'll be able to

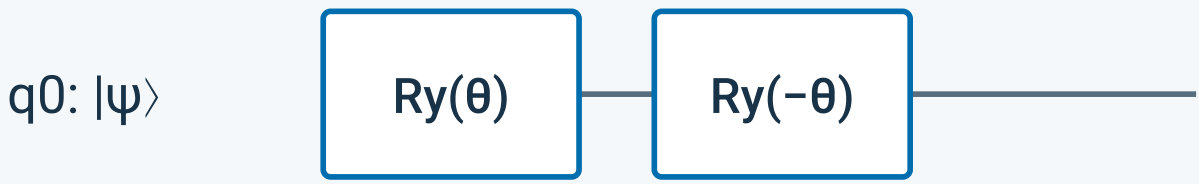
- Express a quantum identity as a test property.
- Construct metamorphic input transformations.
- Explain what a passing property leaves untested.

Begin with a clear question

Sometimes the exact output of a large circuit is difficult to compute. You can still check relations that must hold. A property-based test explores many valid inputs; a metamorphic test changes an input or program in a way that predicts a related result. These techniques extend familiar software testing to coherent operations.

Build the reasoning

A rotation followed by its inverse should preserve every state. Inserting HH should also preserve an ideal circuit because H squared is identity. For an oracle, test the promised input structure independently from the algorithm using it. The BV notebook checks recovery across secret strings, while the Simon example checks every promised pair and solves the resulting binary equations. Each relation gives a specific kind of evidence.



Follow time from left to right. Qiskit statevector labels put q0 on the right; diagrams put q0 on the top wire.

Worked example

The code constructs Ry(theta) followed by Ry(−theta) for several angles. It compares the whole one-qubit unitary with identity and then checks four diagnostic states, including a complex superposition. These deterministic boundary and

phase cases complement random parameter generation.

$$R_y(-\theta)R_y(\theta) = I, \qquad H^2 = I, \qquad U_f^2 = I \text{ for } U_f|x, y\rangle = |x, y \oplus f(x)\rangle.$$

```
from qiskit import QuantumCircuit
from qiskit.quantum_info import Statevector, Operator
import numpy as np

identity = Operator(np.eye(2))
for theta in [0, 0.1, -0.7, np.pi, 2*np.pi]:
    circuit = QuantumCircuit(1)
    circuit.ry(theta, 0)
    circuit.ry(-theta, 0)
    assert Operator.from_circuit(circuit).equiv(identity)
    for label in ["0", "1", "+", "r"]:
        state = Statevector.from_label(label)
        assert state.evolve(circuit).equiv(state)
```

Practical lab

A property can be too weak. Two identical incorrect inverse builders may cancel and pass while both use the wrong angle convention. Keep a few independently calculated reference cases alongside relational tests. Cover zero, negative and wrap-around angles, and include states with imaginary amplitudes. Record a failing input and simplify it so the defect can be explained in one small example.

Run the corresponding notebook from the software practice collection:

- Download Two oracle algorithms: recover bits and recover a hidden XOR period
- Download Quantum software tests that catch plausible wrong programs

Keep a short execution record: predicted result, observed result, environment, inputs and the check that would reject a plausible defect. Explain a failure before changing the expected value.

Chapter summary

- Choose relations that follow from the intended mathematics.
- Combine varied parameters with phase-sensitive states.
- Use independent reference cases to catch shared defects.

Exercises

1. An implementation replaces every Ry(theta) by Ry(2theta), including its inverse. Does the inverse-pair test catch it?
2. Change one part of the worked program to create a plausible fault. State which assertion should fail and why.

Worked answers

1. No: Ry(−2theta)Ry(2theta) is still identity. Compare a single rotation at a known angle, such as Ry(π)|0⟩, against its independent expected state.
2. A useful answer identifies a specific changed gate, parameter, mapping or model and links it to an independent expected behaviour. Keep the intended specification fixed while changing the implementation; save the failing case as a regression test.

Check your understanding

PAUSE & THINK

Why keep an independent reference test beside an inverse-pair property?

- [ ] Two matching implementation mistakes can cancel
- [ ] Relational tests prove every possible specification
- [ ] A random seed is a mathematical proof

Answer: Two matching implementation mistakes can cancel. Independent expected values can expose a shared mistake in both builders.

Sources

IBM's bit-ordering guide, Statevector API and Operator API describe the SDK conventions and equivalence tools. The notebook bundle records its pinned environment and local execution evidence.