Microsoft

Programming assignment 7
Phase estimation

Mariia Mykhailova
Senior Software Engineer
Microsoft Quantum Systems

Task 1.1: Eigenstates of the Hadamard gate

$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ is self-adjoint, so eigenvalues $\lambda = \pm 1$

Find eigenvectors:

- $\lambda = 1$: $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$; $\begin{cases} \frac{1}{\sqrt{2}}(x_0 + x_1) = x_0 \\ \frac{1}{\sqrt{2}}(x_0 - x_1) = x_1 \end{cases}$; $\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 1 + \sqrt{2} \\ 1 \end{pmatrix}$
- $\lambda = -1$: similarly $\begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 1 - \sqrt{2} \\ 1 \end{pmatrix}$
- Both eigenvectors should be normalized to produce eigenstates!

Task 1.1: Eigenstates of the Hadamard gate

Prepare state $\beta_\pm\left((1 \pm \sqrt{2})|0\rangle + |1\rangle\right)$ ($\beta_\pm$ is normalization coef.)

- Represent the states as $\cos\alpha_\pm |0\rangle + \sin\alpha_\pm |1\rangle$:

$$\tan\alpha_\pm = \frac{1}{1 \pm \sqrt{2}}$$

- Then use Ry gate to prepare the state, per BasicGates task 1.4

```
let sign = state == 0 ? -1.0 | 1.0;
let alpha = ArcTan(1.0 / (1.0 + Sqrt(2.0) * sign));
Ry(2.0 * alpha, q);
```

Task 1.2: Power oracle for T gates

Return a single-qubit unitary equal to T^P.

You can use at most one T gate!

$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/4} \end{pmatrix}$$

Can we express higher powers of T using other gates?

$$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi/2} \end{pmatrix} = T^2$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = S^2 = T^4$$

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = Z^2 = T^8$$

Express the unitary T^P as a product of powers of T that correspond to binary digits of P: $T^1 \cdot T^2 \cdot \ T^4 \cdot \ T^8 \cdot \ldots$

Task 1.2: Power oracle for T gates

$$T^1 \cdot T^2 \cdot \; T^4 \cdot \; T^8 \cdot \cdots = T \cdot S \cdot Z \cdot I \cdot \cdots$$

(each power is included only if corresponding digit is 1)

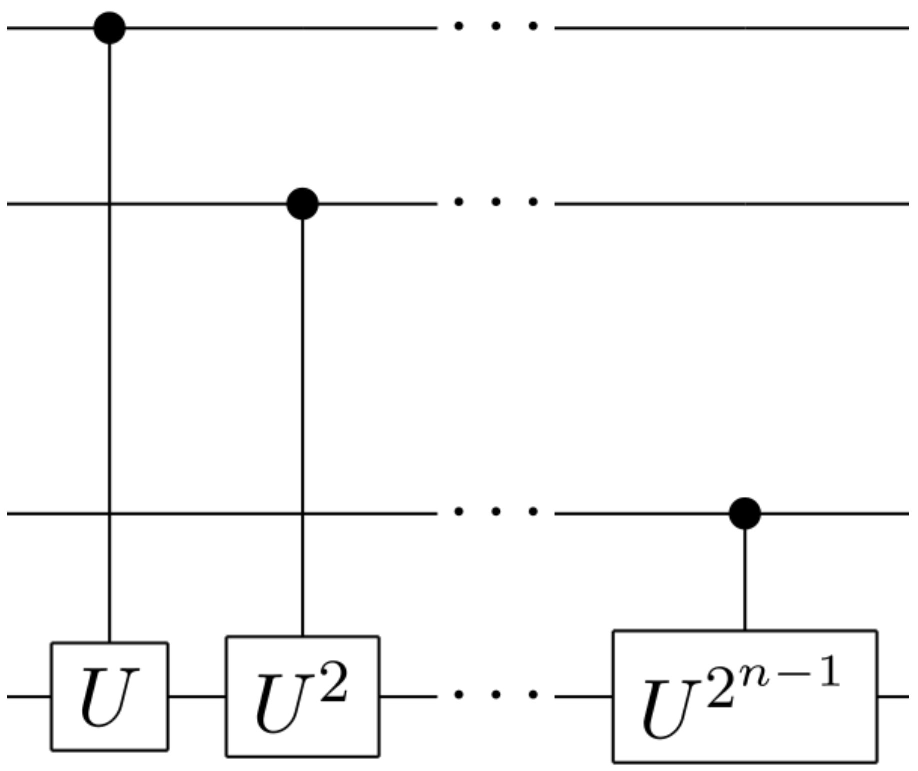
```
operation TPowerImpl (P : Int, q : Qubit) : Unit is Adj+Ctl {
    if (P % 2 == 1) { T(q); }
    if ((P >>> 1) % 2 == 1) { S(q); }
    if ((P >>> 2) % 2 == 1) { Z(q); }
}

function Task12 (P : Int) : (Qubit => Unit is Adj+Ctl) {
    return TPowerImpl (P, _);
}
```

Task 1.3: Quantum version of unitary power

Apply k-th power of unitary U: $|k\rangle|\psi\rangle \rightarrow |k\rangle U^k|\psi\rangle$

The part of the phase estimation circuit presented in the lecture:
top wire is the least significant bit,
bottom wire – the most significant bit

This allows to apply unitary power in superposition, like oracles



Task 1.3: Quantum version of unitary power

Apply k-th power of unitary U: $|k\rangle|\psi\rangle \rightarrow |k\rangle U^k|\psi\rangle$

If you don't know anything about the unitary $\boldsymbol{U}$, the only way you can apply its power is by repeated application of $\boldsymbol{U}$.

```
for (i in 0 .. Length(powerRegister)-1) {
    for (j in 1 .. (1 <<< i)) {
        Controlled U(powerRegister[i..i], target);
    }
}
```

Task 1.4: Reverse-engineer QPE

Return any unitary and its eigenstate with eigenvalue $e^{2i\pi\varphi}$.

$$R_1(\theta) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$$

Can use $R_1(2\pi\varphi)$ and X (to prepare eigenstate $|1\rangle$), but…

When does QPE procedure fail?

When the eigenphase can not be expressed in binary exactly!

What can you do to make sure it does not fail on the unitaries you return?

- The eigenphase returned by the QPE needs to be accurate within 0.01
- Find a phase close enough to φ that *can* be expressed in binary φ_{bin}
- And return a unitary $R_1(2\pi\varphi_{bin})$ and X.

Task 1.4: Reverse-engineer QPE

Return any unitary and its eigenstate with eigenvalue $e^{2i\pi\varphi}$.

```
// Find the closest phase that can be represented exactly
// in binary with 8 digits of precision
let binaryPhase = IntAsDouble(
    Round(phase * IntAsDouble(1 <<< 8))) /
    IntAsDouble(1 <<< 8);
// Use R1 rotation with the given angle
return (R1(2.0 * PI() * binaryPhase, _), X);
```