Homework 11 Solutions

1. Answer each part TRUE or FALSE.

   (a) $2n = O(n)$.

   Answer: TRUE. We can see this by letting $c = 2$, and noting that $2n \leq cn = 2n$ for all $n \geq 1$. Thus, the definition of big-O holds for $c = 2$ and $n_0 = 1$.

   (b) $n^2 = O(n)$.

   Answer: FALSE. For it to be true, we would need that there exist positive constants c and n_0 such that $n^2 \leq cn$ for all $n \geq n_0$. By dividing both sides by n, we see that "$n^2 \leq cn$ for all $n \geq n_0$" is true if and only if "$n \leq c$ for all $n \geq n_0$" is true, but clearly there cannot exist constants c and n_0 for which the last statement is true.

   (c) $n^2 = O(n \log^2 n)$.

   Answer: FALSE. To see why, note that $n^2 = O(n \log^2 n)$ if and only if there exist positive constants c and n_0 such that $n^2 \leq cn \log^2 n$ for all $n \geq n_0$, which holds if and only if

   $$\frac{n^2}{n \log^2 n} \leq c \quad \text{for all} \quad n \geq n_0. \tag{1}$$

   By cancelling out an n from the numerator and denominator, we can rewrite equation (1) as $n/(\log^2 n) \leq c$ for all $n \geq n_0$. Writing $n = n^{1/2}n^{1/2}$, we see that the last statement is true if and only if $[n^{1/2}/(\log n)]^2 \leq c$ for all $n \geq n_0$. But because we know that $\log n = o(n^{1/2})$, we have that $n^{1/2}/(\log n) \to \infty$ as $n \to \infty$, so $[n^{1/2}/(\log n)]^2 \leq c$ cannot be true for all $n \geq n_0$ for any constant c.

   (d) $n \log n = O(n^2)$.

   Answer: TRUE. Because $\log n = O(n)$, there exist positive constants c and n_0 such that $\log n \leq cn$ for all $n \geq n_0$. Multiplying both sides by n gives $n \log n \leq cn^2$ for all $n \geq n_0$ for the same positive constants c and n_0, so $n \log n = O(n^2)$.

(e) $3^n = O(2^n)$.

Answer: FALSE. Suppose that it were true. Then there exists constants c and n_0 such that $3^n \leq c2^n$ for all $n \geq n_0$. The last requirement is equivalent to $(3/2)^n \leq c$ for all $n \geq n_0$. However, $(3/2)^n \to \infty$ as $n \to \infty$, so $(3/2)^n \leq c$ cannot be true for all $n \geq n_0$ for any constant c.

(f) $3^n = 2^{O(n)}$.

Answer: TRUE because $3^n = 2^{n \log_2 3} = 2^{O(n)}$.

(g) $2^{2^n} = O(2^{2^n})$.

Answer: TRUE. Any function $f(n)$ is $O(f(n))$.

2. Let $b > 1$ be a constant. Show that $O(t(n)) \times O(b^{t(n)}) = 2^{O(t(n))}$.

Answer: Let $f_1(n) = O(t(n))$ and $f_2(n) = O(b^{t(n)})$, so we want to show that $f_1(n) f_2(n) = 2^{O(t(n))}$. Because $f_1(n) = O(t(n))$, there exist constants c_1 and n_1 such that $f_1(n) \leq c_1\, t(n)$ for all $n \geq n_1$. Because $f_2(n) = O(b^{t(n)})$, there exist constants c_2 and n_2 such that $f_2(n) \leq c_2\, b^{t(n)}$ for all $n \geq n_2$. Consequently, letting $c_0 = c_1 c_2$, we get

$$f_1(n)\, f_2(n) \;\leq\; c_0\, t(n)\, b^{t(n)}$$

for all $n \geq n_0 \equiv \max(n_1, n_2)$. Now recall that $x = 2^{\log_2(x)}$ for any x, so

$$\begin{aligned} c_0\, t(n)\, b^{t(n)} &= 2^{\log_2(c_0\, t(n)\, b^{t(n)})} \\ &= 2^{\log_2(c_0) + \log_2(t(n)) + t(n) \log_2(b)} \\ &= 2^{O(t(n))} \end{aligned}$$

because $\log_2(t(n)) = O(t(n))$.

