CS452 HW8

Yuting Chen

7.6

Union: Assume that language $L_1 \in P$, $L_2 \in P$.

$\Rightarrow$ There are polynomial-time TMs M_1 and M_2 that decide L_1 and L_2 respectively.

A Turing-Machine M that decides $L_1 \cup L_2$ as follow:

M = "On input w:

1. Run M_1 with input w. If M_1 accepts, accept.
2. Run M_2 with input w. If M_2 accepts, accept.
3. Otherwise, reject.

M accepts w if and only if either M_1 or M_2 accepts w.

Both stages take polynomial time, the algrithm runs in polynomial time.

So, M decides the union of L_1 and L_2.

Concatenation: Assume that language $L_1 \in P$, $L_2 \in P$.

$\Rightarrow$ There are polynomial-time TMs M_1 and M_2 that decide L_1 and L_2 respectively.

A Turing-Machine M that decides $L_1 \circ L_2$ as follow:

M = "On input w:

1. For each way cut w into two substrings such that $w = w_1 w_2$
2. Run M_1 on w_1 and run M_2 on w_2.
3. If M_1, M_2 are both accept, accept.
4. If we finished try all possible cuts, w is still not accepted, reject.

M accepts w if and only if w can be written as $w_1 w_2$ and M_1 accepts w_1, M_2 accepts w_2.

Stage 2 runs in polynomial time and it is repeated at most $O(n)$.

The algorithm runs in polynomial time.

So, M' decides the concatenation of L_1 and L_2.

Complement: Assume that the language $L_1 \in P$.

$\Rightarrow$ There is a polynomial time TM M_1 that decides L_1.

A Turing-Machine M that decides $\overline{L_1}$ as follow:

M = "On input w:

1. Run M_1 with input w.
2. If M_1 accepts, reject.
3. Otherwise, accept.

The Turing Mechine M just output the opposite of M_1 does.

So, M decides $\overline{L_1}$.

M_1 runs in polynomial time, M also runs in polynomial time.

So, M decides the complement of L.

7.7

Union: Let A and B be languages are decide by NP.

Let M_1, M_2 be the polynomial time nondeterministic decider.

We construct a NTM M that decides $A \cup B$ in polynomial time.

M = "On input w:

1. Check if w is a member of A or B.
2. If A was decided, run M_1 on w, output M_1's decision.
3. If B was decided, run M_2 on w, output M_2's decision.

Runtime: $O(1) + MAX(O(M_1), O(M_2)) = MAX(O(M_1), O(M_2))$

M_1, M_2 run in polynomial time.

$\Rightarrow MAX(O(M_1), O(M_2))$ is polynomial time.

M decides $A \cup B$ in polyonimial time.

So, A union B is in NP.

Concatenation: Let A and B be languages are decide by NP.

Let M_1, M_2 be the polynomial time nondeterministic decider.

We construct a NTM M that decides $A \circ B$ in polynomial time.

M = "On input w:

1. No-deterministically divide w into X and Y.

2. Run M_1 on X
3. Run M_2 on Y.
4. If M_1 and M_2 are both accepted, accept.
5. Otherwise, reject.

Runtime: $O(1) + MAX(O(M_1), O(M_2)) = MAX(O(M_1), O(M_2))$

M_1, M_2 run in polynomial time.

$\Rightarrow MAX(O(M_1), O(M_2))$ is polynomial time.

M decides $A \circ B$ in polyonimial time.

So, A concatinate B is in NP.

7.15

Let $A \in P$

$\Rightarrow$ There exists a deterministic Turing machine M_A. The runtime complexity $O(n^k)$ for $k \geq 0$.

We need to show that $A^* \in P$.

Let $W_{i:j}$ denote the substring of $W = W_1W_2W_3 \cdots W_n$ starting with W_i, ending with W_j.

Build a table where $T(i,j) = 1$ as true. if $W_{i:j} \in A^*$.

Consider all substrings of w starting with substrings of length 1 and ending with n.

