Tutorial 9

Exercise 1 (compulsory)

Consider the following context-free grammar G in Chomsky normal form:

$$S \to AA \mid \epsilon$$
$$A \to BB \mid AB \mid a$$
$$B \to BA \mid b$$

Following the algorithm in the proof of Theorem 7.16 (see also Lecture 9) compute the entries $table(i, j)$ for all $1 \leq i \leq j \leq 4$ in order to show that $abba \in L(G)$.

Solution:

$l = 1$

	1	2	3	4
1	A			
2		B		
3			B	
4				A

$l = 2$

	1	2	3	4
1	A	A		
2		B	A	
3			B	B
4				A

$l = 3$

	1	2	3	4
1	A	A	S, A	
2		B	A	A, S
3			B	B
4				A

$l = 4$

	1	2	3	4
1	A	A	S, A	S, A
2		B	A	A, S
3			B	B
4				A

Notice that S appears in the entry $(1, 4)$, which implies that $abba \in L(G)$.

Exercise 2 (compulsory)

Prove that the class P is closed under intersection, complement and concatenation.

Solution:

- **Intersection.** Let $L_1, L_2 \in$ P. We want to show that $L_1 \cap L_2 \in$ P. Because $L_1 \in$ P then there exists a TM M_1 with time complexity $O(n^{k_1})$ for some constant k_1. Because $L_2 \in$ P then there exists a TM M_2 with time complexity $O(n^{k_2})$ for some constant k_2. We construct a decider M with polynomial time complexity deciding $L_1 \cap L_2$:

  M = "On input x:

  1. Run M_1 on input x
  2. If M_1 accepted, then run M_2 on input x, else reject.
  3. If M_2 also accepted, then accept, else reject."

  In the worst case M will run both M_1 and M_2, in which case it uses $O(n^{k_1}) + O(n^{k_2})$ steps. Let $k = \max(k_1, k_2)$. We then see that M has time complexity $O(n^k)$ and hence $L(M) = L_1 \cap L_2 \in$ P. (Note that we omitted the details where a copy of the string x for the machine M_2 is stored while the machine M_1 is computing; this can be done e.g. on a second tape but we know that all deterministic variants of Turing machines are polynomial-time equivalent.)

- **Complement.** The same construction as for decidable languages (see e.g. slide 13 in Lecture 3). We simply swap the accept and reject state. The running time of the modified machine does not change.
- **Concatenation.** We want to show that if $L_1, L_2 \in$ P then $L_1 \circ L_2 \in$ P. Assume so that $L_1 \in$ P and that $L_2 \in$ P. By definition, this means that there exist deciders M_1 and M_2 such that M_1 is a decider for L_1 with time complexity $O(n^{k_1})$ and M_2 is a decider for L_2 with time complexity $O(n^{k_2})$ for some constants k_1 and k_2.

  The concatenation $L_1 \circ L_2$ is defined as

  $$L_1 \circ L_2 = \{x_1 x_2 \mid x_1 \in L_1, x_2 \in L_2\}$$

  The decider for $L_1 \circ L_2$ must, given an input x, try to find a partition of x into $x_1 x_2$ such that $x_1 \in L_1$ and $x_2 \in L_2$. Here is the decider:

  "On input $x = a_1 \dots a_n$

  1. For $i = 0$ to n do
     i. Let $x_1 = a_1 \dots a_i$ and $x_2 = a_{i+1} \dots a_n$. (By agreement $a_1 \dots a_0 = \epsilon$ and $a_{n+1} \dots a_n = \epsilon$).
     ii. Run M_1 on the input x_1.
     iii. Run M_2 on the input x_2.
     iv. If both M_1 and M_2 accepted, then accept
  2. If no choice of x_1 and x_2 led to acceptance, then reject"

  We must now show that the decider has polynomial time complexity. The main loop of the decider is traversed at most (n+1)-times. If we run M_1 on a substring of x, this will take at most $O(n^{k_1})$ steps. Similarly, running M_2 on a substring of x will take at most $O(n^{k_2})$ steps. Consequently, a single traversal of the loop body uses no more than $O(n^{k_1}) + O(n^{k_2}) = O(n^k)$ steps, where $k = \max(k_1, k_2)$. The whole decider thus uses $(n+1) \cdot O(n^k) = O(n^{k+1})$ steps. Hence $L(M) = L_1 \circ L_2 \in$ P.

