PHYS440 - Exercises from Nielsen and Chuang (2016)

John Hurst

2023/2024

3.1. (Non-computable processes in Nature) How might we recognize that a process in Nature computes a function not computable by a Turing machine?

If a process in Nature contains any random element, then it cannot be exactly computed by a Turing machine, because Turing machines are completely deterministic. (It may be possible to approximate or simulate the process with a Turing machine, using pseudorandom numbers.)

If a process in Nature is continuous, then it cannot be computed by a Turing machine, because Turing machines are discrete. (It may be possible to approximate the process to arbitrary accuracy, using rational numbers.)

3.2. (Turing numbers) Show that single-tape Turing machines can each be given a number from the list 1, 2, 3, ... in such a way that the number uniquely specifies the corresponding machine. We call this number the *Turing number* of the corresponding machine.

Let's suppose that our machine has the minimal alphabet $\Gamma = \{0, 1, \square, \triangleright\}$, and that there are m states $q_1, q_2, \ldots, q_m$, besides the special states q_s, q_h.

With these fixed, a given Turing machine is uniquely determined by its *program*, which is an ordered list of program lines:

$$\begin{aligned}
&\langle r_1, x_1, r_1', x_1', s_1\rangle\\
&\langle r_2, x_2, r_2', x_2', s_2\rangle\\
&\langle r_3, x_3, r_3', x_3', s_3\rangle\\
&\cdots\\
&\langle r_n, x_n, r_n', x_n', s_n\rangle
\end{aligned}$$

where

$$\begin{aligned}
r_k, r_k' &\in \{q_1, q_2, \ldots, q_m, q_s, q_h\} & k &\in \{1, 2, \ldots, n\}\\
x_k, x_k' &\in \{0, 1, \square, \triangleright\} & k &\in \{1, 2, \ldots, n\}\\
s_k &\in \{-1, 0, 1\} & k &\in \{1, 2, \ldots, n\}
\end{aligned}$$

We can assign a unique number N to each possible program by

$$N = \prod_{k=1}^{n} p_k^{w_k}$$

where

$$
\begin{aligned}
p_k &= k\text{th prime number} \\
w_k &= 2^{a_k} \times 3^{b_k} \times 5^{c_k} \times 7^{d_k} \times 11^{e_k} \\
a_k &= \begin{cases} i & \text{if } r_k = q_i, i \in \{1, 2, \ldots, m\} \\ m+1 & \text{if } r_k = q_s \\ m+2 & \text{if } r_k = q_h \end{cases} \\
b_k &= \begin{cases} 1 & \text{if } x_k = 0 \\ 2 & \text{if } x_k = 1 \\ 3 & \text{if } x_k = \square \\ 4 & \text{if } x_k = \triangleright \end{cases} \\
c_k &= \begin{cases} i & \text{if } r'_k = q_i \\ m+1 & \text{if } r'_k = q_s \\ m+2 & \text{if } r'_k = q_h \end{cases} \\
d_k &= \begin{cases} 1 & \text{if } x'_k = 0 \\ 2 & \text{if } x'_k = 1 \\ 3 & \text{if } x'_k = \square \\ 4 & \text{if } x'_k = \triangleright \end{cases} \\
e_k &= \begin{cases} 1 & \text{if } s_k = -1 \\ 2 & \text{if } s_k = 0 \\ 3 & \text{if } s_k = 1 \end{cases}
\end{aligned}
$$

3.3. (Turing machine to reverse a bit string) Describe a Turing machine which takes a binary number x as its input, and outputs the bits of x in reverse order.

The hint suggests to use a multi-tape Turing machine and additional symbols. We can do it reasonably with a single tape, with some extra symbols X and Z.

We'll describe a Turing machine that does these steps:

1. Append separator X and terminator Z.
2. Remove digit from end of input and append to output.
3. Repeat until no more input.
4. Remove separator X.

5. Shift output to beginning of tape.
6. Remove terminator.

We will use these states:

- ✿ `ax`: Initialise: append separator ’X’ to end of input
- ✿ `az`: Append terminator ’Z’ to end of tape
- ✿ `bx`: Reverse from ’Z’ to ’X’
- ✿ `bi`: Reverse from ’X’ to end of input
- ✿ `m0`: Move ’0’ from input to end of output
- ✿ `m1`: Move ’1’ from input to end of output
- ✿ `rx`: Remove separator ’X’
- ✿ `so`: Shift output
- ✿ `s0`: Shift ’0’ digit
- ✿ `p0`: Put shifted ’0’ digit
- ✿ `s1`: Shift ’1’ digit
- ✿ `p1`: Put shifted ’1’ digit

