QHack

Quantum Coding Challenges

Ctrl-Z

100 points

Backstory

Zenda and Reece work at Trine's Designs, a startup run by the eccentric inventor Doc Trine. Trine promises to tell Zenda and Reece about a revolutionary new type of quantum resource she has invented called "*timbits*". Before explaining timbits, she insists on demonstrating Bennett's Laws of Infodynamics, governing the behaviour of quantum information. "*Only then*," she says, "*will the power of timbits be revealed in their full glory*."

Reversible computation

▾ **Laws of Infodynamics Part I: The First Law**

This box contains some interesting but nonessential details. A qubit can be used to imitate a classical bit (which we'll call *cbits*), since instead of sending a cbit j, we can send a basis state $|j\rangle$. We can write this as an inequality, the First Law of Infodynamics:

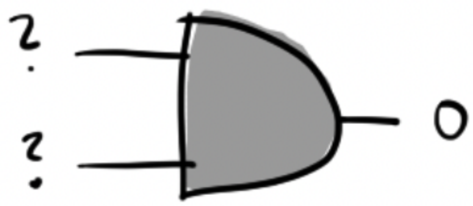
$$1 \text{ qubit} \geq 1 \text{ cbit}. \qquad (1)$$

But although we can encode classical data into qubits, it's not obvious we can always compute in the same way.

Some classical logical operations are *irreversible*. For instance,

$$\text{AND}(0,0) = \text{AND}(0,1) = \text{AND}(1,0) = 0,$$

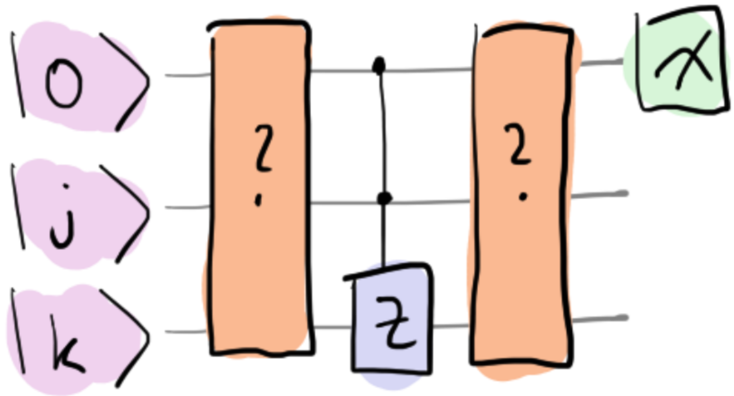
so given that $\text{AND}(j,k) = 0$, we can't tell the values of j and k.



Put differently, there is no way to press `ctrl-Z` and learn what went in! In contrast, quantum circuits are built out of unitary gates, which are always reversible. We can always press `ctrl-Z`! How can we encode something irreversible, like an AND gate, into a quantum circuit? Aptly, the answer is a controlled Z gate! It encodes the classical operation into a *phase*:

$$CZ|j,k\rangle \mapsto (-1)^{\text{AND}(j,k)}|j,k\rangle.$$

A phase by itself is unobservable, so we need to interfere this state with some others to detect it. A simple way to do this is to use a *controlled* controlled Z gate, with some extra operations on either side:



Your job: figure out which operations to apply so that measurement on the first qubit is guaranteed to be in state $|\mathrm{AND}(j,k)\rangle$.

Challenge code

In the code below, you are given a function called `AND(j, k)`. **You must complete this circuit** and provide gates which implement a classical AND gate. More precisely, if the second and third qubits are in states $|j\rangle$ and $|k\rangle$, the circuit should place the first qubit in state $|\mathrm{AND}(j,k)\rangle$.

Inputs

As input to this problem, you are given two bits `j (int)` and `k (int)`, encoded onto the second and third qubits for you.

Output

Your circuit must place the first qubit in basis state `AND(j, k)`. This will be checked using `qml.probs(wires = 0)`, which gives `[1, 0]` for $|0\rangle$ and `[0, 1]` for $|1\rangle$.

If your solution matches the correct one within the given tolerance specified in `check` (in this case it's a `1e-4` relative error tolerance), the output will be `"Correct!"` Otherwise, you will receive a `"Wrong answer"` prompt.

Code

Help

```
import json
import pennylane as qml
import pennylane.numpy as np
```

```python
dev = qml.device("default.qubit", wires=3)

@qml.qnode(dev)
def AND(j, k):
    """Implements the AND gate using quantum gates and computes

    Args:
        j (int): A classical bit, either 0 or 1.
        k (int): A classical bit, either 0 or 1.

    Returns:
        float: The probabilities of measurement on wire 0.
    """

    if j == 1:
        qml.PauliX(wires=1)
    if k == 1:
        qml.PauliX(wires=2)

```

```python
    # Put your code here #

```

```python
    qml.ctrl(qml.PauliZ, control =[0, 1])(wires = [2])


```

```python
    # Your code here #

    return qml.probs(wires=0)

```

```python
# These functions are responsible for testing the solution.
def run(test_case_input: str) -> str:
    j, k = json.loads(test_case_input)
    output = AND(j, k).tolist()

    return str(output)

def check(solution_output: str, expected_output: str) -> None:
    solution_output = json.loads(solution_output)
    expected_output = json.loads(expected_output)
    assert np.allclose(solution_output, expected_output, rtol=1e

```

```python
test_cases = [['[0, 0]', '[1, 0]'], ['[1, 1]', '[0, 1]']]
```

```python
for i, (input_, expected_output) in enumerate(test_cases):
    print(f"Running test case {i} with input '{input_}'...")

    try:
        output = run(input_)

    except Exception as exc:
        print(f"Runtime Error. {exc}")

    else:
        if message := check(output, expected_output):
            print(f"Wrong Answer. Have: '{output}'. Want: '{expe

        else:
            print("Correct!")
```

Copy all

Submit

Open Notebook

Reset