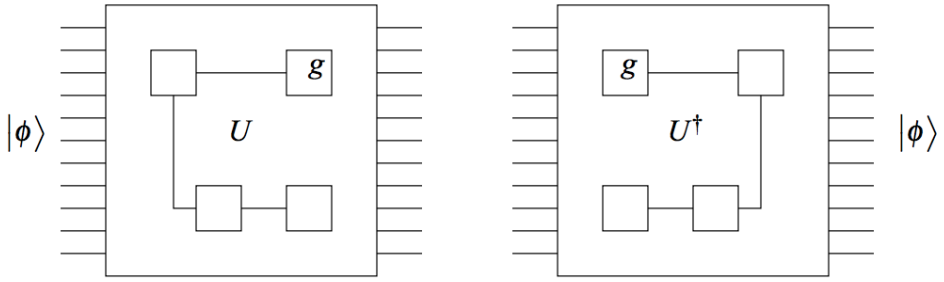
Chapter 4

Fourier Sampling & Simon's Algorithm

4.1 Reversible Computation

A quantum circuit acting on n qubits is described by an $2^n \times 2^n$ unitary operator U. Since U is unitary, $UU^\dagger = U^\dagger U = I$. This implies that each quantum circuit has an inverse circuit which is the mirror image of the original circuit and which carries out the inverse operator $U^\dagger$.



The circuits for U and $U^\dagger$ are the same size and have mirror image gates. Examples:

$$H = H^\dagger$$

$$\text{CNOT} = \text{CNOT}^\dagger$$

$$R_\theta = R_{-\theta}^\dagger$$

4.2 Simulating Classical Circuits

Let us first consider whether given any classical circuit there is an equivalent quantum circuit. More concretely, suppose there is a classical circuit that computes a function $f(x) \in \{0,1\}^m$ on input $x \in \{0,1\}^n$, is there a quantum circuit that does the same? Obviously such a quantum circuit must map computational basis states to computational basis states (i.e. it must map each state of the form $|x\rangle$ to the state $|f(x)\rangle$). A unitary transformation taking basis states to basis states must be a permutation. (Indeed, if $U|x\rangle = |u\rangle$ and $U|y\rangle = |u\rangle$, then $|x\rangle = U^{-1}|u\rangle = |y\rangle$.) Therefore we need the input, or domain, to be the exact same number of bits as the range: $m = n$. What is more, the function $f(x)$ must be a permutation on the n-bit strings. Since this must hold after every application of a quantum gate, it follows that if a quantum circuit computes a classical function, then it must be *reversible*: it must have an inverse.

How can a classical circuit C which takes an n bit input x and computes $f(x)$ be made into a reversible quantum circuit that computes the same function? The circuit must never lose any information, so how could it compute a function mapping n bits to $m < n$ bits (e.g. a boolean function, where $m = 1$)?

The solution to this problem is to have the circuit take the n input qubits in the state $|x\rangle$ and send them to $|x\rangle$, while in the process taking some m qubits in the $|0\rangle$ state to $|f(x)\rangle$. Then the inverse map is simple: the n-bit string $|x\rangle$ goes back to $\mathbf{x}$, and $|f(x)\rangle$ goes to an m bit string of 0's: $|0\rangle$.

However, this is not always perfectly easy, some times to make the circuit work it needs scratch qubits in the input. A scratch qubit is a qubit that starts out in the $|0\rangle$ state, and ends in the $|0\rangle$ state. Its purpose is to be used in computations inside of the circuit. Of course, since the quantum circuit does not alter these qubits, the inverse circuit also leaves them alone. While these bits are a necessary ingredient to a reversible quantum circuit, they are not the main character and are often let out of circuit diagrams. Take a look at Figure 4.2 for the full picture.

How is this done? It is a fact that any classical AND and OR gates can be simulated with a C-SWAP gate and some scratch $|0\rangle$ qubits (Figure 4.2). For example, if we want to make $a \wedge b$, then we input a as the control bit and b as one of the swap bits, with $c = 0$ as the other swap bit. In the end, we measure the third register where c went in, and this will be $a \wedge b$ Now look what happens: if $a = b = 1$, then b and c swap, so that the third register reads $b = 1$: true! If a is one but b is 0, then b and c swap, but the third register is $b = 0$: false. And clearly, if $a = 0$ then so to will read the third register.