Exercise 3 (compulsory)

Prove the following theorem.

Theorem: Let $t(n) \geq n$ be a function from natural numbers to positive reals. Then for every language $L \in$ NTIME($t(n)$) there is a constant c such that $L \in$ TIME($2^{c \cdot t(n)}$).

Solution:
This is in fact a simple test on applying the definitions and Theorem 7.11 on page 284.

Assume a language $L \in$ NTIME($t(n)$) for some function $t(n) \geq n$. By definition of NTIME there is a nondeterministic single-tape TM M running in time $O(t(n))$ and deciding L. By Theorem 7.11 we know that for M we can construct an equivalent TM M' such that M' is a single-tape deterministic decider for L running in time $2^{O(t(n))}$. Because $2^{O(t(n))}$ means $O(2^{c \cdot t(n)})$ for some constant c, we get (by definition) that $L \in$ TIME($2^{c \cdot t(n)}$) and we are done.

Exercise 4 (compulsory)

Every week someone manages to "prove" that P = NP or that P $\neq$ NP. Last week a very famous professor published the following proof that P $\neq$ NP:

Proof: Consider the following decider for $HAMPATH$:

"On input $\langle G, s, t \rangle$:

1. Generate all possible permutations of nodes from G.
2. If one of these permuations (sequences of nodes) forms a Hamiltonian path, then accept.
3. Otherwise reject."

Because there are $n!$ different permuations of nodes to examine, the algorithm clearly does not run in polynomial time. Therefore we have proved that $HAMPATH$ has exponential time complexity and this means $HAMPATH \notin$ P. Because we know that $HAMPATH \in$ NP, we conclude that P $\neq$ NP.

Describe the error in the above proof.

Solution:
It is true that the suggested *algorithm* for $HAMPATH$ does not run in polynomial time, but from this fact we cannot conclude that the *language* $HAMPATH$ does not belong to the class P. There can still be other (faster) algorithms for $HAMPATH$ that run in polynomial time; we simply did not exclude this possibility by presenting one particular algorithm with an exponential running time. To conclude that $HAMPATH \notin$ P we would have to show *no* algorithm for $HAMPATH$ has a polynomial time complexity.

Exercise 5 (optional, but highly recommended)

Prove that the class P is closed under Kleene star. (**Hint:** use dynamic programming.)

Solution:
Let $A \in$ P. We want to show that $A^* \in$ P. Since $A \in$ P there exists a deterministic Turing machine M_A with time complexity $O(n^k)$ for some $k \geq 0$.

We now build, using M_A, a deterministic decider for A^* and show that its time complexity is bounded by a polynomial. The central observation in our construction is that $w \in A^*$ if and only if one of the following conditions is true

- $w = \varepsilon$, or
- $w \in A$, or
- $\exists u, v : w = uv$ and $u \in A^*$ and $v \in A^*$.

In the decider described below we let $w_{i,j}$ denote the substring of $w = w_1 w_2 \dots w_n$ starting with w_i and ending with w_j. The decider builds a table where $table(i, j) = true$ if $w_{i,j} \in A^*$. We do this by considering all substrings of w starting with substrings of length 1 and ending with the substring of length n.

"On input $w = w_1 w_2 \dots w_n$:
1. If $w = \varepsilon$ then accept, else
2. For $\ell := 1$ to n
3. For $i := 1$ to $n - (\ell - 1)$
4. $j := i + \ell - 1$
5. Run M_A on $w_{i,j}$
6. If M_A accepts $w_{i,j}$ then $table(i, j) := true$
7. Else
8. For $k := i$ to $j - 1$
9. If $table(i, k) = true$ and $table(k + 1, j) = true$
10. then $table(i, j) := true$
11. If $table(1, n) = true$ then accept, else reject."

We now analyze the complexity of our decider. The algorithm uses three nested loops, each of which can be traversed at most $O(n)$ times. In the second loop we run M_A on an input of length at most n, so the total time is at most $O(n) \cdot O(n) \cdot (O(n^k) + O(n)) = O(n^{2+(\max(k,1)}))$ steps, which is polynomial in n.