QUANTUM ALGORITHMS
HOMEWORK 2 SELECTED SOLUTIONS

PROF. MATTHEW MOORE

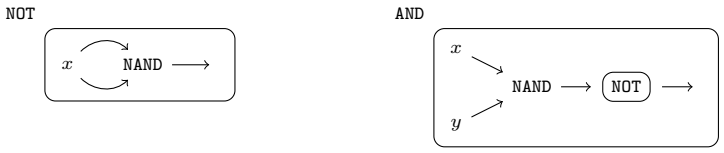
AP 1. Find a complete Boolean basis $\mathcal{A}$ of size 1 (that is, it consists of just a single function). Prove your answer is correct.

Solution: Define $\texttt{NAND}(x, y) = \neg(x \wedge y)$ and let $\mathcal{A} = \{\texttt{NAND}\}$. We will prove that $\mathcal{A}$ is a complete Boolean basis.

Claim. *$\mathcal{A} = \{\texttt{NAND}\}$ is a complete Boolean basis.*

Proof of claim. We know that $\mathcal{B} = \{\texttt{NOT}, \texttt{AND}\}$ is a complete basis, so it is sufficient to show that we can express NOT and AND as circuits over $\mathcal{A}$.

We have that $\texttt{NOT}(x) = \texttt{NAND}(x, x)$ and that $\texttt{AND}(x, y) = \texttt{NOT} \circ \texttt{NAND}(x, y)$, as illustrated by the circuits below.



Since $\mathcal{A}$ can express each function in the complete basis $\mathcal{B}$ as a circuit, it follows that $\mathcal{A}$ is also a complete basis. ∘

AP 2. An n-ary function $f : \mathbb{B}^n \to \mathbb{B}$ is *idempotent* if

$$f(0, \ldots, 0) = 0 \qquad \text{and} \qquad f(1, \ldots, 1) = 1.$$

Find a basis $\mathcal{A}$ so that every idempotent Boolean function is representable as a circuit over $\mathcal{A}$. Prove your answer is correct. [*Hint 1: Post's Lattice.*] [*Hint 2:* ? :.]

Solution: The idempotent functions are those functions which "preserve" the all 0 tuple and the all 1 tuple, in the language of Post's lattice. They correspond to the set P in the Wikipedia article on it. In that same article, a basis for the set is given — ? :, the "inline if" statement. It is defined as follows

$$x \mathbin{?} y : z = \begin{Bmatrix} y & \text{if } x = 1, \\ z & \text{otherwise} \end{Bmatrix} = \text{“if } x \text{ then } y, \text{ else } z\text{”}.$$

Let $\mathcal{I} = \{? :\}$. We will prove that every idempotent function can be generated by a circuit over $\mathcal{I}$.

Claim. *$\mathcal{I}$ is a complete basis for the set of idempotent functions.*

Date: February 3, 2020.

Proof of claim. Similar to how we proved $\{\texttt{NOT}, \texttt{AND}, \texttt{OR}\}$ is a complete basis for all functions, the proof shall be by induction the function we wish to express as a circuit over $\mathcal{I}$. Let $f : \mathbb{B}^n \to \mathbb{B}$ be idempotent. We proceed by induction on n.

For the base case of $n = 1$, there is only one idempotent function, namely $f(x) = x$. We have that

$$f(x) = x = x \ ? \ x : x,$$

so $f(x)$ is a circuit in $\mathcal{I}$, establishing the base case. In the argument below, we implicitly assume that $n \geq 3$ (additionally, we make use of $\texttt{AND}$), so we will also need to prove the claim for 2-ary functions. There are just four 2-ary idempotent functions:

(x, y)	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
$g_1(x, y)$	0	0	0	1
$g_2(x, y)$	0	0	1	1
$g_3(x, y)$	0	1	0	1
$g_4(x, y)$	0	1	1	1

Observe that $g_2(x, y) = x$ and $g_3(x, y) = y$, functions already covered by the base case. A closer look at g_1 and g_4 reveals that $g_1(x, y) = x \wedge y$ and $g_4(x, y) = x \vee y$. We have that

$$x \ ? \ y : x = x \wedge y = g_1(x, y) \qquad \text{and} \qquad x \ ? \ x : y = x \vee y = g_4(x, y).$$

This establishes the claim for 2-ary functions.

Suppose now that we have proven that every n-ary idempotent function is expressible as a circuit over $\mathcal{I}$, and that f is $(n + 1)$-ary and idempotent. Let us imagine evaluating f on some arguments, say $f(a_1, \ldots, a_{n+1})$. Looking at the first 3 arguments of f, there must be two of these values which are equal. Therefore, $f(a_1, a_2, a_3, \ldots, a_{n+1})$ is equal to one of

$$f(a_1, a_1, a_3, \ldots, a_{n+1}), \quad f(a_1, a_2, a_1, \ldots, a_{n+1}), \quad \text{or} \quad f(a_1, a_2, a_2, \ldots, a_{n+1})$$

for this particular input. Define n-ary functions f_{12}, f_{13}, f_{23} by

$$\begin{aligned} f_{12}(x_1, x_2, x_3, \ldots, x_n) &= f(x_1, x_1, x_2, x_3, \ldots, x_n), \\ f_{13}(x_1, x_2, x_3, \ldots, x_n) &= f(x_1, x_2, x_1, x_3, \ldots, x_n), \\ f_{23}(x_1, x_2, x_3, \ldots, x_n) &= f(x_1, x_2, x_2, x_3, \ldots, x_n). \end{aligned}$$

