CSE 555 Homework Four Sample Solutions

Question 1

Define $\texttt{INTDFA} = \{\langle M_1, M_2\rangle : M_1, M_2 \text{ are DFAs for which } L(M_1) \cap L(M_2) \neq \emptyset\}$. Prove that $\texttt{INTDFA}$ is in P.

Solution: consider the "standard" algorithm to do this:

1. Create a DFA D such that $L(D) = L(M_1) \cap L(M_2)$ using the product construction.
2. Run E_{DFA} on input $\langle D\rangle$, and output the opposite.

Since D will have $|Q_1| \times |Q_2|$ states, forming this DFA will take $O(n^2)$ time. The second step is running E_{DFA} and trying to find any reachable final state. This is linear in the size of D. Therefore, this algorithm takes polynomial time, showing that $\texttt{INTDFA} \in$ P.

Question 2

In class we showed that whenever L is a context-free language, L is in P. To do this, we gave a polynomial time dynamic programming algorithm that ran in time $O(n^3)$ on a computer with random access memory in which one integer operation takes constant time. Determine the most accurate bound that you can for the time complexity on a single-tape deterministic Turing machine. That is, what is the smallest function $f(n)$ you can find so that every CFL L belongs to TIME($f(n)$)? (Note: Here g is "smaller than" f if $g(n)$ is o($f(n)$).)

Solution: assume that we are presented $\langle G, w\rangle$ where G is a CFG in CNF and $w = w_1 \cdots w_n$. We split up G and w by a delimiter while also verifying that G is in CNF and $w \in \Sigma^*$ (which takes $O(n)$ time). At this point, the tape head is at the end of w; we scan back to the beginning. If $w = \epsilon$, we linearly scan through the rules and check if $S \to \epsilon$ is a rule; if yes, accept, and otherwise, reject.

We now build a table of size $n \times n \times |\Sigma|$, which takes $O(n^2)$ time (since $|\Sigma|$ is a constant). We also include delimiters between each "column" element, and another delimiter between every "row" element (which only adds to the constant hidden in the notation).

Next, we need to check if there is a rule of the form $A \to b$ with $b = w_i$, and add it to $T[i, i]$. We can create a separate character $\hat{w}_i$ marking where the currently marked symbol is in w. We analyze this run-time: putting the tape head back to the first symbol in w takes $O(n^2)$ time. We mark w_1, then move left until we find a rule of the form $A \to w_1$; if not (i.e., we hit the left-hand end of the tape), then we scan right until we find $\hat{w}_1$, change it back to w_1, and repeat with w_2 (now $\hat{w}_2$). If we do find $A \to w_1$, we will need to scan $O(n^2)$ elements to the appropriate index. This can be easily managed if we keep an $O(\log n)$-size index at each of the elements (making the table size $O(n^2 \log n)$). Therefore, since the input is of size n, this takes $O(n^3 \log n)$.

Now we need to look at each rule of the form $A \to BC$, and update elements in the table. We keep track of 4 counters for i, j, k, ℓ after the table T (in binary) with initialization of $i = 1$, $\ell = 2$, $j = i + \ell - 1$, and $k = i$. We scan back and look at the first rule of the correct form, remembering the triplet of variables (A, B, C); we then scan right until we encounter the 4 counters, also remembering their values. We scan back to the beginning of the table, and progress right until we access $T[i, k]$ and $T[k + 1, j]$. We look at $T[i, k]$: if it contains B, then we scan until we reach $T[k + 1, j]$. If it contains C, we scan to $T[i, j]$ and add A if it isn't already there. We can do comparisons of integers, and since we remembered the 4 values, we can compare to the logarithmic-sized counters written on the tape. We then scan right to the integers, and add 1 to k; if it is equal to $j - 1$, increment i and set k back to i; if i is equal to $n - \ell + 1$, then increment ℓ and set i back to 1 (and also always set j to be $i + \ell - 1$).

Now for the run-time of the previous part. The number of updates to the 4 counters is $O(n^3 \log n)$ (because they are $O(\log n)$-sized). We repeat this over every rule, which is another constant. Searching over the table once requires $O(n^2)$ time overall, so this takes $O(n^5 \log n)$.

Checking if S is in $T[1, n]$ takes $O(n^2)$ time. Therefore, this algorithm takes $O(n^5 \log n)$ time overall.

Question 3

INTPAR is the decision problem: Given a set $S = \{s_1, \ldots, s_n\}$ of integers, an integer k, and an integer b, does there exist a partition of S into k (disjoint) classes, so that the sum of the integers within each class is at most b?
Note: this is the Bin-Packing problem.

Question 3a

Show that INTPAR is in NP.
Solution: the certificate is the k classes. We first verify that each integer is in exactly 1 class, and that each class has sum at most b. Verification of the first part takes polynomial time, as well does the second. Therefore, INTPAR $\in$ NP.