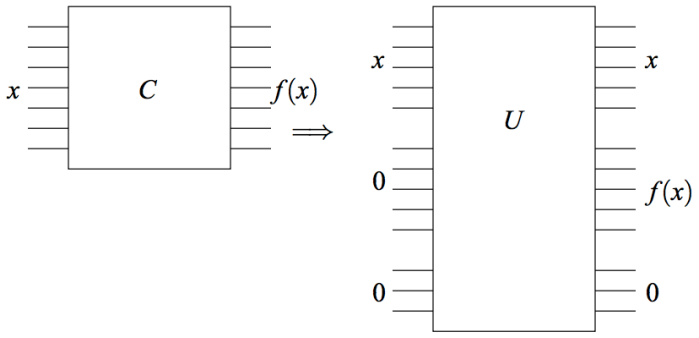
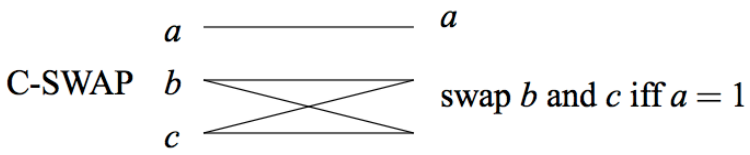


Figure 4.1: Note that the input and output have the same number of qubits in the reversible quantum circuit.



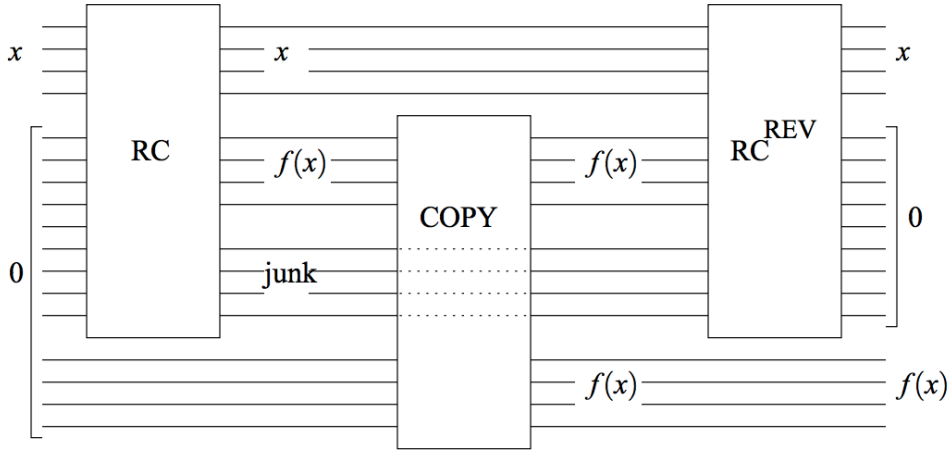
If we construct the corresponding reversible circuit (lets call it RC), we have a small problem. The CSWAP gates end up converting input scratch bits to garbage. Why is this a problem? We have our output $|f\rangle(x)$, don't we. This seems like it should be good enough. But in fact it is not. All the junk that gets made in the in-between steps can be entangled with the output qubits. Thus if it gets measured, it will screw alter our function. Furthermore, there is a principle which states that any unmeasured, thrown away qubits are just as good as measured. This is called the principle of deferred measurement, and it means that junk qubits are no good.

So how do we restore the scratch bits to 0 on output? We use the fact that RC is a reversible circuit. We use the CSWAP gates, for example, to produce the output $f(x)$. We can then copy the output onto some scratch qubits, which we will keep as our output. We then use the reverse of our circuit RC on the input, $f(x)$, and the junk to turn it all back to 0's and x. But because we copied $f(x)$ in the middle, we keep it at the end.

The sequence of steps for the overall circuit is

$$(x, 0^k, 0^m, 0^k, 1) \xrightarrow{C'} (x, y, \text{garbage}_x, 0^k, 1) \xrightarrow{\text{copy } y} (x, y, \text{garbage}_x, y, 1) \xrightarrow{(C')^{-1}} (x, 0^k, 0^m, y, 1) \ .$$

Overall, this gives us a clean reversible circuit $\hat{C}$ corresponding to C.



4.3 Fourier Sampling

Consider a quantum circuit acting on n qubits, which applies a Hadamard gate to each qubit. i.e. the circuit applies the unitary transformation $H^{\otimes n}$, or H tensored with itself n times.

Another way to define this unitary transformation H_{2^n} is as the $2^n \times 2^n$ matrix in which the (x, y) entry is $2^{-n/2}(-1)^{x \cdot y}$.

