CoCo - Exam 2015

Johannes Agerskov

Dated: April 13, 2021

Question 1

Let

$$
\begin{aligned}
&L_0 = \{a^k b^l c^m d^n \mid k, l, m, n \geq 0\} \\
&L_1 = \{a^k b^l c^m d^n \mid k, l, m, n \geq 0 \text{ and at least two of } k, l, m, n \text{ are equal}\}, \\
&L_3 = \{a^k b^l c^m d^n \mid k, l, m, n \geq 0 \text{ and at least three of } k, l, m, n \text{ are equal}\}
\end{aligned} \tag{0.1}
$$

Part 1.1

We show that L_0 is regular.

Proof. This follows since $L_0 = L(a^* b^* c^* d^*)$, *i.e.* L_0 is the language of a regular exression. □

Part 1.2

We show that L_1 is context-free, but not regular.

Proof. That L_1 is context free follows from the fact, that it is generated by the CFG

$$\begin{aligned}
S &\to E_{ab}A_{cd} \mid A_{ab}E_{cd} \mid E_{ac}A_d \mid E_{ad} \mid A_aE_{bc}A_d \mid A_aE_{bd}, \\
E_{ab} &\to aE_{ab}b \mid \epsilon, \\
A_{cd} &\to cA_{cd} \mid A_{cd}d \mid \epsilon, \\
E_{ac} &\to aE_{ac}c \mid A_b, \\
A_b &\to bA_b \mid \epsilon, \\
E_{cd} &\to cE_{cd}d \mid \epsilon, \\
A_{ab} &\to aA_{ab} \mid A_{ab}b \mid \epsilon, \\
E_{ad} &\to aE_{ad}d \mid A_{bc}, \\
A_{bc} &\to bA_{bc} \mid A_{bc}c \mid \epsilon, \\
A_a &\to aA_a \mid \epsilon, \\
A_d &\to A_dd \mid \epsilon, \\
E_{bc} &\to bE_{bc}c \mid \epsilon, \\
E_{bd} &\to bE_{bd}d \mid A_c, \\
A_c &\to cA_c \mid \epsilon.
\end{aligned} \tag{0.2}$$

Evidently, any derivation of this CFG will first choose a branch, each of which correspond to choosing which two letters will appear in the same multiplicity (this is what E_ij does) in the final expression, and the rest of such a derivation then simply produce the remaining letters at arbitrary multiplicity ($A_{ij},\ A_i$). That L_1 is not regular follows from the pumping lemma, by the following argument: Assume that L_1 is regular and let p denote the pumping length as given by the pumping lemma. Then a^pb^pc has length greater than p, so by the pumping lemma it may be pumped. Thus $a^pb^pc = xyz$ with $|y| > 0$ and $|xy| \leq p$. Hence we see that $xy = a^k$ and hence $y = a^m$ for some $0 < k, m \leq p$. But then $xy^iz = a^{p+m(i-1)}b^pc$, which for $i = 1$ has no two letters which are of equal multiplicity, and therefore $xy^iz \notin L_1$ and we have a contradiction. We thus conclude that L_1 is not regular. □

Part 1.3

We show that L_2 is in L.

Proof. This follows by constructing a log-space TM that desides L_2. Consider the machine $M =$"On input w

1. Check that $w = a^kb^lc^md^n$ for some $k, l, m, n \geq 0$. If not *reject.*
2. Scan the worktape from left to right, and store counters for the numbers of as, bs, cs, and ds.
3. Check if any three counters are equal, if yes, *accept*, else, *reject.*

Clearly this TM desides L_2, and it stores only counters which are log-space. Notice that step one can be done in log-space since it essentially just checks that $w \in L_1$, and L_1 is regular from which it follows that $L_1 \in L$. Thus it shows that $L_2 \in \mathrm{L}$ □

Question 2

In the following M and N are NFAs. Let $E_{NFA} = \{\langle M \rangle \mid L(M) = \emptyset\}$ and
$EQ_{NFA} = \{\langle M, N \rangle \mid L(M) = L(N)\}$.

Part 2.1

We show that E_{NFA} is in NL.

Proof. By theorem 8.27 it is sufficient to show that E_{NFA} is in coNL. Consider therefore $\overline{E_{NFA}} = \{\langle M \rangle \mid L(M) \neq \emptyset\}$. We claim that the following non-deterministic log-space TM desides $\overline{E_{NFA}}$, $M =$"On input $\langle M \rangle$

1. Store the start state of M on the worktape.
2. Set $i := 0$ on the worktape.
3. While M is not in the accept state:
4. Non-deterministically feed M a letter from the input alphabet of M, and non-deterministically update the state according to Ms transition function on the current state and the fed letter.
5. Update $i := i + 1$
6. If $i > n$, where n is the number of states in M, *reject*
7. *accept.*

Clearly, this machine stores only a counter and the current state of M at all times in the calculation and therefore, it uses at most log-space. On the other hand it desides $\overline{E_{NFA}}$ since it clearly accepts if it find any sequence of letters that consitute an accept string of M, on the other hand, if such a string exists, we know by the pigeon hole principle, that there is an accepting string of length at most the number of states in M. Thus $\overline{E_{NFA}}$ is in NL, and therefore E_{NFA} is in coNL=NL. □

Part 2.2

We show that EQ_{NFA} is in PSPACE.