3. (a) Show that P is closed under union.

Answer: Suppose that language $L_1 \in$ P and language $L_2 \in$ P. Thus, there are polynomial-time TMs M_1 and M_2 that decide L_1 and L_2, respectively. Specifically, suppose that M_1 has running time $O(n^{k_1})$, and that M_2 has running time $O(n^{k_2})$, where n is the length of the input w, and k_1 and k_2 are constants. A Turing machine M_3 that decides $L_1 \cup L_2$ is the following:

M_3 = "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*.

Otherwise, *reject*."

Thus, M_3 accepts w if and only if either M_1 or M_2 (or both) accepts w. The running time of M_3 is $O(n^{k_1}) + O(n^{k_2}) = O(n^{\max(k_1, k_2)})$, which is still polynomial in n; i.e., the sum of two polynomials is also polynomial. Thus, the overall running time of M_3 is also polynomial.

(b) Show that P is closed under concatenation.

Answer: Suppose that language $L_1 \in$ P and language $L_2 \in$ P. Thus, there are polynomial-time TMs M_1 and M_2 that decide L_1 and L_2, respectively. Specifically, suppose that M_1 has running time $O(n^{k_1})$, and that M_2 has running time $O(n^{k_2})$, where n is the length of the input, and k_1 and k_2 are constants. A Turing machine M_3 that decides the concatenation $L_1 \circ L_2$ is the following:

M_3 = "On input $w = a_1 a_2 \cdots a_n$, with each $a_i \in \Sigma$ a symbol:

1. For $i = 0, 1, 2, \ldots, n$, do

2. Run M_1 with input $w_1 = a_1 a_2 \cdots a_i$, and run M_2 with input $w_2 = a_{i+1} a_{i+1} \cdots a_n$. If both M_1 and M_2 accept, *accept*.

3. If none of the iterations in Stage 2 accept, *reject*."

The TM M_3 checks every possible way of splitting the input w into two parts w_1 and w_2, and checks if the first part w_1 is accepted by M_1 (i.e., $w_1 \in L_1$) and the second part w_2 is accepted by M_2 (i.e., $w_2 \in L_2$), so that $w_1 w_2 \in L_1 \circ L_2$. Suppose that the input w to M_3 has length $|w| = n$. Stage 2 is executed at most $n + 1$ times. Each time Stage 2 is executed, M_1 and M_2 are run on strings w_1 and w_2 with $|w_1| \leq |w| = n$ and $|w_2| \leq |w| = n$. Thus, running M_1 on w_1 takes $O(n^{k_1})$ time, and running M_2 on w_2 takes $O(n^{k_2})$ time, so Stage 2 runs in time $O(n^{k_1}) + O(n^{k_2}) = O(n^{\max(k_1,k_2)})$, which is polynomial in n. Because Stage 2 is executed at most $n+1$ times, we get that the time complexity of M_3 is $O(n+1)O(n^{\max(k_1,k_2)}) = O(n^{1+\max(k_1,k_2)})$, which is polynomial in n. Hence, the overall running time of M_3 is polynomial in n.

(c) Show that P is closed under complementation.

Answer: Suppose that language $L_1 \in$ P, so there is a polynomial-time TM M_1 that decides L_1. A Turing machine M_2 that decides $\overline{L_1}$ is the following:

M_2 = "On input w:

1. Run M_1 with input w. If M_1 accepts, *reject*; otherwise, *accept*."

The TM M_2 just outputs the opposite of what M_1 does, so M_2 decides $\overline{L_1}$. Because M_1 is a polynomial-time TM, so is M_2.

4. A **triangle** in an undirected graph is a 3-clique. Show that $TRIANGLE \in$ P, where $TRIANGLE = \{ \langle G \rangle \mid G \text{ contains a triangle} \}$.

Answer: Let $G = (V, E)$ be a graph with a set V of vertices and a set E of edges. We enumerate all triples (u, v, w) with vertices $u, v, w \in V$ and $u < v < w$, and then check whether or not all three edges (u, v), (v, w) and (u, w) exist in E. Enumeration of all triples requires $O(|V|^3)$ time. Checking whether or not all three edges belong to E takes $O(|E|)$ time. Thus, the overall time is $O(|V|^3\,|E|)$, which is polynomial in the length of the input $\langle G \rangle$. Therefore, *TRIANGLE* $\in$ P.