Applying the Hadamard transform (or the Fourier transform over Z_2^n) to the state of all zeros gives an equal superposition over all 2^n states

$$\mathcal{H}_{2^n} |0 \cdots 0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle .$$

In general, applying the Hadamard transform to the computational basis state $|u\rangle$ yields:

$$\mathcal{H}_{2^n} |u\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{u \cdot x} |x\rangle$$

We define the Fourier sampling problem as follows: Input an n qubit state $|\phi\rangle = \sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$. Compute $H^{\otimes n} |\phi\rangle$ and measure the resulting state $\sum_y \hat{\alpha}_y |y\rangle$ to output y with probability $|\hat{\alpha}_y|^2$.

Fourier sampling is probably the most fundamental primitive we use in quantum algorithms (where in place of the Hadamard transform, we will use a more general form of type of Fourier transform). Fourier sampling is easy on a quantum computer, but appears to be difficult to carry out on a classical

computer. In what follows, we will explore some of the power of Fourier sampling.

4.4 Phase State

We will now see how to set up an interesting state for fourier sampling. Given a classical circuit for computing a boolean function $f : \{0,1\}^n \to \{0,1\}$, this procedure due to Deutsch and Jozsa, shows how to transform it into a quantum circuit that produces the quantum state $|\phi\rangle = 1/2^{n/2} \sum_x (-1)^{f(x)} |x\rangle$.

The quantum algorithm to carry out this task uses two quantum registers, the first consisting of n qubits, and the second consisting of a single qubit.

- Start with the registers in the state $|0^n\rangle |0\rangle$
- Compute the Fourier transform on the first register to get $\sum_{x\in\{0,1\}^n} |x\rangle \otimes |0\rangle$.
- Compute f to get $\sum_x |x\rangle |f(x)\rangle$.
- Apply a conditional phase based on $f(x)$ to get $\sum_x (-1)^{f(x)} |x\rangle |f(x)\rangle$.
- Uncompute f to get $\sum_x (-1)^{f(x)} |x\rangle \otimes |0\rangle$.

4.5 Extracting n bits with 2 evaluations of Boolean Function

Suppose we are given a black box (or an obfuscated classical circuit) that computes the function function $f_s : \{0,1\}^n \to \{1,-1\}$, where $f(x) = s \cdot x$. $s \cdot x$ denotes the dot product $s_1 x_1 + \cdots + s_n x_n \bmod 2$. The challenge is to use this black box to efficiently determine s.

It is easy to see how to perform this task with n queries to the black box: simply input in turn the n inputs x of Hamming weight 1. The outputs of the black box are the bits of s. Since each query reveals only one bit of information, it is clear that n queries are necessary.

Remarkably there is a quantum algorithm (the base case of the Bernstein-Vazirani algorithm) that requires only two (quantum) queries to the black box:

- Use the black box to set up the phase state $|\phi\rangle = 1/2^{n/2} \sum_x (-1)^{f(x)} |x\rangle$.

- Apply the Fourier transform $H^{\otimes n}$ and measure. The outcome of the measurement is s.

To see that the outcome of the measurement is s, recall that $H^{\otimes n}|s\rangle = 1/2^{n/2}\sum_x(-1)^{s\cdot x}|x\rangle = |\phi\rangle$. Since $H^{\otimes n}$ is its own inverse, it follows that $H^{\otimes n}|\phi\rangle = |s\rangle$.

More generally, the transformation $H^{\otimes n}$ maps the standard basis $|s\rangle$ to the fourier basis $|\phi_s\rangle = 1/2^{n/2}\sum_x(-1)^{s\cdot x}|x\rangle$ and vice-versa.

We have shown that a quantum algorithm can be more efficient than any probabilistic algorithm in terms of the number of queries. One way to use this difference in the number of queries in order to demonstrate a gap between quantum and probabilistic algorithms is to make the queries very expensive. Then the quantum algorithm would be $n/2$ times faster than any probabilistic algorithm for the given task. But this does not help us in our goal, which is to show that quantum computers violate the extended Church-Turing thesis. The idea behind proving a superpolynomial gap (which we will outline below) is to make each query itself be the answer to a Fourier sampling problem. Now each query itself is much easier for the quantum algorithm than for any probabilistic algorithm. Carrying this out recursively for $\log n$ levels leads to the superpolynomial speedup for quantum algorithms.

