CS 278 - HW2

Joshua Turcotti

February 27, 2021

1 Universal Circuits

(a) First, consider the boolean functions on 1 bit. There are exactly 4 of them, described by the formulas $0, 1, x, \bar{x}$, where we denote the negation of the bit x by $\bar{x}$. To be exact, we will express the former two formulas in terms of x as $x \wedge \bar{x}$ and $x \vee \bar{x}$, respectively, establishing that all boolean functions on 1 bit can be computed with a maximum of 2 gates. Assume that all functions on n bits can be computed with a_n gates, and let $f : \{0,1\}^{n+1} \to \{0,1\}$ be an arbitrary function on $n+1$ bits. We note that $f(x_1, \ldots, x_n, 0)$ and $f(x_1, \ldots, x_n, 1)$ are both boolean functions on n bits, and thus can be computed with a_n gates each. We note that $f(x, \ldots, x_{n+1}) = ((x_{n+1} \wedge f(x_1, \ldots, x_n, 1)) \vee ((\overline{x_{n+1}} \wedge f(x_1, \ldots, x_n, 0))$, which establishes that f can be computed with $2a_n + 4$ gates. This establishes the following asymptotic behavior: any boolean function f on n bits can be computed with a maximum of $a_n = O(2^n)$ gates.

(b) A boolean function on k bits sends each of 2^k possible inputs to one of two outputs, and thus there are 2^{2^k} such functions. In the case of $k = \log_2(n/2)$, there are $2^{n/2}$ such functions. As seen above, each can be computed with $O(2^k) = O(n/2)$ gates, so to compute all functions on k bits requires $O(n2^{n/2})$ gates.

(c) We wish to compute $f_y(x_1, \ldots, x_\ell)$ for all $y \in \{0,1\}^{n-\ell}$. For any given y, we assume access to circuits computing $f_{(1,y)}$ and $f_{(0,y)}$ on all inputs, and so we use the relation $f_y(x_1 \ldots, x_\ell) = ((x_\ell \wedge f_{(1,y)}(x_1, \ldots, x_{\ell-1})) \wedge (\overline{x_\ell} \wedge f_{(0,y)}(x_1, \ldots, x_{\ell-1}))$ to compute f_y on all inputs with 4 additional gates. Since there are $2^{n-\ell}$ choices of y, it takes $O(4 \cdot 2^{n-\ell}) = O(2^{n-\ell})$ additional gates to compute f_y on all inputs.

(d) We wish to compute $f(x_1, \ldots, x_n)$. We begin with the circuit from part b, which can compute all possible functions on the first k bits of x using $O(n2^{n/2})$ gates. Then we apply the inductive step from part c to reason that $f_y(x_1, \ldots, x_{k+1})$ can be computed for all $y \in \{0,1\}^{n-(k+1)}$ using $O(2^{n-(k+1)})$ additional gates. Continuing the induction until we can compute $f_\emptyset(x_1, \ldots, x_n)$ we see that we will require $O(n2^{n/2} + 2^{n-(k+1)} +$

$2^{n-(k+2)}+\ldots+2^0) = O(n2^{n/2}+2^{n-k}) = O(n2^{n/2}+2^n/(n/2)) = O(2^n/n)$ gates.

2 Equivalence Modulo the Modulo of the Modulo Gates

(a) First, we wish to implement MOD_2 and MOD_3 using MOD_6. For the former, simply take each input wire and feed 3 copies of it into a MOD_6 gate, which will then output 0 iff the original count of 1 input wires was divisible by 2. Similarly, for the latter, take each input wire and feed 2 copies of it into a MOD_6 gates, which will then output 0 iff the original count of 1 inputs wires was divisble by 3.

(b) Conversely, we wish to implement MOD_6 using MOD_2 and MOD_3. To do this, we feed each wire into both the MOD_2 and the MOD_3 gate, and output the $\wedge$ of the outputs from the two respective MOD gates. This final output will be 0 iff the original count of 1 wires was divisible by 6.

(c) Pad the input to the MOD_m gate with 0 inputs so that, WLOG, we may assume that $n = m2^d$ for some positive integer d. Even with this padding, $d = O(\log n)$ in the original n. We note that given m bits, it is possible to compute $MOD_{m,k}(x_1, \ldots, x_m)$ for all $k \in [m]$ with a constant depth. Given $MOD_{m,k}(x_1, \ldots, x_{m2^i})$ and $MOD_{m,k}(x_{m2^i+1}, \ldots, x_{m2^{i+1}})$ for all $k \in [m]$, it is possible to compute

$$MOD_{m,k}(x_1, \ldots, x_{m2^{i+1}}) = \bigvee_{\ell=0}^{m-1} \left(MOD_{m,\ell}(x_1, \ldots, x_{m2^i}) \wedge MOD_{m,(k-\ell)}(x_{m2^i+1}, \ldots, x_{m2^{i+1}})\right)$$

This step shows that $MOD_{m,k}$ can be computed for any k on inputs of length $m2^{i+1}$ with an $O(1)$ increase in depth given circuits computing $MOD_{m,k}$ for all k on inputs of length $m2^i$. Inductively, we can see that it is possible to compute $MOD_{m,k}$ on inputs of length $m2^i$ with depth $O(i)$, and thus $MOD_m = \overline{MOD_{m,0}}$ in general can be computed on inputs of length n by a circuit with depth $O(\log n)$.

