CS 505 Spring 2025 — Homework 4 (Sample Solutions)

Alexander R. Block

Due Date: April 15, 2025, no later than 2:00pm Central Time.

1 Boolean Circuits (25 Points)

1.1 Part 1 (5 Points)

Let $\mathrm{add}_n \colon \{0,1\}^{2n} \to \{0,1\}^{n+1}$ be the function $\mathrm{add}(x,y) = x + y$ for two n-bit integers x and y. Show that add_n is computable by an $O(n)$-sized circuit family. Here, the circuit has multiple outputs instead of $\{0,1\}$.

Proof of Problem 1 Part 1. We construct a Boolean circuit family for add_n. This family will simply perform the grade-school addition algorithm over binary. For example, if we add the bit strings 1101 and 0111, we would obtain 10100. For $n = 1$, the circuit is easy, as the addition of two bits x, y results in a string $z_2 z_1$ with the following properties: $z_1 = x \oplus y$ and $z_2 = x \wedge y$. Notably, $x \oplus y = (x \wedge \overline{y}) \vee (\overline{x} \wedge y)$. So here, the circuit implementing this functionality has size 4.

For general n, the circuit operates as follows. Let $x = x_n x_{n-1} \cdots x_1$ and $y = y_n y_{n-1} \cdots y_1$. We will have two gadget circuits: the XOR gadget and the CARRY gadget. Here, $\mathrm{XOR} \colon \{0,1\} \times \{0,1\} \to \{0,1\}$ is simply defined as $\mathrm{XOR}(a,b) = a \oplus b$. We will define CARRY later. Recalling how the grade-school algorithm for addition works, consider $z = z_{n+1} z_n \cdots z_1$ and, in particular, how we compute z_1. We have that $z_1 = x_1 \oplus y_1 = \mathrm{XOR}(x_1, y_1)$. Now, to compute z_2, we know that $z_2 = x_2 \oplus y_2 \oplus c_1$, where c_1 is the carry bit from $x_1 \oplus y_1$. In particular, if $x_1 = y_1 = 1$, then $c_1 = 1$. In this case, $\mathrm{CARRY}(x_1, y_1) = x_1 \wedge y_1$. Given this, $z_2 = x_2 \oplus y_2 \oplus c_1 = \mathrm{XOR}(\mathrm{XOR}(x_2, y_2), \mathrm{CARRY}(x_1, y_1))$.

Now, how do we compute z_3? As with z_2, we have $z_3 = x_3 \oplus y_3 \oplus c_2$, where c_2 is the carry bit from computing z_2. However, in this case, it is not enough to set $c_2 = x_2 \wedge y_2$, as computing z_2 also used the carry bit c_1. In this case (and for all other c_i for $i \geq 2$), the carry bit is 1 if at least two of x_2, y_2, c_1 (and, in general, x_i, y_i, c_{i-1}) are 1. Encoding thing as a boolean formula, we have $c_2 = (x_2 \wedge y_2) \vee (x_2 \wedge c_1) \vee (y_2 \wedge c_1)$. Thus, we can define $\mathrm{CARRY} \colon \{0,1\}^3 \to \{0,1\}$ as $\mathrm{CARRY}(a,b,c) = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$.

With this in mind, we have that $z_3 = \mathrm{XOR}(\mathrm{XOR}(x_3, y_3), \mathrm{CARRY}(x_2, y_2, c_1))$, for $c_1 = \mathrm{CARRY}(x_1, y_1, c_0)$ and $c_0 := 0$ (i.e., there is no carry bit to start with). In general, computing z_i is done as

$$z_i = \mathrm{XOR}(\mathrm{XOR}(x_i, y_i), c_{i-1}),$$

where $c_{i-1} = \mathrm{CARRY}(x_{i-1}, y_{i-1}, c_{i-2})$. Moreover, for $n+1$, we set $x_{n+1} = y_{n+1} = 0$ so that $z_{n+1} = c_n$ and $c_{n+1} = 0$. Overall, the Boolean expressions for XOR and CARRY are of constant size, each z_i is computed by 2 XOR computations and one CARRY computation. Therefore, the circuit implementing add_n has size $O(n)$ as required. □

1.2 Part 2 (10 Points)

A language $L \subseteq \{0,1\}^*$ is sparse if there exists a polynomial p such that for every $n \in \mathbb{N}$, $|L \cap \{0,1\}^n| \leq p(n)$. Show that every sparse language is in $\mathbf{P}_{/\mathbf{poly}}$.

