CoCo - Assignment 7 (Exam 2020)

Johannes Agerskov, Mads Friis Frand-Madsen, and Sifan Huang

Dated: March 26, 2021

Question 1

Part 1.1

We prove that the following languages over the alphabet $\{a, b\}$ are regular:

$$
\begin{aligned}
L_i &= \{a^n b^n \mid 1 \leq n \leq 10^n\}, \\
L_{ii} &= \{aa\} \cup \{b^n \mid n \geq 0\}, \\
L_{iii} &= \{a^n b^n \mid n \geq 1\} \cap \{(ab)^n \mid n \geq 1\}, \\
L_{iv} &= \{a^{2n+1} \mid n \geq 0\},
\end{aligned}
\tag{0.1}
$$

and for any fixed $k, d \geq 1$:

$$L_v = \{a^{kn+d} \mid n \geq 0\}.$$

Proof. In the following we will write equality between languages and regular expressions, if and only if the regular expression generates the laguage. $L_i = \cup_{n=1}^{10^{10}} a^n b^n$ which is a regular expression. Therefore, the language L_i is regular. $L_{ii} = aa \cup b^*$, which is a regular expression, therefore it is regular. $L_{iii} = ab$ since the $\{a^n b^n \mid n \geq 1\}$ intersects $\{(ab)^n \mid n \geq 1\}$ only at ab. This is a regular expression, therefore L_{iii} is regular. $L_{iv} = a(a^2)^*$ which is a regular expression, therefore it is regular. And finally $L_v = a^d(a^k)^*$ which is a regular expression, therefore it is regular. $\square$

Part 1.2

Consider the language $L_1 = \{a^{2n} b^n \mid n \geq 1\}$. We show that L_1 is not regular, but it is context free.

Proof. Clearly, if L_1 was regular it would have a pumping length, p. Thus assume L_1 is regular, and let p denote its pumping length as given by the pumping lemma. Consider then the string $w = a^{2p} b^p \in L_1$. According to the pumping lemma, we can split w in $w = xyz$ such that $|xy| \leq p$ and $|y| > 0$, and such that $xy^i z \in L_1$ for any $i \geq 0$. Clearly in this case we have $y = a^k$ for some $0 < k \leq p$ and thus $xy^2 z = a^{2p+k} y^p$ which is clearly not in L_1 contradicting

the pumping lemma. Hence, L_1 is *not* regular.
That L_1 *is* a CFL, can be seen by the fact that it is generated by the CFG

$$\begin{aligned} S &\to a^2Xb, \\ X &\to a^2Xb \mid \epsilon. \end{aligned} \tag{0.2}$$

This concludes the proof. □

1.3

Let L be an arbitrary language that does not contain the empty string, ϵ. Define

$$OTHER(L) = \{x_1x_2x_3...x_k \mid x_1 \in L,\ x_2 \notin L,\ x_3 \in L,\ x_4 \notin L, ...x_k \in L,\ k \in \mathbb{N}_{\text{odd}}\}, \tag{0.3}$$

where $\mathbb{N}_{\text{odd}}$ are the odd integers. We show that if L is regular, then $OTHER(L)$ is regular. We also show that if L is in NL then $OTHER(L)$ is in NL.
Notice there is something not well defined in this question, since in the explanation in the problem it is said that if $L = \{aa\}$ then $aaaaaa \notin OTHER(L)$, but clearly

$$aaaaaa = \underbrace{aa}_{\in L}\underbrace{\epsilon}_{\notin L}\underbrace{aa}_{\in L}\underbrace{\epsilon}_{\notin L}\underbrace{aa}_{\in L} \in OTHER(L).$$

Thus if this explanation is true, we must redefine $OTHER(L)$ under the assumption that x_i are all not the empty string in the definition of $OTHER(L)$, in which case $aaaaaa \notin L$.

