Problem 1: NFA Minimization (30)

High Level Idea

• We will show that \(3CNF \leq_p NFLALL\), where \(NFLALL\) is the language containing all NFAs that accept \(\Sigma^*\). Formally, \(NFLALL = \{N \mid L(N) = \Sigma^*\}\).

• Next, we will show that if we can minimize NFAs in poly time, then \(NFLALL \in P\).

• From this, it follows that \(coNP = P\), which gives \(P = NP\).

Step 1: \(3CNF \leq_p NFLALL\)

• We will define a function \(f : 3CNF \rightarrow NFLA\) such that \(L(f(\phi)) = \Sigma^*\) iff \(\phi\) is unsatisfiable.

• Let \(\phi\) be a \(3CNF\) with \(m\) clauses and \(n\) variables, where
  - \(\phi = c_1 \land c_2 \land ... \land c_m\)
  - \(\forall i : c_i = l_{i,1} \lor l_{i,2} \lor l_{i,3}\)
  - forall \(i,j\): \(l_{i,j}\) is either \(x_k\) or \(\overline{x_k}\).

• For each \(c_i\), we can construct \(N_i\), where:
  - \(L(N_i) =\) set of all strings that make \(c_i\) evaluate to false. Formally, \(L_i = \{s \mid |s| = n, l_{i,1} = l_{i,2} = l_{i,3} = 0\}\)
  - We can construct \(N_i\) using \((n + 2)\) gates by not caring about the bits that \(l_{i,*}\) do not touch, and forcing the bits that \(l_{i,*}\) does touch.

• Thus, \(N = (\cup N_i)\) accepts all strings \(x\) such that \(\phi(x) = false\) and \(|x| = n\).

• Define \(O = \cup_{i=0}^{n-1} \sum_i\) (to be a NFA accepting all strings of length < \(n\)), and \(P\) to be a NFA accepting all strings of length > \(n\).

• Note that \(N\), \(O\), and \(P\) all have size polynomial in terms of \(n\).

• Finally, output the NFA \(Q = N \cup O \cup P\).

• \(L(Q) = \Sigma^* \iff \phi\) is unsatisfiable.

Step 2: If we can minimize NFAs in poly time, then \(NFLALL \in P\).

Given a NFA \(N\), we run \(\text{minimize}(N)\), and check if the output is a single state NFA where:

• start state = final state

• loops back on itself on 0,1

Step 3: Thus \(P = NP\)

Combining steps 1 and 2, we get \(coNP = P\), from which it follows that \(P = NP\), since \(\overline{P} = P\) and \(\overline{coNP} = NP\).
Problem 2: ExactClique is both NP-hard and coNP-hard (30)

Main Idea:
- We will define $f$, a reduction from 3CNF to Clique.
- If $\phi \in 3CNF$ is satisfiable, the largest clique in $f(\phi)$ will have size exactly $m$.
- If $\phi \in 3CNF$ is unsatisfiable, the largest clique in $f(\phi)$ will have size exactly $m - 1$.

Reduction:
Consider the following reduction from 3CNF: (this is similar to the textbook Clique reduction)
- Let $\phi$ be a 3CNF with $m$ clauses and $n$ variables, where
  - $\phi = c_1 \land c_2 \land \ldots \land c_m$
  - $\forall i: \forall 1 \leq i_1 \leq n, 1 \leq i_2 \leq 3$
    - $l_{i,j}$ is either $x_k$ or $\overline{x_k}$.
- def $f'(\phi)$:
  create 3m nodes, one for each $l_{i,j}$ where $1 \leq i \leq n, 1 \leq j \leq 3$
  for each $1 \leq i_1 \leq n, 1 \leq j_1 \leq 3$
    $1 \leq i_2 \leq n, 1 \leq j_2 \leq 3$
    if $(i_1 \neq i_2)$ and $l_{i_1,j_1} \neq \neg l_{i_2,j_2}$
    then draw edge between $l_{i_1,j_1}$ and $l_{i_2,j_2}$
  return Graph
- def $f(\phi)$:
  $G = f'(\phi)$
  $H = a$ clique of size $(m-1)$
  return ($G$ union $H$)

Useful Lemma:
- If $\phi$ is satisfiable, the largest clique of $f(\phi)$ has size $m$.
- If $\phi$ is unsatisfiable, the largest clique of $f(\phi)$ has size $m - 1$.