Remark: Note that for *TRIANGLE*, we are looking for a clique of fixed size 3, so even though the 3 is in the exponent of the time bound, the exponent is a constant, so the time bound is polynomial. We could modify the above algorithm for *TRIANGLE* to work for $CLIQUE = \{\, \langle G, k \rangle \mid G$ is an undirected graph with a k-clique $\}$ by enumerating all collections of k vertices, where k is the size of the clique desired. But the number of such collections is

$$\binom{|V|}{k} = \frac{|V|!}{k!(|V|-k)!} = O(|V|^k),$$

so the time bound is $O(|V|^k\, k|E|)$, which is exponential in k. Because k is part of the input $\langle G, k \rangle$, the time bound is no longer polynomial. Hence, we cannot use this algorithm to show that *CLIQUE* $\in$ P. Nor does it show that *CLIQUE* $\notin$ P since we've only shown that one algorithm doesn't have polynomial runtime, but there might be another algorithm for *CLIQUE* that does run in polynomial time. However at this time, it is currently unknown if *CLIQUE* $\in$ P or *CLIQUE* $\notin$ P. Because *CLIQUE* is NP-complete, this question will be answered if anyone solves the P vs. NP problem, which is still unresolved.

5. Using the polynomial-time algorithm for context-free language recognition (i.e., the CYK algorithm or dynamic programming), fill out the table for string $w = abba$ and CFG G:

$$\begin{array}{rcl} S & \rightarrow & RT \\ R & \rightarrow & TR \mid a \\ T & \rightarrow & TR \mid b \end{array}$$

Answer: We start the CYK algorithm by writing an empty $n \times n$ table, where $n = 4$ is the length of the string $w = abba$ that we want to determine if the CFG G (in Chomsky normal form) can generate.

	1	2	3	4
1				
2				
3				
4				
string	a	b	b	a

Note that we numbered the rows and columns, and we wrote the string $abba$ along the bottom of the table. We will fill in the table a diagonal at a time, starting with the main diagonal. Once one diagonal is filled in, we move to the diagonal directly above it. The entry $\mathit{table}(i,j)$ corresponds to the substring starting in position i and ending in position j, and we put into $\mathit{table}(i,j)$ those variables that we can start with to generate that substring.

We start by filling in the main diagonal, which consists of the entries $\mathit{table}(1,1)$, $\mathit{table}(2,2)$, $\mathit{table}(3,3)$, and $\mathit{table}(4,4)$. Entry $\mathit{table}(1,1)$ corresponds to the substring starting in position 1 and ending in position 1, which is the substring a. Because the given CFG is in Chomsky normal form, the only way a variable can generate this substring is if there is a rule that has this variable go directly to a. Because there is a rule $R \to a$, we add the variable R into $\mathit{table}(1,1)$. There are no other rules with a on the right side, so we don't add any other variables to $\mathit{table}(1,1)$; hence, $\mathit{table}(1,1) = \{T\}$.

	1	2	3	4
1	R			
2				
3				
4				
string	a	b	b	a

We now move to the main diagonal's next entry, $\mathit{table}(2,2)$. This corresponds to the substring starting in position 2 and ending in position 2, which is the substring b. The only way to generate this substring is through the rule $T \to b$, so $\mathit{table}(2,2) = \{T\}$.

	1	2	3	4
1	R			
2		T		
3				
4				
string	a	b	b	a

Similarly, for entries $\mathit{table}(3,3)$ and $\mathit{table}(4,4)$, which correspond to substrings b and a, respectively, we have $\mathit{table}(3,3) = \{T\}$ and $\mathit{table}(4,4) = \{R\}$.

	1	2	3	4
1	R			
2		T		
3			T	
4				R
string	a	b	b	a

Now that the main diagonal is complete, we next fill in the diagonal just above it. Entry $\mathit{table}(1,2)$ corresponds to the substring starting in position 1 and ending in position 2, which is the substring ab. We can divide this substring into two shorter parts, a and b, and we see how we can generate these two shorter parts.

- For the first part a, which starts in position 1 and ends in position 1, we look in entry $table(1,1)$ to see that R can generate this part a.
- For the second part b, which starts in position 2 and ends in position 2, we look in entry $table(2,2)$ to see that T can generate this part b.