Proof. If L is regular, then there is a regular expression, R such that $L(R) = L$. But then $OTHER(L) = L(R(\overline{R}R)^*)$, where $\overline{R}$ denotes the regular expression for $\overline{L} \setminus \{\epsilon\} = \overline{L \cup \{\epsilon\}}$ (regular languages are closed under complementation). $R(\overline{R}R)^*$ is a regular expression, and therefore $OTHER(L)$ is regular.
If L is in NL, then there is a log-space TM, M, that desides it. By theorem 8.27 we know that coNL=NL and therefore there is also a log-space TM, $\overline{M}$, that desides $\overline{L}$. Consider now the log-space TM M_{OTHER} =”On input $w = w_1....w_n$

1. Store the length of the input, n, on the worktape. If $n = 0$ *reject.*
2. Non-deterministically choose $i = 1$ to n, and run M on $w_1...w_i$. If $i = n$ and M accepts, *accept.* If $i < n$ and M rejects, skip step 3.
3. Non-deterministically choose $j = i+1$ to n, and run M on the $w_j...w_n$ and $\overline{M_{OTHER}}$ on $w_{i+1}...w_j$. If for some j M and $\overline{M_{OTHER}}$ accept, *accept.*

where $\overline{M_{OTHER}}$ is the same as M_{OTHER} but with all M replaced by $\overline{M}$ and with $\overline{M_{OTHER}}$ in step 3 replaced by M_{OTHER} and also with new counters $\bar{i}$ and $\bar{j}$. Thus M_{OTHER} calls $\overline{M_{OTHER}}$ and vice versa. Hence this uses the recursion theorem. Evidently, this machine recursively check that the input w can be split in $w = x_1w_1z_1$ and further that $w_i = x_{i+1}w_{i+1}z_{i+1}$ where

$x_i, z_i \in L$ for i odd and $x_i, z_i \notin L$ for i even, until some point where $w_n \in L$ and the machine accepts. If this is not the case then clearly $w \notin OTHER(L)$. Clearly, M_{OTHER} stores only the four counters counters $i, j, \bar{i}, \bar{j}$ which are alternatingly overwritten during the recursion, and simulates log-space TMs on smaller inputs than the original input, and therefore, M_{OTHER} runs itself in log-space.
Alternatively, we might reduce to PATH which is known to be in NL. This done by the following log-space transducer, T="On input $w = w_1...w_n$

1. Produce graph G which have two vertices for each $i = 1, ..., n$, v_i, $\tilde{v}_i$ on the output tape.
2. Add one edge $(v_i, \tilde{v_j})$ for every $i < j$ such that $w_{i+1}...w_j \in L$ and one edge $(\tilde{v}_i, v_j)$ for every $i < j$ such that $w_i + 1...w_j \notin L$ (i.e. $w_i...w_j \in \overline{L}$) .
3. Add vertices s, t and edges $(s, \tilde{v_j})$ if $w_1...w_j \in L$ and (v_j, t) if $w_j...w_n \in L$.

Clearly G has a directed path from s to t if and only if $w = x_1 x_2 ....x_k$ such that $x_1 \in L$, $x_2 \notin L$,...,$x_k \in L$. Furthermore, T is a log-space transducer as all step requires only G to store simple counters. Thus we see that we have constructed a log-space tranducer such that $T(w) \in PATH$ if and only if $w \in OTHER(L)$. Hence $OTHER(L) \leq_L PATH$, and since $PATH$ is in NL, we conclude that also $OTHER(L)$ is in NL. □

Question 2

Part 2.1

Let

$$MAXCELL_{TM} = \{\langle M, w, k\rangle \mid M \text{ is a determinitstic TM, } w \text{ is a string, and } k \in \mathbb{N}_0 \text{ such that for every tape cell } c \text{ of } M,\ M \text{ writes to } c \text{ at most } k \text{ times during its execution on input } w\}.$$

We show that there is a constant $k \in \mathbb{N}_0$ and a TM, N, which on input $\langle M, w\rangle$ simulates M on w such that N writes to every tape cell at most k times. Simply construct the TM N as the universal turing machine, but with the extra criterion, that it marks the last character on the tape. Then every time it simulates a single step of the computation of M on w, it copies the entire tape, to the right of the marked character on the tape, and updates it simultaneously according to the single computation step. Clearly this TM uses a lot of space, but writes at most once to every cell.