Step 1: Append separator X and terminator Z:

$$\begin{aligned}
\langle qs, start, ax, start, +1\rangle \\
\langle ax, 0, ax, 0, +1\rangle \\
\langle ax, 1, ax, 1, +1\rangle \\
\langle ax, blank, az, X, +1\rangle \\
\langle az, 0, az, 0, +1\rangle \\
\langle az, 1, az, 1, +1\rangle \\
\langle az, blank, bx, Z, -1\rangle \\
\langle bx, 0, bx, 0, -1\rangle \\
\langle bx, 1, bx, 1, -1\rangle \\
\langle bx, X, bi, X, -1\rangle \\
\langle bi, blank, bi, blank, -1\rangle
\end{aligned}$$

Step 2: Remove digit from end of input and append to output. Step 3: Repeat until no more input.

$$\langle bi, 0, m0, blank, +1\rangle$$
$$\langle m0, blank, m0, blank, +1\rangle$$
$$\langle m0, X, m0, X, +1\rangle$$
$$\langle m0, 0, m0, 0, +1\rangle$$
$$\langle m0, 1, m0, 1, +1\rangle$$
$$\langle m0, Z, az, 0, +1\rangle$$
$$\langle bi, 1, m1, blank, +1\rangle$$
$$\langle m1, blank, m1, blank, +1\rangle$$
$$\langle m1, X, m1, X, +1\rangle$$
$$\langle m1, 0, m1, 0, +1\rangle$$
$$\langle m1, 1, m1, 1, +1\rangle$$
$$\langle m1, Z, az, 1, +1\rangle$$

Step 4: Remove separator 'X'.

$$\langle bi, start, rx, start, +1\rangle$$
$$\langle rx, blank, rx, blank, +1\rangle$$
$$\langle rx, X, so, blank, +1\rangle$$

Step 5: Shift output to beginning of tape.

$$\langle so, blank, so, blank, +1\rangle$$
$$\langle so, 0, s0, blank, -1\rangle$$
$$\langle s0, blank, s0, blank, -1\rangle$$
$$\langle s0, start, p0, start, +1\rangle$$
$$\langle s0, 0, p0, 0, +1\rangle$$
$$\langle s0, 1, p0, 1, +1\rangle$$
$$\langle p0, blank, so, 0, +1\rangle$$
$$\langle so, 1, s1, blank, -1\rangle$$
$$\langle s1, blank, s1, blank, -1\rangle$$
$$\langle s1, start, p1, start, +1\rangle$$
$$\langle s1, 0, p1, 0, +1\rangle$$
$$\langle s1, 1, p1, 1, +1\rangle$$
$$\langle p1, blank, so, 1, +1\rangle$$

Step 6: Remove terminator.

$$\langle so, Z, qh, blank, 0\rangle$$

Given the input 11000, the machine will progress like this:

$$\begin{array}{ccccccccccccc}
\triangleright & 1 & 1 & 0 & 0 & 0 & & & & & & & \\
\triangleright & 1 & 1 & 0 & 0 & 0 & X & & & & & & \\
\triangleright & 1 & 1 & 0 & 0 & 0 & X & Z & & & & & \\
\triangleright & 1 & 1 & 0 & 0 & \square & X & 0 & Z & & & & \\
\triangleright & 1 & 1 & 0 & \square & \square & X & 0 & 0 & Z & & & \\
\triangleright & 1 & 1 & \square & \square & \square & X & 0 & 0 & 0 & Z & & \\
\triangleright & 1 & \square & \square & \square & \square & X & 0 & 0 & 0 & 1 & Z & \\
\triangleright & \square & \square & \square & \square & \square & X & 0 & 0 & 0 & 1 & 1 & Z \\
\triangleright & \square & \square & \square & \square & \square & \square & 0 & 0 & 0 & 1 & 1 & Z \\
\triangleright & 0 & \square & \square & \square & \square & \square & \square & 0 & 0 & 1 & 1 & Z \\
\triangleright & 0 & 0 & \square & \square & \square & \square & \square & \square & 0 & 1 & 1 & Z \\
\triangleright & 0 & 0 & \square & \square & \square & \square & \square & \square & \square & 1 & 1 & Z \\
\triangleright & 0 & 0 & 0 & \square & \square & \square & \square & \square & \square & \square & 1 & Z \\
\triangleright & 0 & 0 & 0 & 1 & \square & \square & \square & \square & \square & \square & \square & Z \\
\triangleright & 0 & 0 & 0 & 1 & 1 & \square & \square & \square & \square & \square & \square & Z \\
\triangleright & 0 & 0 & 0 & 1 & 1 & & & & & & &
\end{array}$$

3.8. (Universality of NAND) Show that the NAND gate can be used to simulate the AND, XOR and NOT gates, provided wires, ancilla bits and FANOUT are available.