4.6 Recursive Fourier Sampling

Our goal is to give a superpolynomial separation between quantum computation and classical probabilistic computation. The idea is to define a recursive version of the fourier sampling problem, where each query to the function (on an input of length n) is itself the answer to a recursive fourier sampling problem (on an input of length n/2). Intuitively a classical algorithm would need to solve n subproblems to solve a problem on an input of length n (since it must make n queries). Thus its running time satisfies the recurrence $T(n) \geq nT(n/2) + O(n)$ which has solution $T(n) = \Omega(n^{\log n})$. The quantum algorithm needs to make only two queries and thus its running time satisfies the recurrence $T(n) = 2T(n/2) + O(n)$, which solves to $T(n) = O(n \log n)$.

Here is how it works for two levels: we are given a black box computing a function $f : \{0,1\}^{3n/2} \to \{0,1\}$, with the promise that for every n bit string x, the function $f_x : \{0,1\}^{n/2} \to \{0,1\}$ defined by $f_x(y) = f(xy)$ (xy is denotes the concatenation of x and y) satisfies $f_x(y) = s_x \cdot y$ for some $s_x \in \{0,1\}^{n/2}$. We are also given a black box $g : \{0,1\}^{n/2} \to \{0,1\}$ which satisfies the condition that if we construct a boolean function h on n bits as $h(x) = g(s_x)$, then $h(x) = s \cdot x$ for some n-bit string s. The challenge is to figure out s.

The proof that no classical probabilistic algorithm can reconstruct s is somewhat technical, and establishes that for a random g satisfying the promise, any algorithm (deterministic or probabilistic) that makes $n^{o(\log n)}$ queries to g must give the wrong answer on at least $1/2 - o(1)$ fraction of g's. This lemma continues to hold even if the actual queries are chosen by a helpful(but untrusted) genie who knows the answer.

For those with a background in computational complexity theory – this establishes that relative to an oracle $BQP \not\subseteq MA$. MA is the probabilistic generalization of NP. It is conjectured that recursive fourier sampling does not lie in the polynomial hierarchy. In particular, it is an open question to show that, relative to an oracle, recusive fourier sampling does not lie AM or in BPP^{NP}.

4.7 Simon's Algorithm

The Problem

Suppose we are given a black box for computing a 2-to-1 function $f : \mathbf{Z}_2^n \to \mathbf{Z}_2^n$ (from n-bit strings to n-bit strings), with the promise that there is a non-zero string $s \in \mathbf{Z}_2^n \setminus \{0\}$ such that

$$\text{for all } x \neq y,\ f(x) = f(y) \text{ if and only if } x \oplus y = s.$$

Here $\oplus$ is the bitwise direct sum modulo 2. For example, $\begin{array}{r} 1101 \\ \oplus 0111 \\ \hline 1010 \end{array}$ or $\begin{array}{r} 01 \\ \oplus 01 \\ \hline 00 \end{array}$.

A quick example of such a function with $n = 3$ is

$$f(x) = \begin{cases} 001 & \text{if } x = 000 \text{ or } 011 \\ 010 & \text{if } x = 001 \text{ or } 010 \\ 100 & \text{if } x = 111 \text{ or } 100 \\ 111 & \text{if } x = 110 \text{ or } 101 \end{cases}$$

where $s = 011$.

The *problem* of Simon's algorithm is to determine s.

Classically

A simple way to solve this problem classically would be to randomly input values to the black box until we find two inputs that produce the same output, and compute their direct sum. But there are 2^{n-1} possible outputs, so despite the help from the birthday paradox, we expect it to take $\sqrt{2^{n-1}} = 2^{(n-1)/2}$ attempts to find s: still exponential time.

Furthermore, it can be shown that no classical computer can find s faster than exponential time. We will use the power of quantum computing to find a faster way.

Quantum

To utilize the power of quantum computing, we will access the function in superposition. So suppose instead of a black box we are given the circuit C_f for computing $|f\rangle$, from which we can construct the unitary transformation U_f:

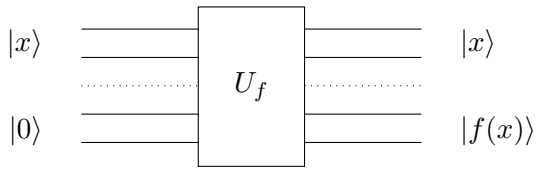


Figure 4.2: Black Box Circuit

The point here is that the input can be a superposition over all n-bit strings $\sum_{x=0}^{N-1} \alpha_x |x\rangle$ $(N = 2^n)$, yielding the output $\sum_{x=0}^{N-1} \alpha_x |x\rangle |f(x)\rangle$. This can be thought of as querying f in superposition.