Question 3b

Now suppose that we are given an oracle for INTPAR. The oracle takes zero time to compute, but we must write the question to it, and read its answer, and these do take time. The oracle can be consulted as many times as you need (subject to time constraints). Using the oracle, devise a polynomial time algorithm to solve: Given a set $S = \{s_1, \ldots, s_n\}$ of integers, an integer k, and an integer b, find (and print) a partition of S into k (disjoint) classes, so that the sum of the integers within each class is at most b, if one exists.
Solution: we first query the oracle on the input instance if there exists a partitioning; if the oracle says "No," we reject. Otherwise we proceed as follows:

Find(S) /* returns the k classes; k and b never change */
  If S has at most k entries,
    Form k classes so that each contains at most one entry of S
    Return the classes formed.
  Else
    Choose distinct $\sigma, \tau \in S \setminus \{\sigma\}$ so that the oracle says that $S \setminus \{\sigma, \tau\} \cup \{\sigma + \tau\}$ has a solution. ($\star$)
    $P \leftarrow$ Find($S \setminus \{\sigma, \tau\} \cup \{\sigma + \tau\}$)
    In the class of P containing $\sigma + \tau$, replace that entry by two entries, σ and τ.
    Return P as modified.

Line ($\star$) involves calling the oracle $O(n^2)$ times, and the depth of recursion is also $O(n)$. So the oracle is called $O(n^3)$ times. Also note that ($\star$) must succeed, because some solution for S, which is known to be feasible, must have a class with at least two integers in it.

Question 4

Continuing with INTPAR... Using the oracle from the previous question, devise a polynomial time algorithm to solve: Given a set $S = \{s_1, \ldots, s_n\}$ of integers, and an integer k, find (and print) the smallest b for which a partition of S into k (disjoint) classes, so that the sum of the integers within each class is at most b, exists.

Because of the answer to 3(b), all we need to do is determine the value of b. If we just try values of b until we find the smallest one, we may not get a polynomial time method because the *value* of b can be exponential in the size of the input. To avoid this, we use a binary search.

The minimum value of b that works, say b_{min} satisfies $b_{min} \geq 0$ and $b_{min} \leq \sum_{i=1}^{n} s_i$. A binary search of the range $[0, \sum_{i=1}^{n} s_i]$ takes time proportional to $\log_2(\sum_{i=1}^{n} s_i)$. This is polynomial in the size of the input.

Question 5

The *regular closure* of $\mathcal{L}$, $\mathtt{rc}(\mathcal{L})$, is the minimal set $\mathcal{M}$ for which (1) $\mathcal{L} \subseteq \mathcal{M}$, (2) if $L_1, L_2 \in \mathcal{M}$ then $L_1 L_2 \in \mathcal{M}$, (3) if $L_1, L_2 \in \mathcal{M}$ then $L_1 \cup L_2 \in \mathcal{M}$, and (4) if $L_1 \in \mathcal{M}$ then $L_1^\star \in \mathcal{M}$.

Question 5a

Prove that rc(P) = P.
Solution: this is equivalent to saying P is closed under union, concatenation, and star.

1. P closed under union: let $L_1, L_2 \in$ P be decided in poly-time by M_1, M_2 respectively. We make a TM T to decide $L_1 \cup L_2$:
   T = "On input w:

(a) Run M_1 on w. If it accepts, *accept.*
(b) Run M_2 on w. If it accepts, accept; otherwise, *reject.*"

2. P closed under concatenation: let $L_1, L_2 \in$ P be decided in poly-time by M_1, M_2 respectively. We make a TM T to decide L_1L_2:
   $T =$ "On input w:
   (a) For each way to split w into two pieces w_1w_2:
      i. Run M_1 on w_1, and M_2 on w_2. If both machines accept, *accept.*
   (b) If w cannot be split to have both machines accept, *reject.*"
3. P closed under star: Let M be a TM that decides A in polynomial time. For any $w \in A^*$, $w = \epsilon, w \in A$, or there exist u, v such that $w = uv$ and $u, v \in A^*$. We build a table $T[i, j]$ to be 1 if $w_i \cdots w_j \in A^*$. We construct a TM T to decide A^* in polynomial time:
   $T =$ "On input $w \in \Sigma^*$:
   (a) If $w = \epsilon$, accept.
   (b) Create a table $T[i, j]$ for $1 \leq i \leq j \leq n$, initially all 0's.
   (c) Set $T[i, i] = 1$ for all i if $w_i \in A$.
   (d) For $\ell = 2$ to n, $i = 1$ to $n - \ell + 1$ do:
      i. Set $j = i + \ell - 1$.
      ii. If $w_i \cdots w_j \in A$, set $T[i, i] = 1$.
      iii. For $k = i$ to $j - 1$, if $T[i, k] = 1$ and $T[k, j] = 1$, set $T[i, j] = 1$.
   (e) If $T[1, n] = 1$, accept; otherwise, reject."

Question 5b

Prove that `rc`(NP) = NP.

Solution: this is equivalent to saying NP is closed under union, concatenation, and star.

1. NP closed under concatenation: this is the same as for P above, but we instead nondeterministically split w into two pieces, and running M_1, M_2 on these pieces only once.
2. NP closed under union: this is the same as for P above, but we instead nondeterministically choose whether to run M_1 or M_2.
3. NP closed under star: this is Exercise 7.16 in the text, and is answered on page 329.