Proof:
- The size of the largest clique can never be less than $m - 1$ due to the union with $H$.
- The size of the largest clique can never be more than $m$ since in $G$, we can take a most one node from each clause.
- If $\phi$ is satisfiable, we take the clique from $G$, and have size $m$.
- If $\phi$ is unsatisfiable, we take the clause from $H$, and have size $m - 1$.

Proof of NP-hard:
Given an instance $\phi$ of 3CNF, consider the query $\langle f(\phi), m \rangle$. It follows that the largest clique of $f(\phi)$ has size $m$ iff $\phi$ is satisfiable. Thus, ExactClique is NP-Hard.

Proof of coNP-hard:
Given an instance $\phi$ of 3CNF, consider the query $\langle f(\phi), m - 1 \rangle$. It follows that the largest clique of $f(\phi)$ has size $m - 1$ iff $\phi$ is not satisfiable. Thus, ExactClique is NP-Hard.
Problem 3: Circuit Minimization (40)

CircuitMin in polytime ⇒ $P = NP$ (10 points)

- We will solve $\phi \in 3CNF$ is poly time.
- By definition, each $3CNF$ is a circuit, so $\phi$ is a circuit.
- Let $c = minimize(\phi)$.
- If $c$ is a single gate saying “False”, we know that $\phi$ is not satisfiable.
- Otherwise, $\phi$ is satisfiable.

$P = NP$ ⇒ CircuitMin in polytime (30 points)

- Fact 1: a circuit with $m$ gates can be encoded using $10^m \log m$ bits.
  Proof: each gate can be encoded using $10 \log m$ bits. There are $m$ gates. Thus $10^m \log m$.
- Define $decode : \{0, 1\}^* \rightarrow Circuit$ be the function which interprets binary strings as Circuits.
- CircuitEQ is the language consisting of all pairs of circuits that encode the same function. Formally, 
  $CircuitEQ = \{C_1, C_2 \mid \forall x : C_1(x) = C_2(x)\}$.
  We note that $CircuitEQ \in NP$ (since a $x$ s.t. $C_1(x) \neq C_2(x)$ is a witness).
  Thus, $CircuitEQ \in P$ and $CircuitEQ \in P$.
- CircuitExist is the language consisting of all triplets (Circuit, Int, Prefix) where there exists an equivalent circuit of a certain size or smaller, starting with a given prefix. Formally,
  $CircuitExist = \{(C_1, 1^k, x) \mid \exists y : (|x \circ y| \leq 10^k \log k) \land (CircuitEQ(C_1, decode(x \circ y))) \land (size(decode(x \circ y)) \leq k)\}$.
  $CircuitExist \in NP$ since (1) the string $y$ serves as a witness and (2) $CircuitEQ$ can be evaluated in polynomial time.
- We now define our minimizer

```python
def find_opt_size(C):
    smallest_size = size(C);
    while (Circuit_Exist(C, smallest_size, "")):
        smallest_size--;
    return smallest_size+1;

def find_a_circuit_of_opt_size(C, opt_size, cur_prefix):
    if (cur_prefix decodes to circuit of size opt_size): 
        return cur_prefix 
    else if (Circuit_Exist (C, opt_size, cur_prefix ++ "0")):
        // can we add a 0, and still find a circuit? 
        then return Circuit_Exist (C, opt_size, cur_prefix ++ "0"
    else return Circuit_Exist (C, opt_size, cur_prefix ++ "1"

def CircuitMin (C):
    opt_size = find_opt_size(C):
    return find_a_circuit_of_opt_size(C, opt_size, "")
```