Simon's Algorithm consists of 3 main steps.

Step 1: Prepare the random superposition $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle)$

Step 2: Use Fourier sampling to produce a y such that $y \cdot s = 0$

Step 3: Repeat until there are enough such y's that we can classically solve for s.

Now lets see the details on how to do each step.

Step 1: Prepare the random superposition $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle)$

First query the function with a uniform superposition of the n-bit strings. To prepare this uniform superposition, start with the state $|0\rangle$ then apply the Hadamard transform. With $N = 2^n$, this is written:

$$|0\rangle |0\rangle \xrightarrow{H^{\otimes n}} \sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} |x\rangle |0\rangle$$

Next we will use the unitary transformation U_f to query f in uniform superposition.

$$\sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} |x\rangle |0\rangle \xrightarrow{U_f} \sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} |x\rangle |f(x)\rangle$$

Now what happens if we measure the second register containing $|f(x)\rangle$? It must collapse into $|f(x_0)\rangle$ for some $x_0 \in \mathbf{Z}_2^n$. But this reveals information

about the first register, and it will also collapse into the pre-images of $f(x_0)$: x_0 and $x_0 \oplus s$.

$$\sqrt{\frac{1}{N}} \sum_{x=0}^{N-1} |x\rangle |f(x)\rangle \stackrel{\text{measure } f}{\longrightarrow} \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) |f(x_0)\rangle$$

The first register is now the state $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle)$ where x_0 is a random n-bit string. The challenge is to read off s from this superposition. We cannot simply measure the state because the superposition will be destroyed, and the result we get will have no information about s.

Step 2: Use Fourier sampling to find a y such that $y \cdot s = 0$.

We now show that $H^{\otimes 2}(\frac{1}{\sqrt{2}} |x_0\rangle + \frac{1}{\sqrt{2}} |x_0 \oplus s\rangle$ is a uniform superposition over all states $|y\rangle$ such that $y \cdot s = 0$. This means that Fourier sampling $(\frac{1}{\sqrt{2}} |x_0\rangle + \frac{1}{\sqrt{2}} |x_0 \oplus s\rangle$ results in a uniform superposition of y such that $y \cdot s = 0$. Recall that

$$H^{\otimes n} |x\rangle = \sqrt{\frac{1}{N}} \sum_y \alpha_y |y\rangle \text{ where } \alpha_y = (-1)^{x \cdot y}$$

So $H^{\otimes 2} \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) = \frac{1}{2} \sum_y \alpha_y |y\rangle$ where

$$\begin{aligned} \alpha_y &= \frac{1}{\sqrt{2}}(-1)^{x_0 \cdot y} + (-1)^{(x_0 \oplus s) \cdot y} \\ &= \frac{1}{\sqrt{2}}(-1)^{x_0 \cdot y}(1 + (-1)^{s \cdot y}) \end{aligned}$$

Now it is easy to see that if $s \cdot y = 0$, $\alpha_y = \pm \frac{1}{\sqrt{2}}$, but if $s \cdot y = 1$, $\alpha_y = 0$.

Therefore when we measure the first register, we will measure a y such that $y \cdot s = 0$.

Step 3: Repeat until there are enough such y's that we can classically solve for s.

There are exactly n linearly independent values of y such that $y \cdot s = y_1 s_1 + y_2 s_2 + \cdots y_n s_n = 0$, and one of these is the trivial solution $y = 0$. Therefore, there are $n-1$ non-trivial, linearly independent solutions to $y \cdot s = 0$. But if y_1 and y_2 are linearly independent solutions, $(y_1 + y_2) \cdot s = y_1 \cdot s + y_2 \cdot s = 0$ so linear combinations of solutions are also solutions. This gives us a total of 2^{n-1} y's such that $y \cdot s = 0$. To solve for s, we need to find exactly $n - 1$ non-trivial, linearly independent y such that $y \cdot s = 0$.

For example, if $s = 010$, then $y_0 = 000$, $y_1 = 001$, and $y_2 = 100$ are linearly independent solutions to $y \cdot s = 0$. But the linear combination $y_1 + y_2 = 101$ is

also a solution. We need only find two of $\{y_1, y_2, y_1+y_2\}$ in order to classically solve for s.

How long should we expect this to take? The probability that we fail on the first run is the probability that we find $y = 0$, which is one value out of 2^{n-1}. So $P_1 = 1/2^{n-1}$, where P_1 denotes the probability of failing on the first run. Lets call the first nontrivial solution y_1.

