CoCo - Assignment 5

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

Dated: March 21, 2021

8.20

We show that $2SAT$ is NL complete.

Proof. First we show that $2SAT \in$ NL. This follows if we can log-space reduce $2SAT$ to $PATH$ since $PATH \in$ NL. Consider the following construction. Let ϕ be a $2cnf$-formula, and construct the graph G_ϕ consisting of vertices v_x for every literal x. Notice that x and $\bar{x}$ are both literals, which are considered independt when assigning vertices. Now for each clause $(x \vee y)$ in ϕ construct directed the lines $(v_{\overline{x}}, v_y)$ and $(v_{\overline{y}}, v_x)$, where (v_1, v_2) goes from v_1 to v_2. Evidently, the formula ϕ is satisfiable if and only if for all literals x there is not both a path from v_x to $v_{\overline{x}}$ and one from $v_{\overline{x}}$ to v_x. Clearly, if there is a path from v_x to $v_{\overline{x}}$ there are clauses $(\overline{x} \vee y_1) \wedge (\overline{y_1} \vee y_2) \wedge (\overline{y_2} \vee y_3) \wedge ... \wedge (\overline{y_n} \vee \overline{x})$ in ϕ and choosing x true, we see that y_i must be true for all $i = 1$ to n, but then $\overline{x}$ must be true in the last clause, which is a contradiction, thus we must choose x to be false. But if there is a path from $v_{\overline{x}}$ to v_x the same reasoning implies that x must be chosen true, which is a contradiction and ϕ is thus not satisfiable. On the contrary, if there are no paths from v_x to $v_{\overline{x}}$, or the other way around, for any literal x, in G_ϕ, we may color all vertices of literals that are connected to some v_x blue and all literals that are connected to $v_{\overline{x}}$ red, We continue this process for each literal whose vertex has not yet been colored, until all vertices are colored. Now assign 1 to all literals whose vertex is blue and 0 to all literals whose vertex is red. Then ϕ is satisfied, this is clear since for each clause $(x \vee y)$ there is a line $(v_{\overline{x}}, v_y)$ and thus both $v_{\overline{x}}$ and v_y are the same color. If they are red $\overline{x} = 0$ making $x = 1$ and the clause is true. If they are blue, $y = 1$ making the clause true. Consider therefore the log-space transducer, $T =$" On input $\langle \phi \rangle$ where ϕ is a $2cnf$-formula

1. Construct G_ϕ on the output tape.
2. construct all pairs $\{v_x, v_{\overline{x}}\}$ for all literals, x, on the output tape.

Notice that constructing G_ϕ requires only to store one clause at a time on the working tape, and thus this requires only $O(\log(n))$ space. This tranducer have not reduced $2SAT$ to $PATH$, but rather to a modifed $PATH$-problem with laguage

$$\overline{mPATH} = \{\langle G, \{s_1, t_1\}, ..., \{s_n, t_n\}\rangle \mid G \text{ is a directed graph and } s_i, t_i \text{ are vertices such that there does not exist paths in } G \text{ both from } s_i \text{ to } t_i \text{ and from } t_i \text{ to } s_i \text{ for any } i = 1 \text{ to } n\}.$$

Clearly $\overline{mPATH}$ is in NL, as it can be decided by applying the log-space NTM (proof of Theorem 8.27) that desides $\overline{PATH}$, $2n$ times. Since, as argued above, $\langle G_\phi, v_x, v_{\overline{x}}\rangle \in \overline{PATH}$ and $\langle G_\phi, v_{\overline{x}}, v_x\rangle \in \overline{PATH}$ for all literals x in ϕ if and only if ϕ is satisfiable, we see that $\langle G, \{x_1, \overline{x_1}\}, ..., \{x_n, \overline{x_n}\}\rangle$ is in $\overline{mPATH}$ if and only if ϕ is satisfiable. Thus T constitues a log-space reduction from $2SAT$ to $\overline{mPATH}$, showing that $2SAT$ is in NL.
Alternatively we may reduce directly to $\overline{PATH}$ by constructing instead the graph $\tilde{G}_\phi$ consiting of vertices s and t and two copies of G_ϕ for each Boolean variable, x, call the copy corresponding to x G_x^1 and G_x^2. We connect the G_x^1 and G_x^2 for each x by making a line from $\overline{x}$ in G_x^1 to $\overline{x}$ in G_x^2. We then add lines from s to x in G_x^1 for all Boolean variables x. Similarly we add lines from x in G_x^2 to t. Then we clearly see that a path from s to t in $\tilde{G}_\phi$ is equivalent to having a path from x to $\overline{x}$ and then from $\overline{x}$ to x for some Boolean variable x, which is again equivalent to ϕ not being satisfiable.
Next we show that $2SAT$ is NL-hard, which follows if we can log-space reduced $PATH$ to $2SAT$. Consider the following construction. Given a directed graph, G, and two vertices $s, t \in V(G)$, we construct for each vertex a a literal x_a and for each edge (a, b) from a to b the clause $(\overline{x_a} \vee x_b)$. We then construct the formula