Proof of Problem 1 Part 2. Let L be any sparse language with polynomial p. Let $S_n = L \cap \{0,1\}^n$ for any $n \in \mathbb{N}$. To show that $L \in \mathbf{P}_{/\mathbf{poly}}$, it suffices to show that there exists a circuit C_n which decides S_n such that $|C_n| = \mathrm{poly}(n)$.

By definition of a sparse language, we know that $|S_n| \leq p(n)$ for every n. Suppose that $|S_n| = k$ and that $S_n = \{y_1, \ldots, y_k\}$ for $y_i \in \{0,1\}^n$ for all $i \in [k]$. Our circuit C_n will simply have $y_1, \ldots, y_k$ hard-coded and will compare the input to each of them. In particular,

$$C_n(x) := \bigvee_{i=1}^{k} (x = y_i),$$

where $=$ is the bit-wise equality operator. Note that $=: \{0,1\}^{2n} \to \{0,1\}$ is computable by a size $O(n)$ circuit, so $|C_n| = O(k \cdot n) = O(p(n) \cdot n) = \text{poly}(n)$. Thus, $L \in \mathbf{P_{/poly}}$. $\square$

1.3 Part 3 (10 Points)

Prove that a language L is decidable by a family of logspace uniform circuits if and only if $L \in \mathbf{P}$.

Proof of Problem 1 Part 3. First, suppose that L is decidable by a family of logspace uniform circuits. That is, there is a logspace deterministic Turing machine such that for any x, on input 1^n for $n = |x|$ outputs C_n such that $C_n(x) = L(x)$. Clealry, $L \in \mathbf{P}$ since a DTM without space restrictions can simply simulate the above DTM (which runs in polynomial time because it only uses logarithmic space), write down the circuit C_n to its work tape, then evaluate $C_n(x)$ and output the result.

Now suppose that $L \in \mathbf{P}$. Then, the transformation from class from any DTM M_L deciding L to an equivalent poly-sized circuit C_n deciding $L(x)$ for $n = |x|$ and any x is exactly a logspace uniform transformation. So we have that L is decidable by a logspace uniform circuit family. $\square$

2 Interactive Proofs (25 points)

2.1 Part 1 (5 Points)

Show that $\mathbf{AM}[2] = \mathbf{BP} \cdot \mathbf{NP}$.

Proof of Problem 2 Part 1. First, recall that $\mathbf{BP} \cdot \mathbf{NP} = \{L \colon L \leq_r 3\text{SAT}\}$; that is, all languages L which are randomized reducible to 3SAT. This means that for every L, there exists a polynomial-time PTM M_L such that for all x, we have $\Pr[3\text{SAT}(M_L(x)) = L(x)] \geq 2/3$. Equivalently stated, we have that

$$\Pr_r[M_L(x,r) \in 3\text{SAT} \mid x \in L] \geq 2/3 \qquad \Pr_r[M_L(x,r) \notin 3\text{SAT} \mid x \notin L] \geq 2/3,$$

where M_L is not a DTM and $r \in \{0,1\}^{\text{poly}(n)}$ is uniformly sampled.

Recall also that $\mathbf{AM} = \mathbf{AM}[2]$ is the set of all languages L that have public-coin interactive proofs with the following structure:

1. V sends the first message c which is uniformly sampled from $\{0,1\}^{\text{poly}(n)}$, where $n = |x|$ for common input x between P and V.
2. P sends the second message $m \in \{0,1\}^{\text{poly}(n)}$.
3. The output of the protocol is determined as $b = V(x, c, m)$.
4. We have

$$\Pr_c[V(x,c,P(x,c)) = 1 \mid x \in L] \geq 2/3 \qquad \Pr_c[V(x,c,P^*(x,c) = 1 \mid x \notin L] \leq 1/3,$$

where the second probability holds for any cheating prover P^*.

First, assume that $L \in \mathbf{BP} \cdot \mathbf{NP}$. Then, by definition, there exists a DTM M_L such that

$$\Pr_r[M_L(x,r) \in 3\text{SAT} \mid x \in L] \geq 2/3 \qquad \Pr_r[M_L(x,r) \notin 3\text{SAT} \mid x \notin L] \geq 2/3.$$

We design an AM protocol for L. The protocol operates as follows.

