Chapter 6 : Parallel computation : Solutions of the exercises

1. $x \oplus y = (x \vee y) \wedge \overline{(x \wedge y)}$. In order to show that three gates do not suffice, let S_k denote the set of all functions on input x, y, that can be realized using exactly k basic gates. For small values of k we can exhaustively enumerate all elements of S_k. In particular, one can easily check that:

$$\begin{aligned} S_0 &= \{x, y\}, \\ S_1 &= \{x, y, \overline{x}, \overline{y}, x \vee y, x \wedge y\}, \\ S_2 &= \{0, 1, x, y, \overline{x}, \overline{y}, x \vee y, x \vee \overline{y}, \overline{x} \vee y, \overline{x} \vee \overline{y}, x \wedge y, x \wedge \overline{y}, \overline{x} \wedge y, \overline{x} \wedge \overline{y}\}. \end{aligned}$$

Suppose that $x \oplus y$ can be obtained using three basic gates. Call the output gate G. If G is a NOT gate, then G is fed an input which realizes $\overline{x \oplus y}$ using two gates only. But the list S_2 does not include this function (XNOR). So assume now that G is an OR gate. Looking at the truth table of $x \oplus y$ and of the functions in $S_1 \cup S_2$ indicates that the two inputs to G must in this case be $\overline{x} \wedge y$ and $x \wedge \overline{y}$. But we know that these functions are not in S_1 and it is impossible to realize both using two gates only. Finally, if G is an AND gate, its two inputs must be $x \vee y$ and $\overline{x \wedge y} = \overline{x} \vee \overline{y}$. Again it is impossible to realize both the functions using two gates only. Thus all possibilities for G lead to contradictions.

A similar argument shows that $\overline{x \oplus y}$ cannot be realized using three basic gates. Moreover, $\overline{x \oplus y} = (x \wedge y) \vee \overline{(x \vee y)}$ is realizable using only four gates. It follows that:

$$\begin{aligned} S_3 &= \{0, 1, x, y, \overline{x}, \overline{y}, x \vee y, x \vee \overline{y}, \overline{x} \vee y, \overline{x} \vee \overline{y}, x \wedge y, x \wedge \overline{y}, \overline{x} \wedge y, \overline{x} \wedge \overline{y}\}, \\ S_4 &= \{0, 1, x, y, \overline{x}, \overline{y}, x \vee y, x \vee \overline{y}, \overline{x} \vee y, \overline{x} \vee \overline{y}, x \wedge y, x \wedge \overline{y}, \overline{x} \wedge y, \overline{x} \wedge \overline{y}, x \oplus y, \overline{x \oplus y}\}. \end{aligned}$$

2. It suffices to show that $\mathrm{NC}^{j+1} = \mathrm{NC}^j$ implies $\mathrm{NC}^{j+2} \subseteq \mathrm{NC}^{j+1}$. Take $L \in \mathrm{NC}^{j+2}$ and let $(C_0, C_1, C_2, \ldots)$ be a uniform family of circuits realizing L. Then C_n has size $\leqslant p(n)$ and depth $\mathrm{O}(\log^{j+2} n)$ for each n, where $p(n)$ is a fixed polynomial in n. We order the gates in C_n as $g_1, g_2, \ldots, g_{p(n)}$ such that each g_k takes input from (some of) the previous gates. Now break the list into $\mathrm{O}(\log n)$ levels each containing exactly $\log^{j+1} n$ gates. We view each level as a circuit that receives its input from the previous levels. Consider the i-th gate g at the k-th level. The output of g defines a Boolean function on at most $p(n)$ input variables (outputs of gates from the previous levels). Pad the inputs of the circuit for g, so that it has exactly $p(n)$ input variables. Call this circuit $C^{(i)}_{p(n)}$. This has size polynomial in n (and hence in $p(n)$) and depth $\leqslant \log^{j+1} n = \mathrm{O}(\log^{j+1} p(n))$. But then $\left(C^{(i)}_{p(n)}\right)$ is a uniform family of circuits deciding a language $L^{(i)} \in \mathrm{NC}^{j+1}$. Since $\mathrm{NC}^{j+1} = \mathrm{NC}^j$, $L^{(i)}$ is realized also by a circuit family $\left(D^{(i)}_{p(n)}\right)$, where $D^{(i)}_{p(n)}$ has size polynomial in $p(n)$ (and hence n) and depth $\mathrm{O}(\log^j p(n))$, i.e., $\mathrm{O}(\log^j n)$. So we can replace the circuits $C^{(i)}_{p(n)}$ in the current (k-th) level by the respective circuits $D^{(i)}_{p(n)}$ working in parallel.