$$\phi_G = (x_s \vee x_s) \wedge (\overline{x_s} \vee \overline{x_t}) \wedge \left(\bigwedge_{(a,b)\in E(G)} (\overline{x_a} \vee x_b) \right).$$

If there is a path from s to t in G, $s \to a_1 \to a_2 \to ... \to t$, ϕ_G contains the expression

$$(x_s, x_s) \wedge (\overline{x_s} \vee \overline{x_t}) \wedge (\overline{x_s}, x_{a_1}) \wedge (\overline{x_{a_1}}, x_{a_2}) \wedge ... \wedge (\overline{x_{a_n}}, x_t).$$

The first clause says that we have to choose $x_s = 1$ the second one says that then $x_t = 0$. We see that ϕ_G is *not* satisfiable, since x_{a_i} are forced to be equal to 1 and by the last clause we must have $x_t = 1$ which contradicts the second clause.
On the contrary if there is no path from s to t, we may satisfy ϕ_G by just assigning $x_v = 1$ for all $v \in V(G)$ that can be reached by a path from s and $x_w = 0$ for all $w \in v(G)$ that cannot be reached from s. Clearly we have,

$$\phi_G = (x_s \wedge x_s) \wedge (\overline{x_s} \vee \overline{x_t}) \wedge \left(\bigwedge_{\substack{(a,b)\in E(G) \\ a \text{ can be reached from } s}} (\overline{x_a} \wedge x_b) \right) \wedge \left(\bigwedge_{\substack{(a,b)\in E(G) \\ a \text{ cannot be reached from } s}} (\overline{x_a} \wedge x_b) \right).$$

The first two clauses are obviously satisfied with this assignment, furthermore, if a can be reached from s and ϕ_G contain the clause $(\overline{x_a} \vee x_b)$, then x_b can be reached from s and the clause is true. If a cannot be reached from s, $\overline{x_a} = 1$ and the clause $(\overline{x_a}, x_b)$ is true. Thus ϕ_G is satisfied. Clearly, constructing ϕ_G can be done in log-space, as it requires only a transducer to store one edge at a time while typing a clause to the output tape. Thus we have the following log-space transducer $T =$"On input $\langle G, s, t\rangle$

1. Write ϕ_G to the output tape, by constructning one clause at a time, and erasing the worktape inbetween each step."

Since $\phi_G \in \overline{2SAT}$ if and only if $\langle G, s, t\rangle \in PATH$, we see that this shows $PATH \leq_L \overline{2SAT}$, and since the definition of log-space reducibility if invariant under complementation of the involved langauges, we see that $\overline{PATH} \leq_L 2SAT$. Now from theorem 8.27 we know that $\overline{PATH}$ is in NL, that it is also NL-hard follows again from the fact that $A \leq_L PATH$ if and only if $\overline{A} \leq_L \overline{PATH}$, and from Theorem 8.27 saying NL = coNL. To see this notice that any language in, A, in NL, satisfies $\overline{A} \leq_L PATH$ by $PATH$ begin NL-hard, but then $A \leq_L \overline{PATH}$ showing that $\overline{PATH}$ is NL-hard. Thus, since we showed above that $\overline{PATH} \leq_L 2SAT$ we see that also $2SAT$ is NL-hard, which concludes the proof. □

8.20

We show that $2SAT$ is NL complete.