Part 2.2

We now show that $MAXCELL_{TM}$ is not Turing recognizable.

Proof. This follows if we can reduce $\overline{A_{TM}}$ to $MAXCELL_{TM}$, as it is known that $\overline{A_{TM}}$ is not Turing recognizable. Consider the following reduction. Given $\langle M, w\rangle$, construct $\langle \tilde{N}, \langle M, w\rangle, 1\rangle$, where $\tilde{N}$ is the TM from part 2.1 but with the additional property, that it overwrites the entire tape with blanks if it reaches the accept state of M. Thus if M accepts w $\langle \tilde{N}, \langle M, w \rangle, 1 \rangle \notin MAXCELL_{TM}$ and if M does not accept w, $\langle \tilde{N}, \langle M, w \rangle, 1 \rangle \in MAXCELL_{TM}$. Clearly, there exist a TM that given input $\langle M, w \rangle$ halts with $\langle \tilde{N}, \langle M, w \rangle, 1 \rangle$ on the tape. Hence the construction above constitutes a Turing reduction from $\overline{A_{TM}}$ to $MAXCELL_{TM}$ and we have $\overline{A_{TM}} \leq_T MAXCELL_{TM}$, from which it follows that $MAXCELL_{TM}$ is not Turing recognizable. □

Part 2.3

Let $MAXCELL_{LBA}$ be like $MAXCELL_{TM}$ with with TM M replaces by LBA M. We show that $MAXCELL_{LBA}$ is decidable

Proof. This follows easily by noticing that an LBA has only just enough tape to contain the input. Thus we may design a TM, N, that simulates LBA, M on the input w and simultaneously keeps a count of how many times M writes to any tape slot. Thus consider the decider $D =$" On input $\langle M, w, k \rangle$,

1. Run N on $\langle M, w \rangle$ one step at a time, and update counters described above at each computation step.
2. If a counter exeeds k, *reject*.
3. If N halts, and all counters are below k, *accept*.

Clearly this is a decider, since if w has lenght n there can be at most nk computation steps before a counter exeeds k. Furthermore, we see by straightforward inspection that it this decider exactly accepts $MAXCELL_{LBA}$. □

Question 3

Part 3.1

We formulate $WEAK - HAMPATH$ as the laguage

$$k - WEAK - HAMPATH = \{\langle G, s, t \rangle \mid G \text{ is a directed graph, and there exist a simple path } \gamma \text{ in } G \text{ from } s \text{ to } t \text{ such that } |\gamma| \geq |V(G)| - k\}$$

where for a path, γ, we denote by $|\gamma|$ the number of vertices the path visits. Next we show that $k - WEAK - HAMPATH$ is in NP

Proof. Consider the polynomial time verifier $V =$"On input $(\langle G, s, t \rangle, c)$

1. Check that G is a directed graph.
2. Check that c is a simple path in G that starts at s and ends at t.

3. Check that the number of vertices in c is greater than or equal to $|V(G)| - k$

Each step requires trivially only polynomial time. Furhermore, it is evident that $\langle G, s, t \rangle$ is in $k - WEAK - HAMPATH$ if and ony if there exist a c such that $(\langle G, s, t \rangle, c)$ is accepted by V. This shows that $k - WEAK - HAMPATH$ is verified in polynomial time, and we conclude that $k - WEAK - HAMPATH$ is in NP. □

Part 3.2

We show that $k - WEAK - HAMPATH$ is NP-complete by reducing from $HAMPATH$.