Proof. Notice first that PSPACE, since PSPACE is based in deterministic TMs, is closed under complementation. If A is in PSPACE there is a deterministic polynomial space TM, M, that

desides A. Define $\overline{M}$ by flipping accept and reject states, then we see that $\overline{M}$ is a polynomial space determinitstic TM that desides $\overline{A}$.
Now we show that $\overline{EQ_{NFA}} = \{\langle M, N\rangle \mid L(M) \neq L(N)\}$ is in PSPACE. Eqiuvalently we may show that $\overline{EQ_{NFA}}$ is in $NPSPACE = PSPACE$ by Savitch's theorem. Consider the following machine, $M =$"On input $\langle M, N\rangle$

1. Non-deterministically store a string, w, on the worktape of length less that nm where n is number of states in N and m is number of states in M.
2. Run M and N on w, if one accepts and the other rejects, *accept*
3. *reject.*

Notice that if any string is longer than nm, then there must be some substring on which N and M start and end in the same state, since N and M has nm possible different combined configurations. Thus if there is a string that is longer than nm, then we may cut away a substring, making it shorter than nm, without changing the outcome of N and M. Therefore, M desides $\overline{EQ_{NFA}}$, furthermore, it clearly runs in polynomial space. Thus we conclude that EQ_{NFA} is in PSPACE. □

Question 3

We define $ONE\text{-}HALT = \{\langle A, B\rangle \mid A, B$ are TMs and on every input exactly one of them halts$\}$.

Part 3.1

We show that $ONE\text{-}HALT$ is not Turing-recognizable

Proof. We use that $\overline{HALT_{TM}}$ is not Turing-recognizable and show that $\overline{HALT_{TM}} \leq_m ONE\text{-}HALT$. That $\overline{HALT_{TM}}$ is not Turing recognizable follows by the simply observation that $HALT_{TM}$ is trivially Turing recognizable but by Theorem 5.1 it is not decidable, thus the claim follows by theorem 4.22
Consider now the following mapping reduction: Given $\langle M, w\rangle$ construct the TM M_w which ignores it input and run M on w, and output $\langle M_w, N\rangle$ where N is the TM that immeadiately accepts (halts) on all input. Clearly if M does not halt on w we have $\langle M_w, N\rangle \in ONE\text{-}HALT$, and if M does halt on w we have $\langle M_w, N\rangle \notin ONE\text{-}HALT$. Clearly this mapping is computable, and we conclude that $\overline{HALT_{TM}} \leq_m ONE\text{-}HALT$. By Corollary 5.29 it follows that $ONE\text{-}HALT$ is not Turing-recognizable. □

Part 3.2

We show that every finite language is decidable.

Proof. This follows from the fact that every finite language is regular, and every regular language is context-free and by theorem 4.9 every context-free language is decidable. □

Part 3.3

We show that a language L is decidable if and only if it is enumerated in lexicographical order by some enumerater, that never halts.

Proof. Clearly if L is decidable there is a decider, D for L, and we can construct an enumerator that for all string, in lexicographical order runs D on the string, and prints it (with suitable seperator symbols) if and only if D accept. This enumerator will enumerate L in lexicographical order, but it will never halt. On the contrary if there is such an enumerator that enumerates L in lexicographical order then either L is finite in which case it is decidable or it is Turing recognizable by theorem 3.21. On the other side if there is a lexicographical order enumerator for L, then we can construct a lexicographical order enumerator for $\overline{L}$ that simply checks for each string, if the enumerator for L prints (can be done because of ordering) that string, if not print it. But then also $\overline{L}$ is Turing recognizable and by theorem 4.22 L is decidable. $\square$

Question 4

Define

$$SUBGRAPH\text{-}100\text{-}ISOMORPHIMSM = \{\langle G_1, G_2\rangle \mid G_1 \text{ is a subgraph of } G_2 \text{ and } |V(G_1)| \leq 100\}$$

Part 4.1

We show that $SUBGRAPH$-100-$ISOMORPHISM$ belongs to P.

Proof. Consider the following polynomial time TM, $M =$"On input $\langle G_1, G_2\rangle$

1. Check that G_1 has less than 100 vertices, and that G_2 has more vertices than G_1, if not *reject.*
2. In all possible ways, one at a time: For each vertex, v, of G_1, associate a vertex, $\tilde{v}$, in G_2.
3. Check for each edge $(u, v) \in E(G_1)$ that $(\tilde{u}, \tilde{v}) \in E(G_2)$.
4. If step 4 is affirmative for some association of vertices, *accept*, if step 4 fails for all associations of vertices *reject.*

Clearly this TM tries to map the vertices of G_1 one-to-one to the vertices of G_2 in all possible ways. If for some one-to-one map, the image of G_1 is a subgraph of G_2, M accept and if this is not the case for any map, M rejects. Thus $L(M) = SUBGRAPH\text{-}100\text{-}ISOMORPHISM$. Furhtermore, Each of the steps are polynomial in time: Step 1 is clearly polynomial time, Step 2 is polynomial in time, since there are at most $|V(G_2)|^{100}$ ways of associating less than 100 vertices to $|V(G_2)|$ vertices, and Step 3 is for each of these associations trivially polynomial in time ($O(n^2)$). Therefore, we conclude that $SUBGRAPH$-100-$ISOMORPHISM$ belongs to P. $\square$