Now if we have a rule in which the right side pairs a variable from $table(1,1)$ with a variable from $table(2,2)$, then the variable on the left side of the rule can generate the current substring ab. Specifically, we are looking for rules whose right sides are from $table(1,1) \circ table(2,2)$. Since $table(1,1) = \{R\}$ and $table(2,2) = \{T\}$, we have that $table(1,1) \circ table(2,2) = \{RT\}$, so we look for rules with RT on the right side.

- For example, there is a rule $S \to RT$, so the variable S can generate the current substring ab. To see why, consider the right side RT of the rule $S \to RT$. Entry $table(1,1)$ tells us that the R on the right side can generate a, and entry $table(2,2)$ tells us that the T on the right side can generate b. Thus, $S \Rightarrow RT \stackrel{*}{\Rightarrow} ab$, so S can generate ab.

We have now determined that S can generate the current substring ab, which starts in position 1 and ends in position 2, so we add S into $table(1,2)$; hence, $table(1,2) = \{S\}$.

	1	2	3	4
1	R	S		
2		T		
3			T	
4				R
string	a	b	b	a

Now we consider the next entry in the current diagonal. This is $table(2,3)$, which corresponds to the substring starting in position 2 and ending in position 3, which is the substring bb. We divide this substring into two smaller parts, b and b, and we determine how we can generate these two parts.

- For the first part b, which starts in position 2 and ends in position 2, the entry $table(2,2)$ tells us that T can generate this part b.
- For the second part b, which starts in position 3 and ends in position 3, the entry $table(3,3)$ tells us that T can generate this part b.

Now if we have a rule in which the right side is from $table(2,2) \circ table(3,3)$ (i.e., the right side pairs a variable from $table(2,2)$ with a variable from $table(3,3)$), then the variable on the left side of the rule can generate the current substring bb. Since $table(2,2) = \{T\}$ and $table(3,3) = \{T\}$, we have $table(2,2) \circ table(3,3) = \{TT\}$, so we are looking for rules that have TT on the right side. But there are no rules with TT on the right side, so it is impossible to generate the current substring bb using the rules of the CFG; thus, we put a dash "—" in entry $(2,3)$ to denote that $table(2,3) = \emptyset$.

	1	2	3	4
1	R	S		
2		T	—	
3			T	
4				R
string	a	b	b	a

Using similar reasoning for entry $table(3,4)$, we look for rules that have right sides from $table(3,3) \circ table(4,4) = \{T\} \circ \{R\} = \{TR\}$. Thus, we look for rules with TR on the right side, which leads us to put R and T in $table(3,4)$, so $table(3,4) = \{R, T\}$.

	1	2	3	4
1	R	S		
2		T	—	
3			T	R, T
4				R
string	a	b	b	a

Now that the second diagonal is complete, we next fill in the diagonal just above it. Entry $table(1,3)$ corresponds to the substring starting in position 1 and ending in position 3, which is the substring abb. We can divide this substring into two nonempty parts in two different ways: a concatenated with bb, and ab concatenated with b.

First consider splitting abb into parts a and bb.

- For the first part a, which starts in position 1 and ends in position 1, entry $table(1,1) = \{R\}$ tells us that R can generate this part a.
- For the second part bb, which starts in position 2 and ends in position 3, entry $table(2,3) = \emptyset$ tells us that nothing can generate this part bb.

Note that $table(1,1) \circ table(2,3) = \{R\} \circ \emptyset = \emptyset$, so it is impossible to generate the current substring abb by using the split a concatenated with bb.

Now consider the other way of splitting the current substring abb into two parts, by concatenating ab with b.

- For the first part ab, which starts in position 1 and ends in position 2, entry $table(1,2) = \{S\}$ tells us that S can generate this part ab.
- For the second part b, which starts in position 3 and ends in position 3, entry $table(3,3) = \{T\}$ tells us that T can generate this part b.