Proof. Consider the following reduction: Given $\langle G, s, t \rangle$ produce $\langle \tilde{G}, s, t \rangle$ where $\tilde{G}$ is just G with k new vertices that are not connected to anything. Clearly if G has a Hamiltonian path from s to t then $\langle \tilde{G}, s, t \rangle$ is in $k - WEAK - HAMPATH$, and if $\langle \tilde{G}, s, t \rangle$ is in $k - WEAK - HAMPATH$ then G has a Hamiltonian path from s to t, as no path can visit the extra vertices. The reduction is obviously polynomial time, since it only adds a finite number of vertices to the existing data. Thus we have shown that $HAMPATH \leq_T k - WEAK - HAMPATH$. Since $HAMPATH$ is NP-complete, we conclude that $k - WEAK - HAMPATH$ is NP-hard, and it follows by Part 3.1 that $k - WEAK - HAMPATH$ is NP-complete. □

Part 3.3

We now consider $FULLPATH$, which can be formulated as laguage

$$\begin{aligned} FULLPATH = \{\langle G, s, t \rangle \mid\ & G \text{ is a directed graph and there exist a path, } \gamma, \\ & \text{from } s \text{ to } t \text{ such that } |\gamma| = |V(G)|\}. \end{aligned} \tag{0.4}$$

Part 3.4

We show that $FULLPATH$ is in P.

Proof. We use the hint in the problem at let M be the TM that in polynomial time, on input H, where H is a directed graph, computes a partition $(V_1, ..., V_k)$ of the vertices of H such that $H(V_i)$ is strongly connected for each i, and in H there are edges from V_i to V_j if and only if $i \leq j$. A graph H has a path from vertices s to t if and only if $s \in V_1$, $t \in V_k$, and there are edges from V_i to V_{i+1} for each $i < k$. On one hand if these conditions are satisfied we can construct the path from s to t by going s to all vertices in V_1 and then to V_2 and around all vertices in V_2 and so on, untill we end at V_k in which case we visit all vertices in V_k ending up at t. On the contrary if there is no edge from V_i to V_{i+1} for some i, then we either not get past V_i, since we cannot go back to V_l for $l < i$, or we cannot visit V_i at all, if we have already skipped it, again since we can not go back from V_j with $j > i$. Thus we construct the following TM, $M_1 =$"On input $\langle G, s, t \rangle$

1. Run M on G.

2. Check that $s \in V_1$ and $t \in V_k$ (where V_k is the last element in the partition produced by M) if not, *reject.*

3. For $i = 1$ to n choose a vertex $v_i \in V_i$ and run the TM that desides $PATH$ in polynomial time, on $\langle G, v_i, v_{i+1}\rangle$, if yes for all i, *accept*, else, *reject.*

Since V_i is strongly connected for all i, any vertex $v_i \in V_i$ suffices in step 3. □

Question 4

Consider the laguage

$$\begin{aligned}\text{NO-FIERY-DEATH} = \{\langle G, M, D\rangle \mid\ & G = (V, E) \text{ is a directed graph,}\\ & M \subseteq V,\\ & D \subseteq V,\\ & \text{there is no path from any vertex in } M \text{ to any vertex in } D\}\end{aligned}$$

We show that NO-FIERY-DEATH is NL-complete.

Proof. We show that NO-FIERY-DEATH is in NL by reducing to $\overline{PATH}$ which is in NL by Theorem 8.27. Consider the following log-space transducer, $T =$"On input $\langle G, M, D\rangle$ Construct the graph $\tilde{G}$ by