Doing this for every level (i.e., for every value of k) converts C_n to a circuit of polynomial size and of depth $\mathrm{O}(\log^{j+1} n)$. We conclude that $L \in \mathrm{NC}^{j+1}$.

3. Let $\langle C, \alpha \rangle$ be an input for CIRCUIT-VALUE with $|\alpha| = n$. We want to convert it to an instance $\langle C', \alpha' \rangle$ such that C' is monotone and $C'(\alpha') = 1$ if and only if $C(\alpha) = 1$. We take $\alpha' = \alpha\overline{\alpha}01$, where $\overline{\alpha}$ is the bit-wise complement of α. We order the gates of C as $g_1, g_2, \ldots, g_k$ such that each g_i takes input(s) from the outputs of previous gates and from the input variables. Let z_i be the output of gate g_i. C' consists of gates that compute both z_i and $\overline{z_i}$. By induction on i I show that this is achievable without using NOT gates. The induction basis is provided by the availability of all the input variables in non-complemented and complemented forms.

Take some $i \geqslant 1$ and let x (and y) be the input(s) to the gate g_i. By induction $x, \overline{x}, y, \overline{y}$ are available in the part of the circuit C' generated so far. If g_i is an OR gate, then $z_i = x \vee y$ and $\overline{z_i} = \overline{x} \wedge \overline{y}$. If g_i is an AND gate, then $z_i = x \wedge y$ and $\overline{z_i} = \overline{x} \vee \overline{y}$. Finally, if g_i is a NOT gate, we have $z_i = \overline{x} \wedge 1$ and $\overline{z_i} = x \vee 0$. (Recall that 0 and 1 are available from the last two bits of the input α'.) In all the cases AND and OR gates compute both z_i and $\overline{z_i}$ from the available values.

Thus C' has twice as many gates as C has. Also $|\alpha'| = 2|\alpha| + 2$. Moreover, a log-space transducer can be employed to do the conversion of $\langle C, \alpha \rangle$ to $\langle C', \alpha' \rangle$. It follows that CIRCUIT-VALUE $\leqslant_{\mathrm{L}}$ MONOTONE-CIRCUIT-VALUE.

4. **(a)** For proving the inclusion $\mathrm{AC}^j \subseteq \mathrm{NC}^{j+1}$ we use the fact that a k-input AND or OR gate can be replaced by a height-balanced binary tree comprising only two-input gates. The depth of this tree is $\mathrm{O}(\log k)$. The inclusion $\mathrm{NC}^{j+1} \subseteq \mathrm{AC}^{j+1}$ is obvious.

(b) $\mathrm{NC} = \bigcup_{j \in \mathbb{N}} \mathrm{NC}^j \subseteq \bigcup_{j \in \mathbb{N}} \mathrm{AC}^j = \mathrm{AC}$. Conversely, $\mathrm{AC} = \bigcup_{j \in \mathbb{N}_0} \mathrm{AC}^j \subseteq \bigcup_{j \in \mathbb{N}_0} \mathrm{NC}^{j+1} = \bigcup_{j \in \mathbb{N}} \mathrm{NC}^j = \mathrm{NC}$.

(c) Use a construction as in Exercise 2.

5. Let c_i denote the input carry at the i-th position. (Initially, $c_0 = 0$.) Note that $c_i = 1$ if and only if it is generated at some position $j \leqslant i$ and propagated through positions $j+1, \ldots, i-1$. The condition for generation of the carry ($c_{j+1} = 1$) is that $x_j = y_j = 1$, whereas the condition for carry propagation is $x_k = 1$ or $y_k = 1$ for all $k = j+1, \ldots, i-1$, i.e.,

$$c_i = \bigvee_{j=0}^{i-1} \left[(x_j \wedge y_j) \wedge \left(\bigwedge_{k=j+1}^{i-1} (x_k \vee y_k) \right) \right]$$

Thus a depth 3 AC^0 circuit can compute all the carry bits $c_0, c_1, \ldots, c_n$ in parallel. The output bits are then obtained in parallel using bit adder circuits (which are of constant depth): $z_i = x_i \oplus y_i \oplus c_i$ and $z_n = c_n$.