Microsoft

Programming assignment 4
Quantum oracles

Mariia Mykhailova
Senior Software Engineer
Microsoft Quantum Systems

Task 1.1: Marking oracle for $|111\ldots000\ldots\rangle$

Transform state $|x, y\rangle$ into state $|x, y \oplus f(x)\rangle$, where

$$f(x) = 1 \text{ if } x = 111\ldots000\ldots, \text{and } 0 \text{ otherwise}$$

- Transform the input so that marked state becomes $|1\ldots1\rangle$ and apply `Controlled X`

```
within {
    ApplyToEachA(X, queryRegister[Length(queryRegister) / 2 ...]);
} apply {
    Controlled X(queryRegister, target);
}
```

- Or GroversAlgorithm kata, task 1.3: arbitrary bit pattern oracle

```
let pattern = ConstantArray(N2, true) + ConstantArray(N2, false);
```

Task 1.2: Prefix matching oracle

Transform state $|x, y\rangle$ into state $|x, y \oplus f(x)\rangle$, where

$$f(x) = 1 \text{ if prefix of bit string } x \text{ matches the given pattern}$$

- GroversAlgorithm kata, task 1.3: arbitrary bit pattern oracle
- Applied to the prefix of the qubit array that should match the given pattern

```
(ControlledOnBitString(pattern, X))
(queryRegister[... Length(pattern) - 1], target);
```

Task 1.3: Regexp matching oracle

Transform state $|x, y\rangle$ into state $|x, y \oplus f(x)\rangle$, where

$$f(x) = 1 \text{ if the given bits of } x \text{ match the given pattern}$$

- Same logic, but more work to extract the qubits and the pattern

```
mutable controlMask = new Bool[0];
mutable controlQubits = new Qubit[0];
for (i in 0 .. Length(pattern) - 1) {
    if (pattern[i] != -1) {
        set controlMask = controlMask + [pattern[i] == 1];
        set controlQubits = controlQubits + [queryRegister[i]];
    }
}
(ControlledOnBitString(controlMask, X))(controlQubits, target);
```

Task 1.4: Substring searching oracle

Transform state $|x, y\rangle$ into state $|x, y \oplus f(x)\rangle$, where

$$f(x) = 1 \textbf{ if the bit string } x \textbf{ contains the given substring}$$

- Check N – K + 1 separate conditions: substring of x that starts at index j equals the given string (`ControlledOnBitString`)
- Allocate scratch qubits to store individual conditions
- Overall function is OR of those conditions
- Remember to uncompute the scratch qubits before releasing

Task 1.4: Substring searching oracle (code)

Transform state $|x, y\rangle$ into state $|x, y \oplus f(x)\rangle$, where

$$f(x) = 1 \text{ if the bit string } x \text{ contains the given substring}$$

```
let N = Length(queryRegister);
let K = Length(substring);
using (anc = Qubit[N - K + 1]) {
    within {
        for (i in 0 .. N - K) {
            (ControlledOnBitString(substring, X))
            (queryRegister[i .. i + K - 1], anc[i]);
        }
    } apply {
        (ControlledOnInt(0, X))(anc, target);
        X(target);
    }
}
```

Task 1.5: Majority function on 5 qubits

Transform state $|x, y\rangle$ into state $|x, y \oplus f(x)\rangle$, where

$$f(x) = 1 \text{ if the 5bit string } x \text{ contains more 1s than 0s}$$

- Can always iterate over all input bit strings, calculate $f(x)$ classically, and use `ControlledOnBitString` if it is 1
  - For this oracle, it will take $C_5^3 + C_5^4 + C_5^5 = 16$ controlled NOTs
  - We can do that for any function, but it negates any potential quantum speedup
- Let's represent $\mathbf{MAJ}(x)$ as XOR of all 3-bit and 4-bit ANDs:

$$(x_0 \wedge x_1 \wedge x_2) \oplus (x_0 \wedge x_1 \wedge x_3) \oplus \ldots \oplus (x_2 \wedge x_3 \wedge x_4) \oplus$$
$$(x_0 \wedge x_1 \wedge x_2 \wedge x_3) \oplus \ldots \oplus (x_1 \wedge x_2 \wedge x_3 \wedge x_4)$$

  - 10 3-bit clauses and 5 4-bit clauses, so 15 controlled NOTs (with fewer controls)
  - For 3, 4 or 5 bits set to 1 an odd number of clauses will evaluate to 1