1. For each pair $(u, v) \in M \times D$ add a copy of G, call the copy $G^{u,v}$.

2. Add vertex s and for each $u \in M$ the line $(s, u^{(u,v)})$ where $u^{(u,v)}$ denotes the vertex u in $G^{(u,v)}$.

3. Add vertex t and for each $v \in D$ the line $(v^{(u,v)}, t)$.

Clearly, we see that if there is a path from a vertex $u_0 \in M$ to $v_0 \in D$ then there is a path from $s \to u_0^{(u_0,v_0)} \to ... \to v_0^{(u_0,v_0)} \to t$ in $\tilde{G}$, so $\langle \tilde{G}, s, t\rangle \in PATH$, on the other hand if there is no path from any $u \in M$ to any $v \in D$, then there is also no path from s to t, as such a path would be bound to go from some vertex in M to some vertex D along the way. Therefore, $\langle G, M, D\rangle \in$ NO-FIERY-DEATH if and only if $\langle \tilde{G}, s, t\rangle \in \overline{PATH}$. Thus T shows that NO-FIERY-DEATH $\leq_L \overline{PATH}$, and it follows that NO-FIERY-DEATH is in NL.
That NO-FIERY-DEATH is NL-hard follows by the completely trivial reduction $\overline{PATH} \leq_L$ NO-FIERY-DEATH. This follows by viewing $\overline{PATH}$ as being a subset of NO-FIERY-DEATH by restricting to cases where $M = \{s\}$ and $D = \{t\}$ are singeltons. This restriction can clearly be computed by the log-space tranducer, $T' =$On input $\langle G, s, t\rangle$ output $\langle G, \{s\}, \{t\}\rangle$. Thus we need only notice that $\overline{PATH}$ is NL-complete, which follows by Theorem 8.27, saying that NL=coNL. Thus for any language A in NL, we have $\overline{A} \leq_L PATH$, by PATH being NL-hard. But then since complementation, and noting that the definition of $\leq_L$ is symmetric under complementation, we notice that $A \leq_L \overline{PATH}$. Since A was any language in NL, we see that $\overline{PATH}$ is NL-hard. The

reduction $\overline{PATH} \leq_L$ NO-FIERY-DEATH hence shows that NO-FIERY-DEATH is NL-hard, and by the above it follows that it is NL-complete. $\square$

Part 4.2

We consider now the language

$$\text{NO-FIERY-DEATH-WITHOUT-ROD} = \{\langle G, M, D, R\rangle \mid \begin{array}{l} G = (V, E) \text{ is a directed graph,} \\ M \subseteq V,\ D \subseteq V,\ R \subseteq V, \\ \text{If there is a path from a vertex in } M \\ \text{to a vertex in } D, \text{ then it contains} \\ \text{a vertex in } R\} \end{array}$$

We show that NO-FIERY-DEATH-WITHOUT-ROD is NL-complete.

Proof. We notice that if $\langle G, D, M, R\rangle \in$ NO-FIERY-DEATH-WITHOUT-ROD if and only if either $\langle G, D, M\rangle \in$ NO-FIERY-DEATH, or $\langle G, D, R\rangle \in$ NO-FIERY-DEATH, or $\langle G, R, M\rangle \in$ NO-FIERY-DEATH. In the following we let N be the log-space NTM that desides NO-FIERY-DEATH, whose existence was shown in Part 4.1. Thus we may construct the log-space NTM $N_1 =$"On input $\langle G, M, D, R\rangle$

1. Non-deterministically run N on $\langle G, M, D\rangle$, $\langle G, M, R\rangle$, and $\langle G, R, D\rangle$. If one of them accepts, *accept*.
2. *Reject*.

Clearly, by the above observation, this NTM accept, NO-FIERY-DEATH-WITHOUT-ROD and runs in logarithmic space. Hence NO-FIERY-DEATH-WITHOUT-ROD is in NL. On the other hand we may easily log-space reduce, NO-FIERY-DEATH to NO-FIERY-DEATH-WITHOUT-ROD by the following tranducer, $T_2 =$"On input $\langle G, M, D\rangle$

1. Add vertex $\dot{v}$ to G (call the new graph $\dot{G}$).
2. Write to output tape $\langle \dot{G}, M, D, \{\dot{v}\}\rangle$.

Clearly if $\langle G, M, D\rangle$ is in NO-FIERY-DEATH then $\langle G, M, D, \{\dot{v}\}\rangle$ is in NO-FIERY-DEATH-WITHOUT-ROD, on the other hand, since $\dot{v}$ is disconnected from all other vertices in $\dot{G}$ we see that no path can pass through $\dot{v}$. Hence if $\langle \dot{G}, M, D, \{\dot{v}\}\rangle$ is in NO-FIERY-DEATH-WITHOUT-ROD we clearly must have $\langle G, M, D\rangle \in$ NO-FIERY-DEATH. Therefore, it follows that NO-FIERY-DEATH $\leq_L$ NO-FIERY-DEATH-WITHOUT-ROD, which show that NO-FIERY-DEATH-WITHOUT-ROD is NL-hard, and it follows that its is NL-complete. $\square$