Thus, if we have a rule with right side from $table(1,2) \circ table(3,3) = \{S\} \circ \{T\} = \{ST\}$, then the variable on the left side of the rule can generate the current substring abb. We are thus looking for rules with right side ST, and we add the variables on the left sides of these rules to $table(1,3)$. In this case, there are no rules with ST on the right side, so it is impossible to generate the current substring abb using the split that concatenates ab with b. (Actually, we don't have to even look at the rules to determine that no rule has ST on the right side since the start variable S cannot appear on the right side of any rule because the CFG is in Chomsky normal form.) Since the other way of splitting abb as a concatenated with bb also was impossible, we then see it is not possible to generate abb using the CFG, so $table(1,3) = \emptyset$, which we denote with a dash.

	1	2	3	4
1	R	S	—	
2		T	—	
3			T	R, T
4				R
string	a	b	b	a

Let us now review how we filled in entry $table(1,3)$. We looked for rules that had on the right side either

- a variable from $table(1,1)$ paired with a variable from $table(2,3)$, i.e., a right side from $table(1,1) \circ table(2,3)$, or
- a variable from $table(1,2)$ paired with a variable from $table(3,3)$, i.e., a right side from $table(1,2) \circ table(3,3)$.

Thus, to fill in entry $table(1,3)$, we pair entries by going across row 1 and down column 3. In general, to fill in entry $table(i,j)$, we pair entries by going across row i and down column j, i.e., we look for rules with right sides from $table(i,i) \circ table(i+1,j)$, or $table(i,i+1) \circ table(i+2,j)$, or $\ldots$, or $table(i,j-1) \circ table(n,j)$.

Next we fill in entry $table(2,4)$, which corresponds to the substring starting in position 2 and ending in position 4, which is the substring bba. Using the observation from the previous paragraph, we thus want to pair entries going across row 2 and down column 4. In particular, we look for rules with right sides from

- $table(2,2) \circ table(3,4)$, i.e., we pair a variable from $table(2,2)$ with a variable from $table(3,4)$, or
- $table(2,3) \circ table(4,4)$, i.e., we pair a variable from $table(2,3)$ with a variable from $table(4,4)$.

In the first case, since $table(2,2) = \{T\}$ and $table(3,4) = \{R,T\}$, we have $table(2,2) \circ table(3,4) = \{T\} \circ \{R,T\} = \{TR, TT\}$, so we look for rules with TR or TT on the right side. We find the rules $R \to TR$ and $T \to TR$, so we add the variables R and T to entry $table(2,4)$. There are no rules with TT on the right side, so we don't any more variables to $table(2,4)$ at this point.

In the second case, $table(2,3) \circ table(4,4) = \emptyset \circ \{R\} = \emptyset$, so there is no way to generate the substring bba by the split that concatenates bb with a. Thus, we add no other variables to entry $table(2,4)$, so $table(2,4) = \{R,T\}$ from the R and T in the first case.

	1	2	3	4
1	R	S	—	
2		T	—	R, T
3			T	R, T
4				R
string	a	b	b	a

This completes the third diagonal, so now we move to the next diagonal. The entry $table(1,4)$ corresponds to the substring starting in position 1 and ending in position 4, which is the substring $abba$. To determine how we can generate this substring, we pair entries in the table by going across row 1 and down column 4. In particular, we pair

- a variable from $table(1,1)$ with a variable from $table(2,4)$, or
- a variable from $table(1,2)$ with a variable from $table(3,4)$, or
- a variable from $table(1,3)$ with a variable from $table(4,4)$.

Thus, we look for rules with right sides from

- $table(1,1) \circ table(2,4) = \{R\} \circ \{R, T\} = \{RR, RT\}$, or
- $table(1,2) \circ table(3,4) = \{S\} \circ \{R, T\} = \{SR, ST\}$, or
- $table(1,3) \circ table(4,4) = \emptyset \circ \{R\} = \emptyset$.

For any rule that has any of these right sides, we add the variable on the left of the rule to entry $table(1,4)$. The only rule with these possible right sides are $S \to RT$, so we add S to entry $table(1,4)$.

	1	2	3	4
1	R	S	—	S
2		T	—	R, T
3			T	R, T
4				R
string	a	b	b	a

The table is now complete since we have filled in the upper triangle. To determine if the CFG can generate original string $abba$, which is the substring starting in position 1 and ending in position 4, we check if the starting variable $S \in table(1,4)$. Since it is, CFG G generates the string $abba$. If S were not in the upper right corner, then the CFG would not generate the string.