We fail when looking for y_2 if we find 0 or y_1, so $P_2 = 2/2^{n-1} = 1/2^{n-2}$. When looking for y_3, we fail if we find any of $\{0, y_1, y_2, y_1 + y_2\}$, so $P_3 = 4/2^{n-1} = 1/2^{n-3}$. Carrying on in this way, the probability of failing to find y_i is $P_i = 1/2^{n-i}$.

The chance that we fail *up to and including* y_i can be approximated by $P < 1/2^{n-1} + 1/2^{n-2} + \cdots + 1/2^{n-i}$. If we push this approximation all the way to $i = n - 1$, we see that we fail with probability less than 1 (compute the geometric sum). That's not a strong enough approximation, so instead notice that our probability of failure up to and including $i = n - 2$ is less than 1/2. Then our probability of success up to the $n - 2$st run is greater than 1/2. We find the final linearly independent term on the last run with probability 1/2 (if you don't believe this, notice that half of the solutions are linear combinations that include y_{n-1}). Finally our total probability of success is $P(success) > 1/2 * 1/2 = 1/4$. Therefore, we expect our process to take $O(n)$ steps (our limit says 4 by n runs of the algorithm should be enough for success).

Simon's algorithm is summed up by the following circuit.

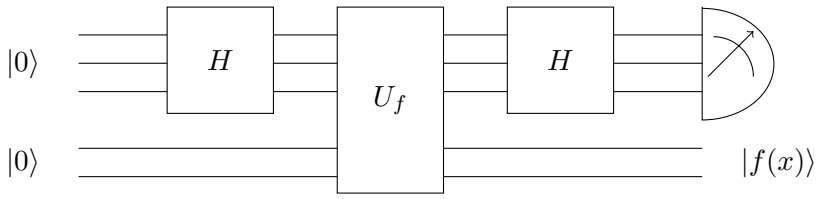


Figure 4.3: Circuit for Simon's Algorithm

In summary, the above circuit for Simon's algorithm corresponds to the

following sequence of transformations.

$$\begin{aligned}
|0\rangle\,|0\rangle &\xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}}\sum_x |x\rangle\,|0\rangle \\
&\xrightarrow{U_f} \frac{1}{\sqrt{2^n}}\sum_x |x\rangle\,|f(x)\rangle \\
&\xrightarrow{\text{measure}} \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus s\rangle) \otimes |f(x_0)\rangle \\
&\xrightarrow{H^{\otimes n}} \frac{1}{\sqrt{2^n}}\sum_y \alpha_y\,|y\rangle\,|f(x_0)\rangle
\end{aligned}$$

for some numbers α_y.

As above, for each y, if $s \cdot y = 1$, then $\alpha_y = 0$, whereas if $s \cdot y = 0$, then $\alpha_y = (-1)^{x_0 \cdot y}\sqrt{2}$.

When we observe the first register, we get a uniformly random y such that $s \cdot y = s_1 y_1 + \cdots + s_n y_n = 0$. We repeat to collect more and more equations, and recover s from $n-1$ linearly independent equations.

Example

Let $n = 2$ and $f(x) = \begin{cases} 00 \text{ if } x = 00 \text{ or } 10 \\ 01 \text{ if } x = 01 \text{ or } 11 \end{cases}$ so that $s = 10$.

First, apply the Hadamard transform to prepare $|x\rangle$, then U_f to prepare $|f(x)\rangle$

$$|0\rangle\,|0\rangle \xrightarrow{H^{\otimes 2}} \frac{1}{2}\sum_{x=0}^{3} |x\rangle\,|0\rangle \xrightarrow{U_f} \frac{1}{2}\sum_{x=0}^{3} |x\rangle\,|f(x)\rangle$$

Then measure the second register to finalize the first step of the process. For the purpose of argument, lets suppose we measure $f(x) = 01$, so that $x_0 = 01$ and $x_0 \oplus s = 11$:

$$\frac{1}{2}\sum_{x=0}^{3} |x\rangle\,|f(x)\rangle \xrightarrow{\text{measure}} \frac{1}{\sqrt{2}}(|01\rangle + |11\rangle) \otimes |01\rangle$$

We then impose the Hadamard transform to achieve:

$$\frac{1}{2}\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \end{pmatrix}$$

We expect it will take 2 runs through the above process to measure $y = 01$. Then the only nonzero solution for s is $s = 10$.