De Morgan's laws are:

$$\begin{aligned}
\mathtt{NOT}(x \text{ AND } y) &= \mathtt{NOT}x \; \mathtt{OR} \; \mathtt{NOT}y \\
\mathtt{NOT}(x \; \mathtt{OR} \; y) &= \mathtt{NOT}x \text{ AND } \mathtt{NOT}y
\end{aligned}$$

We can obtain NOT from NAND:

$$\begin{aligned}
x \; \mathtt{NAND} \; x &= \mathtt{NOT}(x \text{ AND } x) \\
&= \mathtt{NOT}x
\end{aligned}$$

Now we can get OR from NAND and NOT:

$$\begin{aligned}
(\mathtt{NOT}x) \; \mathtt{NAND} \; (\mathtt{NOT}y) &= \mathtt{NOT}((\mathtt{NOT}x) \text{ AND } (\mathtt{NOT}y)) \\
&= \mathtt{NOT}(\mathtt{NOT}(x \; \mathtt{OR} \; y)) \\
&= x \; \mathtt{OR} \; y
\end{aligned}$$

AND can be defined from from NOT and OR:

$$\begin{aligned}\texttt{NOT}((\texttt{NOT}x)\ \texttt{OR}\ (\texttt{NOT}y)) &= \texttt{NOT}(\texttt{NOT}x)\ \texttt{AND}\ \texttt{NOT}(\texttt{NOT}y)\\ &= x\ \texttt{AND}\ y\end{aligned}$$

Finally, XOR can be defined from AND and OR:

$$\begin{aligned}(x\ \texttt{AND}\ \texttt{NOT}y)\ \texttt{OR}\ (\texttt{NOT}x\ \texttt{AND}\ y) &= (x\ \texttt{AND}\ y)\ \texttt{OR}\ (\texttt{NOT}x\ \texttt{AND}\ \texttt{NOT}y)\\ &= x\ \texttt{XOR}\ y\end{aligned}$$

3.9. Prove that $f(n)$ is $O(g(n))$ if and only if $g(n)$ is $\Omega(f(n))$. Deduce that $f(n)$ is $\Theta(g(n))$ if and only if $g(n)$ is $\Theta(f(n))$.

Suppose $f(n)$ is $O(g(n))$. Then there exist constants c and n_0 such that $f(n) \le cg(n)$ for all $n \ge n_0$. Then $g(n) \ge \frac{1}{c} f(n)$ for all $n \ge n_0$, so $g(n)$ is $\Omega(f(n))$.

Similarly, if $g(n)$ is $\Omega(f(n))$, then there exist constants c and n_0 such that $g(n) \ge cf(n)$ for all $n \ge n_0$. Then $f(n) \le \frac{1}{c} g(n)$ for all $n \ge n_0$, so $f(n)$ is $O(g(n))$.

If $f(n)$ is $\Theta(g(n))$, then $f(n)$ is $O(g(n))$ and $\Omega(g(n))$. Then

$$\begin{aligned} f(n) \text{ is } O(n) &\implies g(n) \text{ is } \Omega(f(n))\\ f(n) \text{ is } \Omega(n) &\implies g(n) \text{ is } O(f(n))\end{aligned}$$

And so $g(n)$ is $\Theta(f(n))$.

3.10. Suppose $g(n)$ is a polynomial of degree k. Show that $g(n)$ is $O(n^l)$ for any $l \ge k$.

Let $g(n) = \sum_{j=0}^{k} a_j n^j$, and then choose $c = k \times \max\{|a_0|, |a_1|, \ldots, |a_k|\}$. Then

$$\begin{aligned} g(n) &= \sum_{j=0}^{k} a_j n^j\\ &\le \sum_{j=0}^{k} \max\{|a_i|\} n^l\\ &= cn^l\end{aligned}$$

Therefore $g(n)$ is $O(n^l)$.

3.11. Show that $\log n$ is $O(n^k)$ for any $k > 0$.

Consider $\lim_{n\to\infty} \frac{n^k}{\log n}$:

$$\begin{aligned}
\lim_{n\to\infty} \frac{n^k}{\log n} &= \lim_{n\to\infty} \frac{kn^{k-1}}{\frac{1}{n}} && \text{(L'Hôpital's rule)} \\
&= \lim_{n\to\infty} kn^k \\
&= \infty
\end{aligned}$$

Therefore $\log n$ is $O(n^k)$.

3.12. ($n^{\log n}$ is super-polynomial) Show that n^k is $O(n^{\log n})$ for any k, but that $n^{\log n}$ is never $O(n^k)$.

