$\mathbf{NC}^1$, Branching Programs, **PSPACE**, and Bottleneck Machines

Advanced Complexity, Homework Assignment

There are several parts in this homework assignment. They are *not* independent.

1 The Class $\mathbf{NC}^1$

The class $\mathbf{NC}^1$ is a curious class, defined in the same way as (uniform) $\mathbf{P/poly}$: it is the class of languages L that are decided by a family of circuits $\mathcal{C}_n$, $n \in \mathbb{N}$:

- whose depth is an $O(\log n)$—depth is the largest number of gates traversed from input to output;
- whose size is polynomial in n;
- that have *bounded fan-in*: each gate ($\wedge$, $\vee$, $\overline{\wedge}$, $\overline{\vee}$-gate) takes at most two inputs—the circuits we were considering in the lectures had unbounded fan-in, as each gate could have arbitrarily many inputs;
- and that are *uniform*, in the sense that there is a logspace Turing machine $\mathcal{M}$ that, on input x (of size n), computes $\mathcal{C}_n$.

1. In the above definition, why is the second requirement (that $\mathcal{C}_n$ be of polynomial size) redundant?
2. Show that $\mathbf{NC}^1 \subseteq \mathbf{L}$. Hint: once you realize the naive algorithm you have written first takes more than logarithmic space, realize that given any logarithmic depth circuit $\mathcal{C}_n$, with output wire number q, then one can use logarithmic length bit strings to denote wires $i < q$ (instead of the number i itself): the empty string ϵ denotes q itself, while if w denotes a wire i, output of a gate G_i with two inputs i_1, and i_2, then $w0$ denotes i_1 and $w1$ denotes i_2. We shall call such bit strings w *paths*.

   Please use the notations: i_w to mean the wire denoted by path w; and op_w to denote the operator implemented by the fate G_{i_w}.

   Note that paths can be used to implement recursion stacks, as well.

2 Branching Programs

A *length n width k branching program* π (for short, an n, k-BP) is any non-empty finite sequence of *instructions* of the form `if` x_i `then` $R := f(R)$ `else` $R := g(R)$, where $0 \leq i < n$, and f and g are functions from $\{1, \ldots, k\}$ to $\{1, \ldots, k\}$. These are meant to work on a read-only bit string x of length n (or more) and with a unique read-write *register* R, taking its values in $\{1, \ldots, k\}$. The *size* of π is its number of instructions.

We assume the obvious semantics—x_i denotes bit i of input x, and the `if` test computes $f(R)$ if x_i is 1, $g(R)$ if x_i is 0. For example, the $9, 2$-BP:

$$\begin{array}{l}\texttt{if } x_7 \texttt{ then } R := (1\ 2)(R) \texttt{ else } R := R;\\ \texttt{if } x_4 \texttt{ then } R := R \texttt{ else } R := (1\ 2)(R);\\ \texttt{if } x_8 \texttt{ then } R := (1\ 2)(R) \texttt{ else } R := R;\end{array}$$

will test whether exactly one of x_7, $\neg x_4$, and x_8 is true. If so the final value of R will be 1 if started with $R = 2$, and 2 if started with $R = 1$. Otherwise, the final value of R will be the same as when we started the program. (The map (1 2) is the permutation that swaps 1 and 2).

In general, a *permutation* is any bijective map from $\{1, \ldots, k\}$ to $\{1, \ldots, k\}$. A *cycle* on $\{1, \ldots, k\}$ is any permutation of the form $(m_1\ m_2\ \ldots\ m_k)$ (with m_1, m_2, $\ldots$, m_k pairwise distinct and between 1 and k) that sends m_1 to m_2, m_2 to m_3, $\ldots$, m_{k-1} to m_k, and m_k to m_1. E.g., (1 2 3 4 5) is a cycle (this is *not* the identity) when $k = 5$, as well as (1 3 5 4 2) or (5 4 3 2 1).

It is important to note that every cycle $\sigma = (m_1\ m_2\ \ldots\ m_k)$ yields a unique *associate* permutation σ', which maps each $i \in \{1, \ldots, k\}$ to m_i. E.g., the associate of the cycle (1 2 3 4 5) is the identity map (which is not itself a cycle).

An n, k-BP is a *permutation branching program* (an n, k-PBP) iff the functions (f, g, from $\{1, \ldots, k\}$ to $\{1, \ldots, k\}$) used in its instructions are all permutations.

Given an input x of length n, and n, k-BP (resp., n, k-PBP) π defines a map (resp., a permutation) from $\{1, \ldots, k\}$ to $\{1, \ldots, k\}$, which sends the initial value of R to the value it has at the end of the program. Call f_π this map (resp., permutation).