"On input $W = W_1W_2 \cdots W_n$:

1. If w is empty string, accept.
2. Initailze $T[i,j] = 0$ for $1 \leq i \leq j \leq n$.
3. For $i = 1$ to n.
4. If W_i is in A. Set $T[i,j] = 1$
5. For $l = 2$ to n
6. For $i = 1$ to $n - l + 1$
7. Let $j = i + l - 1$
8. If $W_i \cdots W_j$ is in A, set $T[i,j] = 1$
9. For $k = i$ to $j - 1$
10. If $T[i,k] = 1$ and $T[k,j] = 1$, set $T[i,j] = 1$.

11. If $T[1,n] = 1$, accept.
12. Else, reject.
Each stage takes polynomial time, and it executes takes $O(n^3)$ which is polynomial in n.

7.18

Assume that $P = NP$
Let $A \in P$ such that $A \neq \phi$ and $A \neq \Sigma^*$.
This means there is a string $W_{in} \in A$ and a string $W_{out} \notin A$.
We need to show that A is NP-complete.
By our assumption, the language A is in $NP = P$.
Let B be an arbitrary language from $NP = P$. $B \in NP$, $B \leq_P A$.
So, the language has a polynomial decider M.
The polynomial reduction from B to A will be as follow:
"On input w:
1. Run M (decider for B) on w.
2. If M accepted, then output W_{in}.
3. If M rejected, then output W_{out}.
So, there exists a polynomial time from B to A.
Thus, A is NP-complete.

7.21

b. Assume that NP-completeness of UHAMPATH, the Hamiltonian path problem for undirected graphs.
A simple path of length at least k from a to b.
Verify it in polynomial time. $\Rightarrow$ LPATH $\in$ NP.
Set TM M computes the reduction f:
M = "On input $\langle G, a, b \rangle$ where graph G has nodes a and b:
1. Let k be the number of nodes of G.

2. Output $\langle G, a, b, k \rangle$

$\Rightarrow$ UHAMPATH $\leq_p$ LPATH

If $\langle G, a, b \rangle \in$ UHAMPATH, then G contains a Hamiltonian path of length k from a to b.

$\Rightarrow \langle G, a, b, k \rangle \in$ LPATH.

If $\langle G, a, b, k \rangle \in$ LPATH, then G contains a simple path of length k from a to b.

As we know that G has only k nodes.

So, the path is Hamiltonian. $\langle G, a, b \rangle \in$ UHAMPATH.

Thus LPATH is NP-complete.

7.38

Assume P = NP.

Using the polynomial time algorithm of SAT.

Give a satisfactory assignment to the satisfiable formula φ.

Substitute $X_1 = 0$, $X_2 = 1$ in φ.

Test the satisfiablity of two result φ_0, φ_1.

We know that φ is satisfiable.

So, at least one must be satisfiable.

If φ_0 satisfiable, pick $X_1 = 0$.

If φ_1 satisfiable, pick $X_1 = 1$.

Therefore, we could give value of X_1 in satisfy assignment.

As same, determine a value for X_2, and make that substitution permanent.

Keep doing like this until all variables are replaced.

7.43

On input ϕ.

Build a NFA N that chooses nondeterministically one of the c clauses and reads input length m.

If clause is not satisfied, accept. Otherwise, rejects

N accepts all inputs of length not equal l.

Also N consists $O(cm)$ states that shows it can work in polynomial time.
For any nonsatisfying assignment a that at least one clause is not satisfied. $\Rightarrow$ N accepts a.
If N accept a, clause is not satified. so a is a nonsatisfying assignment.
$\Rightarrow$ N accepts all the nonsatisfying assignment of ϕ.
Let minimization of NFAs take polynomial time.
Build a NFA N that accepts ϕ's nonsatisfying assignments.
Also N accepts all binary string if and only if ϕ is not satisfiable.
For new NFA N' execute the NFA minimizing algorithm, reject ϕ.
If N' contains exactly 1 state, then accepts all binary strings. Otherwise, accept ϕ.
This produces a polynomial time algorithm for 3SAT.
Thus, P = NP.