For the first part, take $n_0 = \lceil e^k \rceil$. Then $\log n_0 \geq k$, and for $n > n_0$:

$$\begin{aligned}
\log n &> k \\
n^{\log n} &> n^k
\end{aligned}$$

Therefore n^k is $O(n^{\log n})$.

For the second part, suppose $n^{\log n}$ is $O(n^k)$. Then there exist constants c and n_0 such that:

$$n^{\log n} \leq cn^k \qquad \text{for all } n > n_0$$

Take $n_1 = \max\{\lceil e^{k+1} \rceil, \lceil c \rceil\}$. Then:

$$\begin{aligned}
n^{\log n} &> n^{\log n_1} && \text{for all } n > n_1 \\
&\geq n^{k+1} \\
&\geq cn^k
\end{aligned}$$

which contradicts the assumption that $n^{\log n}$ is $O(n^k)$.

3.13. ($n^{\log n}$ is sub-exponential) Show that k^n is $\Omega(n^{\log n})$ for any $k > 1$, but that $n^{\log n}$ is never $\Omega(k^n)$.

TODO: Question 3.13.

3.14. Suppose $e(n)$ is $O(f(n))$ and $g(n)$ is $O(h(n))$. Show that $e(n)g(n)$ is $O(f(n)h(n))$.

Because $e(n)$ is $O(f(n))$, there exist constants c_1 and n_1 such that $e(n) \le c_1 f(n)$ for all $n > n_1$. Similarly, there exist constants c_2 and n_2 such that $g(n) \le c_2 h(n)$ for all $n > n_2$. Let $c = c_1 c_2$ and $n_0 = \max\{n_1, n_2\}$. Then for $n > n_0$:

$$\begin{aligned} e(n)g(n) &\le c_1 c_2 f(n)h(n) \\ &= c f(n)h(n) \end{aligned}$$

Therefore $e(n)g(n)$ is $O(f(n)h(n))$.

3.15. (Lower bound for compare-and-swap based sorts) Suppose an n element list is sorted by applying some sequence of compare-and-swap operations to the list. There are $n!$ possible initial orderings of the list. Show that after k of the compare-and-swap operations have been applied, at most 2^k of the possible initial orderings will have been sorted into the correct order. Conclude that $\Omega(n \log n)$ compare-and-swap operations are required to sort all possible initial orderings into the correct order.

For a single compare-and-swap operations, there are two possible outcomes: either the elements are swapped, or they are not. For k compare-and-swap operations, there are 2^k possible outcomes. Therefore, after k compare-and-swap operations, at most 2^k of the possible initial orderings will have been sorted into the correct order.

For a list of n elements, with $n!$ possible initial orderings, we need at least m compare-and-swap operations, where m is the smallest integer such that $2^m \ge n!$.

That is:

$$
\begin{aligned}
2^m &\geq n! \\
2^m &\geq \sqrt{2\pi n}\left(\frac{n}{e}\right)^n && \text{(Stirling's approximation)} \\
m\log 2 &\geq \log\left[\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right] && \text{(Taking logarithm)} \\
&= \log\sqrt{2\pi n} + n\log n - n\log e \\
n\log n &\leq m\log 2 - \log\sqrt{2\pi n} + n\log e \\
\frac{1}{\log 2} n\log n &\leq m + n\frac{\log e}{\log 2}
\end{aligned}
$$

Apart from the extra $n\frac{loge}{\log 2}$, this matches the definition of m is $\Omega(n\log n)$.

TODO: Can we account for the extra term?

3.16. (Hard-to-compute functions exist) Show that there exist Boolean functions on n inputs which require at least $2^n/\log n$ logic gates to compute.

There are 2^{2^n} possible Boolean functions on n inputs.

TODO: Question 3.16.

3.17. Prove that a polynomial-time algorithm for finding the factors of a number m exists if and only if the factoring decision problem is in **P**.

3.18. Prove that if **coNP** $\neq$ **NP**, then **P** $\neq$ **NP**.

3.19. The reachability problem is to determine whether there is a path between two specified vertices in a graph. Show that reachability can be solved using $O(n)$ operations if the graph has n vertices. Use the solution to reachability to show that it is possible to decide whether a graph is connected in $O(n^2)$ operations.

3.20. (Euler's theorem) Prove Euler's theorem. In particular, if vertex has an even number of incident edges, give a constructive procedure for finding a Euler cycle.

3.21. (Transitive property of reduction) Show that if a language L_1 is reducible to the language L_2 and the language L_2 is reducible to L_3 then the language L_1 is reducible to the language L_3.

3.22. Suppose L is complete for a complexity class, and L' is another language in the complexity class such that L reduces to L'. Show that L' is complete for the complexity class.