We say that a language L of bit strings of length n (i.e., the length is fixed) is *cycle-recognized* by an n, k-PBP π iff there is a cycle σ over $\{1, \ldots, k\}$ such that:

- For every $x \in L$, $f_\pi = \sigma$;
- For every $x \notin L$ of length n, f_π is the identity map.

If this is so, we say that L is cycle-recognized by π *with output* σ. Notice that this is well-defined, since σ, as a cycle, cannot be the identity. (A cycle has no fixpoint.)

Our first move is to show that which cycle σ we choose in defining cycle-recognition is irrelevant.

3. Let π be an n, k-PBP cycle-recognizing L with output σ, and τ be any cycle over $\{1, \ldots, k\}$. Build another n, k-PBP π', of the same size as π, that cycle-recognizes

L with output τ. Hint: first show that there is a permutation θ such that $\tau = \theta\sigma\theta^{-1}$ (where we write e.g. $\theta\sigma$ for $\theta \circ \sigma$, for short); this can be built using associate permutations.

4. Let L be a language of bit strings of length n, and $\overline{L}$ be its complement inside the set of length n bit strings. Given an n, k-PBP π that cycle-recognizes L, build another one, $\overline{\pi}$, of the same size, but that cycle-recognizes $\overline{L}$.

5. Let $k = 5$, $\sigma_1 = (1\ 2\ 3\ 4\ 5)$, $\sigma_2 = (1\ 3\ 5\ 4\ 2)$. It is easily checked that the commutator $\sigma_1\sigma_2\sigma_1^{-1}\sigma_2^{-1}$ of these two cycles is the cycle $(1\ 3\ 2\ 5\ 4)$.

   Deduce *Barrington's Lemma*: if L_1 is cycle-recognized by an $n, 5$-PBP π_1 of size t_1 and L_2 is cycle-recognized by an $n, 5$-PBP π_2 of size t_2, then $L_1 \cap L_2$ is cycle-recognized by some $n, 5$-PBP of size $2(t_1 + t_2)$.

Until now, we were recognizing languages of fixed length n. Say that a language L (of words of arbitrary length n) is *decided* by a family $(\pi_n)_{n\in\mathbb{N}}$ of n, k-PBPs π_n (of fixed width k, but varying n) if and only if, for every input x, writing n for the size of x, if $x \in L$ then $f_{\pi_n}(x)(1) = 2$ and if $x \notin L$ then $f_{\pi_n}(x)(1) = 1$.

Such a family is *uniform* iff there is a logspace Turing machine that, given any input of size n (i.e., n written in unary), computes π_n.

6. Using the previous questions, show that every language in $\mathbf{NC}^1$ is decided by a uniform family of PBPs of polynomial size, and of width 5.

7. Conclude that $\mathbf{NC}^1$ is exactly the class of languages decided by uniform families of PBPs of polynomial size and width 5. Hint: split PBPs in two, recursively.

3 PSPACE and Bottleneck Machines

8. Show that, given a language L decided by an alternating Turing machine $\mathcal{M}$ that works in polynomial time $p(n)$, one can build a circuit $\mathcal{C}_n$ of polynomial size, with n input bits, such that $\mathcal{C}_n[x]$ evaluates to 1 iff x is in L (for any bit string x of length n).

9. Using the results above, show that every language L in **PSPACE** is decided by a so-called *bottleneck Turing machine* $\mathcal{M}$. Such a machine runs exponentially many phases in succession, phase 0 through $2^{p(n)} - 1$, where p is a fixed polynomial, but requires incredibly low space—and its memory gets almost completely erased regularly, if this were not enough.

   In each phase, the machine starts with access to:

   - the input tape (with the same input x, of size n, for all phases),
   - a $p(n)$ bit read-only counter tape holding the current phase number,
   - a read-write register R, holding a value in $\{1, 2, 3, 4, 5\}$,

- a fixed number of work tapes, which are empty at the beginning of the phase.

Whenever phase i starts, the machine is allowed to do some logspace computation using the above input (space refers to the work tapes), and write back a new value for R. Then the work tapes are entirely erased, the phase number counter is incremented, and phase $i+1$ starts (unless $i = 2^{p(n)} - 1$). We run all phases 0, 1, ..., $2^{p(n)} - 1$, with R starting as 1 (in phase 0). Once the last phase finishes, the machine accepts if $R = 2$, rejects otherwise. (Note the similarities with PBPs.)

10. Conversely, show that any language L decided by a bottleneck Turing machine is in **PSPACE**.

It follows that **PSPACE** is exactly the class of languages that are decided by bottleneck Turing machines. This is the *Cai-Furst Theorem* (1991).