1. V samples $r \in \{0,1\}^{\text{poly}(n)}$ and sends it to P.
2. P computes $\phi = M_L(x,r)$, computes a satisfying assignment z for ϕ, and sends z to V.
3. V computes $\phi = M_L(x,r)$ and checks if $\phi(z) = 1$, outputting accept if and only if this is true.

By definition of $L \in \mathbf{BP} \cdot \mathbf{NP}$, clearly the above protocol has completeness error and soundness error $\leq 1/3$, and the verifier runs in strict polynomial time. Thus, $L \in \mathbf{AM}$.

Now, assume that $L \in \mathbf{AM}$. Then, there is a proof system (P,V) for L with completeness and soundness error $\leq 1/3$, where for any x, V samples $c \in \{0,1\}^{\text{poly}(n)}$ and sends c to P, P replies with some message $m = g(x,c)$, and V computes some function $f(x,c,m) \in \{0,1\}$, where f is poly-time computable. Notice that f is computable by a polynomial-time DTM when c is fixed; let M_f be this Turing machine. By the Cook-Levin Theorem, we can convert the computation of M_f into a polynomial-sized 3SAT instance, and this reduction is polynomial-time. Moreover, since the protocol has completeness error $\leq 1/3$, it implies that for any $x \in L$, at least $2/3$ of random strings c satisfy $f(x,c,g(x,c)) = 1$. This implies that for $x \in L$, the 3SAT formula encoding $M_f(x,c)$ will be satisfiable for at least $2/3$ of the strings c; similarly, if $x \notin L$, then the 3SAT formula encoding $M_f(x,c)$ will be unsatisfiable for at least $2/3$ of the strings c. This shows that $L \leq_r$ 3SAT, as required. □

2.2 Part 2 (10 Points)

The *graph isomorphism* problem defined as follows. Let $G_0 = (V_0, E_0), G_1 = (V_1, E_1)$ be two graphs, each on n vertices. We say that G_0 and G_1 are isomorphic if there exists a permutation $\pi \colon [n] \to [n]$ such that $\pi(G_0) = G_1$. That is, after relabeling the vertices of G_0 using the permutation π, we obtain the graph G_1. Another way to state this: if we permute the rows or columns of the adjacency matrix of G_0 according to π, we obtain the adjacency matrix for G_1.

Give an interactive proof for deciding if two graphs G_0, G_1 are isomorphic. In particular, answer the following questions.

1. What is the completeness error?
2. What is the soundness error?
3. How many rounds does the IP have?
4. If your soundness error bound is δ, how can you reduce it to δ^k?

Note: *I encourage you to try to solve this problem without consulting online sources first!*

Proof of Problem 2 Part 2. We give a simple interactive proof for the graph isomorphism problem. Let (G_0, G_1) be two n vertex graphs, which are the public inputs to the interactive proof. The algorithms (P,V) on input (G_0, G_1) operate as follows.

1. The prover samples a random permutation σ and sends $H = \sigma(G_0)$ to V.
2. V samples bit $\widetilde{b}$ and sends it to P.
3. The prover sends a permutation π such that $H = \pi(G_{\widetilde{b}})$.
4. V checks that π is a valid permutation and if $H = \pi(G_{\widetilde{b}})$ and accepts if and only if this is true.

First, if $G_0 \cong G_1$, the prover always convinces the verifier in the above interactive proof. In particular, if $\widetilde{b} = 0$, then the prover simply sends $\pi = \sigma$ and the verifier accepts, and if $\widetilde{b} = 1$, then $\pi = \sigma \circ \rho$, where $G_1 = \rho(G_0)$ is the isomorphism between G_0 and G_1 (and again the verifier accepts).

For soundness, suppose that $G_0 \cong G_1$. Note that any cheating prover P^* must first send some graph H^* and some permutation π^*. The verifier only accepts if $H^* = \pi^*(G_{\widetilde{b}})$. So P^* must send H^* that is isomorphic to one of G_0 and G_1. Without loss of generality, assume P^* sends $H^* = \sigma(G_0)$. Then, P^* only succeeds in convincing the verifier if $\widetilde{b} = 0$, which happens with probability $1/2$. This is because since $G_0 \ncong G_1$, then there is no permutation π such that $H = \pi(G_1)$ (since $H \cong G_0$).