We now design a circuit to test which two of x_1, x_2, x_3 are equal and select the appropriate f_{ij}. Define

$$A(z) = (x_1 \wedge x_2) \ ? \ f_{12}(x_1, x_3, x_4, \ldots, x_{n+1}) : \big((x_1 \wedge x_2 \ ? \ z : f_{12}(x_1, x_3, x_4, \ldots, x_{n+1})\big).$$

Note that if $x_1 = x_2$, then

$$\begin{aligned} A(z) &= f_{12}(x_1, x_3, x_4, \ldots, x_n) = f(x_1, x_1, x_3, x_4, \ldots, x_{n+1}) \\ &= f(x_1, x_2, x_3, x_4, \ldots, x_{n+1}), \end{aligned}$$

and if $x_1 \neq x_2$ then $A(z) = z$. Continuing in this vein, define

$$B(z) = (x_1 \wedge x_3) \ ? \ f_{13}(x_1, x_2, x_4, \ldots, x_{n+1}) : \big((x_1 \wedge x_3 \ ? \ z : f_{13}(x_1, x_2, x_4, \ldots, x_{n+1})\big)$$

and

$$C(z) = (x_2 \wedge x_3) \ ? \ f_{23}(x_1, x_2, x_4, \ldots, x_{n+1}) : \big((x_2 \wedge x_3 \ ? \ z : f_{23}(x_1, x_2, x_4, \ldots, x_{n+1})\big).$$

It's not difficult to show that

$$f(x_1, x_2, x_3, x_4, \ldots, x_{n+1}) = A \circ B \circ C(x_1)$$

(the x_1 argument is immaterial — the circuit will never go down that branch). Since each of A, B, and C involves functions of arity at most n, the inductive hypothesis applies and we can construct circuits over $\mathcal{I}$ for each of them. It follows that f is representable as a circuit over $\mathcal{I}$. ∘

2.1. Construct an algorithm that determines whether a given set of Boolean functions $\mathcal{A}$ constitutes a complete basis. (Functions are represented by tables.)

Solution: We will provide an "algebraic" algorithm that is quite different from the one in the textbook. From Additional Problem 1, we know that $\mathcal{A}$ is complete if and only if NAND is definable in terms of the operations from $\mathcal{A}$.

Define vectors in $\{0,1\}^4$,

$$a = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \qquad b = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}, \qquad c = \texttt{NAND}(a,b) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

Next, define a sequence of subsets of $\{0,1\}^4$,

$$X_0 = \{a, b\}, \qquad X_{n+1} = X_n \cup \Big\{ f(c_1, \ldots, c_n) \mid f \in \mathcal{A} \text{ is } n\text{-ary}, \, c_1, \ldots, c_n \in X_n \Big\}.$$

Since $\{0,1\}^4$ is a finite set, this sequence of sets must eventually stop growing in size. Let S be the largest of the sets X_n.

Claim. *NAND is definable from $\mathcal{A}$ if and only if $c \in S$.*

Proof of claim. We begin by observing that every element in S can be written as $t(a,b)$, where t is some formula in terms of the symbols from $\mathcal{A}$.

Suppose that NAND is definable from $\mathcal{A}$. This implies that there is a formula $s(x,y)$ in terms of the symbols from $\mathcal{A}$ such that $s(x,y) = \texttt{NAND}(x,y)$. Since $c = \texttt{NAND}(a,b) = s(a,b)$, by the observation at the beginning of the proof we have $c \in S$.

Suppose that $c \in S$. By the observation at the beginning of the proof, we have that there is a term $s(x,y)$ in terms for the symbols from $\mathcal{A}$ such that $s(a,b) = c$. Writing this out, we have

$$s \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}.$$

This is just the definition of the NAND function though, so it follows that $s(x,y) = \texttt{NAND}(x,y)$.

This construction is called the *free algebra* construction. The set S contains all 2-variable functions which are definable from $\mathcal{A}$. ∘

From the construction prior to the claim, it should be clear that this procedure is computable and therefore constitutes an algorithm — generate the sequence X_n until the sets stop growing (this requires at most $2^4 = 16$ steps), then check for the presence of c in the final set.

3.3. Suppose we have an **NP**-oracle — a magic device that can immediately solve any instance of the **SAT** problem for us. In other words, for any propositional formula the oracle tells whether it is satisfiable or not. Prove that there is a polynomial-time algorithm that finds a satisfying assignment to a given formula by making a polynomial number of queries to the oracle. (A similar statement is true for the Hamiltonian cycle: finding a Hamiltonian cycle in a graph is at most polynomially harder than checking for its existence.)

Solution: Consider the algorithm below.

```
define solve_SAT(S):
  if not is_satisfiable(S):
    return False

  for each variable x_i in S:
    if is_satisfiable(S ∧ x_i):
      assign "True" to x_i
      let S = S ∧ x_i
    elif is_satisfiable(S ∧ (¬x_i)):
      assign "False" to x_i
      let S = S ∧ (¬x_i)
    else:   # should never get here!
      return False

  return the variable assignments
```

There will be a satisfying assignment for S with x_i true if and only if $S \wedge x_i$ is satisfiable.

A common error was to somehow include literals when modifying S, or to fail to update S once a value of x_i has been found. It is possible for x_i to be true in some assignment and x_j to be true in some assignment, but for there to not be an assignment where *both* are true:

$$\big(x_i \vee x_j\big) \wedge \big(\neg x_i \vee \neg x_j\big).$$