Finite Computation

I. DEFINING COMPUTATION

Computers are stupid. We need to describe algorithms precisely.

Informal definition of an algorithm: An Algorithm is a set of instructions of how to compute an output from an input by following a sequence of "elementary steps". An algorithm A computes a function F if for every input x, if we follow the instructions of A on the input x, we obtain the output $F(x)$.

A. Boolean circuit

Definition 1 (Boolan circuits) *Let n, m, s be positive integers with $s \geq m$. A Boolean circuit with n inputs, m outputs, and s gates, is a labeled directed acyclic graph (DAG) $G = (V, E)$ with $s + n$ vertices satisfying the following properties:*

1. *Exactly n of the vertices have no in-neighbors. These vertices are know as inputs and are labeled with the n labels: $X[0], \cdots, X[n-1]$.*
2. *The other s vertices are known as gates. Each gate is labeled with $\wedge, \vee$ and $\neg$. Gates labeled with $\wedge$ or $\vee$ have two in-neighbors. Gates labeled with $\neg$ have one in-neighbor. We will allow parallel edges (and so for example an AND gate can have both its in-neighbors be the same vertex).*
3. *Exactly m of the gates are also labeled with the m labels $Y[0], \cdots, Y[m-1]$ (in addition to their label $\wedge, \vee, \neg$). These are known as outputs.*

Let $f : \{0,1\}^n \to \{0,1\}^m$. We say that the circuit C computes f if for every $x \in \{0,1\}^n$, $C(x) = f(x)$.

Briefly speaking, the computation of a Boolean circuit is layer by layer.

Definition 2 (AON-CIRC Programming language) *An AON-CIRC program is a string of lines of the form foo = AND(bar,blah) , foo = OR(bar,blah) and foo = NOT(bar) where foo , bar and blah are variable names. Variables of the form $X[i]$ are known as input variables, and variables of the form $Y[j]$ are known as output variables. In every line, the variables on the righthand side of the assignment operators must either be input variables or variables that have already been assigned a value before.*

Theorem 1 (Equivalence of circuits and straight-line programs) *Let $f : \{0,1\}^n \to \{0,1\}^m$ and $s \geq m$ be some number. Then f is computable by a Boolean circuit of s gates if and only if f is computable by an AON-CIRC program of s lines.*

B. The NAND function

Theorem 2 *NAND computes AND, OR, NOT. We can compute AND, OR, and NOT by composing only the NAND function.*

$$
\begin{aligned}
NOT(a) &= NAND(a, a) \\
AND(a, b) &= NOT(NAND(a, b)) \\
OR(a, b) &= NOT(NOT(NOT(a), NOT(b)))
\end{aligned}
$$

Theorem 3 *NAND is a universal operation. For every Boolean circuit C of s gates, there exists a NAND circuit C' of at most $3s$ gates that computes the same function as C.*

Just like we did for Boolean circuits, we can define a programming-language analog of NAND circuits. It is even simpler than the AON-CIRC language since we only have a single operation.

Theorem 4 (Equivalence of NAND circuits and straight-line programs) *Let $f : \{0,1\}^n \to \{0,1\}^m$ and $s \geq m$ be some number. Then f is computable by a NAND circuit of s gates if and only if f is computable by an NAND-CIRC program of s lines.*

II. SYNTACTIC SUGAR, AND COMPUTING EVERY FUNCTION

"syntactic sugar" in programming language: not changing the definition of the language, but merely introducing some convenient notational shortcuts.1 We will use several such "syntactic sugar" constructs to make our descriptions of NAND-CIRC programs shorter and simpler.

III. SOME EXAMPLES SYNTACTIC SUGAR

- Constants
- Functions / Macros
- Conditional statements: foo = IF(condition,blah, foo)

IV. THE LOOKUP FUNCTION

Definition 3 (Lookup function) *For every k, the lookup function $LOOKUP_k : \{0,1\}^{2^k+k} \to \{0,1\}$ is defined as follows: For every $x \in \{0,1\}^{2^k}$ and $i \in \{0,1\}$,*

$$LOOKUP_k(x, i) = x_i \tag{1}$$

Theorem 5 (Lookup function) *For every k, there is a NAND-CIRC program that computes the function $LOOKUP_k : \{0,1\}^{2^k+k} \to \{0,1\}$. Moreover, the number of lines in this program is at most $4 \cdot 2^k$.*

Lemma 1 (Lookup function) *For every $k \geq 2$, $LOOKUP_k(x_0, \cdots, x_{2^k-1}, i_0, \cdots, i_{k-1})$ is equal to*

$$IF(i_0, LOOKUP_k(x_0, \cdots, x_{2^{k-1}-1}, i_1, \cdots, i_{k-1}), LOOKUP_k(x_{2^{k-1}}, \cdots, x_{2^k-1}, i_1, \cdots, i_{k-1})) \tag{2}$$

V. COMPUTING EVERY FUNCTION

Theorem 6 (Universality of NAND) *There exists some constant $c > 0$ such that for every $n, m > 0$ and function $f : \{0,1\}^n \to \{0,1\}^m$, there is a NAND circuit with at most $c \cdot m2^n$ gates that computes the function f.*