Task 1.5: Majority function on 5 qubits (code)

Transform state $|x, y\rangle$ into state $|x, y \oplus f(x)\rangle$, where

$$f(x) = 1 \text{ if the 5bit string } x \text{ contains more 1s than 0s}$$

```
for (i1 in 0..4) {
    for (i2 in i1+1..4) {
        for (i3 in i2+1..4) {
            Controlled X([x[i1], x[i2], x[i3]], y);
            for (i4 in i3+1..4) {
                Controlled X([x[i1], x[i2], x[i3], x[i4]], y);
            }
        }
    }
}
```

Task 2.1: Arbitrary bit pattern phase oracle

Transform state $|x\rangle$ into state $(-1)^{f(x)}\,|x\rangle$, where

$$f(x) = 1 \text{ if the bit string } x \text{ equals the given bit pattern}$$

- For a marking oracle, we use `ControlledOnBitString(_, X)`
- For a phase oracle, we use `ControlledOnBitString(_, Z)`
  - The control bit string is all bit pattern except the last digit
  - The target is last qubit; if the last digit of the pattern is 0, it needs to be flipped first (for Z gate to take effect)

```
within {
    if (not Tail(pattern)) { X(Tail(x)); }
} apply {
    (ControlledOnBitString(Most(pattern), Z))(Most(x), Tail(x));
}
```

Task 2.2: Conversion between phase oracles

Transform phase oracle of form $|x\rangle \rightarrow (-1)^{f(x)}\,|x\rangle$
into a phase oracle of form $|x\rangle|b\rangle \rightarrow (-1)^{b\cdot f(x)}\,|x\rangle|b\rangle$.

- Consider two cases for b:
  - $b = 0$: nothing happens
  - $b = 1$: we do the transformation $|x\rangle \rightarrow (-1)^{f(x)}\,|x\rangle$ (the original oracle)
  - That's exactly the controlled version of the oracle!

```
operation Impl(oracle : (Qubit[] => Unit is Adj+Ctl), x : Qubit[], b : Qubit) : Unit is Adj {
    (Controlled phaseOracle)([b], x);
}
function Task22(oracle : (Qubit[] => Unit is Adj+Ctl)) : ((Qubit[], Qubit) => Unit is Adj) {
    return Impl(oracle, _, _);
}
```

Task 2.3: Phase oracle to marking oracle

Transform oracle $|x\rangle|b\rangle \to (-1)^{b \cdot f(x)} |x\rangle|b\rangle$ into oracle $|x\rangle|b\rangle \to |x\rangle|b \oplus f(x)\rangle$.

- Consider two cases for $f(x)$:
  - $f(x) = 0$: both oracles do nothing
  - $f(x) = 1$: phase oracle becomes $|b\rangle \to (-1)^b |b\rangle$ (looks like Z gate), marking oracle - $|b\rangle \to |\neg b\rangle$ (looks like X gate)
  - H gate switches between computational basis and Hadamard basis

```
operation Impl(oracle : (Qubit[] => Unit is Adj+Ctl), x : Qubit[], b : Qubit) : Unit is Adj {
    H(b);
    oracle(x, b);
    H(b);
}
function Task23(oracle : (Qubit[] => Unit is Adj+Ctl)) : ((Qubit[], Qubit) => Unit is Adj) {
    return Impl(oracle, _, _);
}
```

Task 2.4: Oracle with extra state (un)marked

Transform a marking oracle $|x\rangle|b\rangle \rightarrow |x\rangle|b \oplus f(x)\rangle$ into an oracle that marks/unmarks one extra state x_0.

$$f'(x) = \begin{cases} \neg f(x), x = x_0 \\ f(x), x \neq x_0 \end{cases}$$

- $\neg f(x) = f(x) \oplus 1$: if $x = x_0$, we need to additionally flip the result
- Apply the oracle, followed by **`ControlledOnBitString`**

```
operation Impl (markingOracle : ((Qubit[], Qubit) => Unit is Adj), pattern : Bool[],
  x : Qubit[], b : Qubit) : Unit is Adj {
    markingOracle(x, b);
    (ControlledOnBitString(pattern, X))(x, b);
}
function Task24 (markingOracle : ((Qubit[], Qubit) => Unit is Adj), pattern : Bool[]) :
  ((Qubit[], Qubit) => Unit is Adj) {
    return Impl(markingOracle, pattern, _, _);
}
```