(d) To show this we must show that MOD_2 gates can be built with polynomially many $AC^0[4]$ gates, and that MOD_4 gates can be built with polynomially many $AC^0[2]$ gates. The former is trivial, as we may perform our trick from part a of sending 2 copies of each input wire to a MOD_4 gate to simulate perfectly a MOD_2 gate. For the latter, we must observe Lucas's theorem, which states that

$$\binom{m}{n} \cong \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod 2$$

where m_i and n_i are the digits in the binary expansion of m and n respectively. Specifically, we will choose $n = m-2$ here such that $\binom{m}{n} = \frac{m(m-1)}{2}$, and thus m is divisible by 4 iff $\binom{m}{n}$ is even (implying one of m or $m-1$ is divisible by 4, whichever is the unique even member of the pair) and m is even (implying m is that member). Since $\binom{m_i}{n_i}$ takes on the value 1 unless $(m_i, n_i) = (0, 1)$, in which case it takes on the value 0, we can conclude that for input m of length k bits:

$$MOD_4(m) = MOD_2(m) \vee \bigwedge_{i=1}^{k} (m_i \vee \overline{n_i})$$

It suffices to show that $n = m-2$ can be computed with polynomially many gates. We do this by first representing -2 as the k-digit 2's complement number $t' = 11\ldots10$, and then performing the binary addition algorithm which uses linearly many gates. This concludes our demonstration that MOD_4 can be implement with polynomially many gates from $AC^0[2]$, proving that $AC^0[2] = AC^0[4]$

3 Sides of Error

(a) Since $0 \leq 1/3$, it is clear that the definition of **RP** is a strict strengthening of the definition of **BPP**, so **RP** $\subseteq$ **BPP** follows. We must now show that **ZPP** $\subseteq$ **RP**. Assume $L \in$ **ZPP**. It suffices to show $L \in$ **RP**. By assumption, we have a probabilistic Turing machine M and a polynomial p satisfying the definition of **ZPP**. Construct the poly-time probabilistic Turing machine M' that, on input x, simulates M for $3p(|x|)$ steps and returns its result if M halts, otherwise 0. It is clear that if $x \notin$, M will either halt with output 0, and thus M' will output 0, or M will not halt, and M' will output 0. This satisfies the second half of the **RP** definition. Also note that since the expectation of M's running time is $p(|x|)$ the probability that its running time exceeds $3p(|x|)$ is at most $1/3$, so with probability at least $2/3$ M will halt and M' will output 1 if M outputted 1, which happens with probability 1 conditioned on $x \in L$. This establishes that the probability M' outputs 1 on $x \in L$ is at least $2/3$, proving $L \in$ **RP** and thus **ZPP** $\subseteq$ **RP**.

(b) We wish to show that **RP** $\subseteq$ **NP**. Assume $L \in$ **RP**. It suffices to show $L \in$ **NP**. By assumption we have a deterministic turing machine $M(x, r)$ that runs in time polynomial in $|x|$. Additionally, we know that the probability that $M(x, r) = 1$ over random choice of $r \in \{0, 1\}^{p(|x|)}$ for polynomial p and $x \in L$ that $M(x, r)$ is at least $2/3$. Rephrased, for all $x \in L$ there exists $r \in \{0, 1\}^{p(|x|)}$ such that $M(x, r) = 1$ for a deterministic poly-time Turing machine M. This is exactly the definition of **NP**, and thus $L \in$ **NP**, concluding our proof.

(c) We wish to show that **ZPP** = **RP** $\cap$ **coRP**. Our argument from part a allows us to conclude both **ZPP** $\subseteq$ **RP** and **ZPP** $\subseteq$ **coRP** (the latter by outputting 1 in the case of non-halting instead of 0). Thus it suffices to show, for arbitrary $L \in$ **RP** $\cap$ **coRP**, $L \in$ **ZPP**. By assumption on L let M be the poly-time probabilistic TM with no false positives (establishing $L \in$ **RP**) and M' the poly-time probabilistic TM with no false negatives (establishing $L \in$ **coRP**). Let M'' be the machine that, on input x, takes turns simulating M and M' on x until either M halts with output 1 or M' halts with output 0, outputting the output of the machine that halted last. Since this output will be 1 iff $x \in L$, it suffices to show that, on expectation, M'' will perform a constant number of simulations of other machines, in which case its own expected runtime will be polynomial. The expected number of simulations M'' performs is just the series $E = 2/3 + 2 * 2/9 + 3 * 2/18 + \ldots$. Noting $E - (1/3)E$ is just the geometric series $2/3 + 2/9 + 2/18 + \ldots = 1$, we can conclude $E = 3/2 = O(1)$. This establishes $L \in$ **ZPP**, and thus **ZPP** = **RP** $\cap$ **coRP**.