Therefore, the above interactive proof has perfect completeness, soundness error $1/2$, and has 3 messages. We can repeat this IP k times in parallel to keep it at 3 messages and reduce the soundness error to 2^{-k}.

On another note, this IP is actually a zero-knowledge proof! The verifier learns nothing about the isomorphism between G_0 and G_1, as it either gets a random permutation of G_0, or a permutation between H and G_1 (and not a permutation between G_0 and G_1). $\square$

2.3 Part 3 (10 Points)

Let $G = (V, E)$ be an undirected simple graph (i.e., no self loops) on n vertices. A *triangle* is a tuple of vertices $(i, j, k) \in V \times V \times V$ such that $(i, j), (j, k), (k, i) \in E$. Let

$$\mathrm{TRI} = \{(G, k) \colon G \text{ is a simple graph with } k \text{ triangles.}\}.$$

Show that $\mathrm{TRI} \in \mathbf{IP}$. Here, use the "standard" definition of **IP** presented in class (i.e., completeness and soundness error $\leq 1/3$).

Hint: consider the adjacency matrix view of G; let A be its adjacency matrix. View the matrix A as a Boolean function $f_A \colon \{0,1\}^{\log(n)} \times \{0,1\}^{\log(n)} \to \{0,1\}$, where $f_A(i,j) = 1$ if and only if $A[i,j] = 1$. How can you use f_A to count the number of triangles in a graph G with adjacency matrix A? Once you can do this, consider how to use the sum-check protocol in your interactive proof.

Proof of Problem 2 Part 3. Consider the adjacency matrix A of the graph G and consider the Boolean function f_A as described above. Then, we can use f_A to count triangles in G. Let T be the number of triangles in G. Then, we claim that

$$T = \frac{1}{6} \sum_{1 \leq i,j,k \leq n} f_A(i,j) \cdot f_A(j,k) \cdot f_A(k,i),$$

where we compute the sum over the integers. Clearly, if $(i, j), (j, k), (k, i) \in E(G)$, then these form a triangle. Moreover, since G is an undirected graph, the ordering of these vertices do not matter, and there are $3! = 6$ permutations of the vertices (i, j, k) which admit a triangle. So when we sum over $f_A(i,j) \cdot f_A(j,k) \cdot f_A(k,i)$, we are over counting by a factor of 6.

Now, with a few modifications, we can simply invoke the sum-check protocol over a polynomial F, which we describe shortly. First, pick p to be a sufficiently large prime number such that $p \gg \binom{n}{3}$ (i.e., the maximum number of triangles, e.g., if G was fully connected). Now, define the polynomial $F(\mathbf{X}, \mathbf{Y}, \mathbf{Z}) = \widetilde{f}_A(\mathbf{X}, \mathbf{Y}) \cdot \widetilde{f}_A(\mathbf{Y}, \mathbf{Z}) \cdot \widetilde{f}_A(\mathbf{Z}, \mathbf{X})$, where $\widetilde{f}_A$ denotes the multilinear extension of f_A over $\mathbb{Z}_p$. That is, for all $x, y \in \{0,1\}^{\log(n)}$, we have $\widetilde{f}_A(x, y) = f(x, y)$, and $\widetilde{f}_A \colon \mathbb{Z}_p^{\log(n)} \times \mathbb{Z}_p^{\log(n)} \to \mathbb{Z}_p$ is a multilinear polynomial.

This implies that F is a polynomial on $3\log(n)$ variables and has individual degree at most 3. Moreover, clearly we have that $T = \sum_{i,j,k \in \{0,1\}^{\log(n)}} F(i,j,k)$. This tells us that a simple interactive proof for counting triangles is simply the sum-check protocol over the polynomial F. This sum-check will last for $3\log(n)$ rounds, has perfect completeness and soundness error $\leq \frac{3 \cdot 3 \log(n)}{p}$ since the individual degree of F is at most 3 in every variable. Moreover, clearly this is a valid instantiation of the sum-check protocol because the verifier can evaluate F at a random point in $\mathbb{Z}_p$ in polynomial time, since it is an evaluation of $\widetilde{f}_A$ a constant number of times, and $\widetilde{f}_A$ can be evaluated at a random point by the verifier in polynomial time. $\square$