Proof. First we show that $2SAT \in$ NL. This follows if we can log-space reduce $2SAT$ to $PATH$ since $PATH \in$ NL. Consider the following construction. Let ϕ be a $2cnf$-formula, and construct the graph G_ϕ consisting of vertices v_x for every literal x. Notice that x and $\bar{x}$ are both literals, which are considered independt when assigning vertices. Now for each clause $(x \vee y)$ in ϕ construct directed the lines $(v_{\overline{x}}, v_y)$ and $(v_{\overline{y}}, v_x)$, where (v_1, v_2) goes from v_1 to v_2. Evidently, the formula ϕ is satisfiable if and only if for all literals x there is no path from v_x to $v_{\overline{x}}$. Clearly, if there is a path from v_x to $v_{\overline{x}}$ there are clauses $(\overline{x} \vee y_1) \wedge (\overline{y_1} \vee y_2) \wedge (\overline{y_2} \vee y_3) \wedge ... \wedge (\overline{y_n} \vee \overline{x})$ in ϕ and choosing x true, we see that y_i must be true for all $i = 1$ to n, but then $\overline{x}$ must be true in the last clause, which is a contradiction and ϕ is thus not satisfiable. On the contrary, if there are no paths from v_x to $v_{\overline{x}}$ for any literal x, in G_ϕ, we may color all vertices of literals that are connected to some v_x blue and all literals that are connected to $v_{\overline{x}}$ red, We continue this process for each literal whose vertex has not yet been colored, until all vertices are colored. Now assign 1 to all literals whose vertex is blue and 0 to all literals whose vertex is red. Then ϕ is satisfied, this is clear since for each clause $(x \vee y)$ there is a line $(v_{\overline{x}}, v_y)$ and thus both $v_{\overline{x}}$ and v_y are the same color. If they are red $\overline{x} = 0$ making $x = 1$ and the clause is true. If they are blue, $y = 1$ making the clause true. Consider therefore the log-space transducer, $T =$" On input $\langle\phi\rangle$ where ϕ is a $2cnf$-formula

1. Construct G_ϕ on the output tape.

2. construct all pairs $\{v_x, v_{\overline{x}}\}$ for all literals, x, on the output tape.

Notice that constructing G_ϕ requires only to store one clause at a time on the working tape, and thus this requires only $O(\log(n))$ space. This tranducer have not reduced $2SAT$ to $PATH$, but rather to a modifed $PATH$-problem with laguage

$$mPATH = \{\langle G, \{s_1, t_1\}, ..., \{s_n, t_n\}\rangle \mid G \text{ is a directed graph and } s_i, t_i \text{ are vertices such that there exist paths in } G \text{ from } s_i \text{ to } t_i \text{ for all } i = 1 \text{ to } n\}.$$

Clearly $mPATH$ is in NL, as it can be decided by applying the log-space NTM from example 8.19 that desides path, n times. Since, as argued above, $\langle G_\phi, v_x, v_{\overline{x}}\rangle \in PATH$ for all literals x in ϕ if and only if ϕ is satisfiable, we see that $\langle G, \{x_1, \overline{x_1}\}, ..., \{x_n, \overline{x_n}\}\rangle$ is in $mPATH$ if and only if ϕ is satisfiable. Thus T constitues a log-space reduction from $2SAT$ to $mPATH$, showing that $2SAT$ is in NL.

Next we show that $2SAT$ is NL-hard, which follows if we can log-space reduced $PATH$ to $2SAT$. Consider the following construction. Given a directed graph, G, and two vertices $s, t \in V(G)$, we construct for each vertex a a literal x_a line (a, b) from a to b the clause $(\overline{x_a}, x_b)$. We then construct the formula

$$\phi_G = (\overline{x_s}, \overline{x_t}) \wedge (x_s, x_t) \wedge \left(\bigwedge_{(a,b)\in E(G)} ((\overline{x_a}, x_b) \wedge (\overline{x_b}, x_a)) \right).$$

If there is a path from s to t in G,$s \to a_1 \to a_2 \to ... \to t$,ϕ_G contains the expression

$$(\overline{x_s}, \overline{x_t}) \wedge (x_s, x_t) \wedge (\overline{x_s}, x_{a_1}) \wedge (\overline{x_{a_1}}, x_s) \wedge (\overline{x_{a_1}}, x_{a_2}) \wedge (\overline{x_{a_2}}, x_{a_1}) \wedge ... \wedge (\overline{x_{a_n}}, x_t) \wedge (\overline{x_t}, x_{a_n}).$$

The first two clauses says that if we choose $x_s = 1$ then $x_t = 0$ and vice versa. We see that ϕ_G is *not* satisfiable, since x_{a_i} are forced to be equal to

and by choosing $x_{a_i} = 1$ for $i = 1, ..., n$ we see that this expression is satisfied. Thus we see that for all vertices connected to s and t, if we choose their corresponding literals to be 1, and then choose the rest to be 0. ϕ_G is satisfied. This can be seen by noticing that

$$\phi_G = (x_s \wedge x_s)(\overline{x_s}, x_t) \wedge \left(\bigwedge_{\substack{(a,b)\in E(G) \\ a \text{ is connected to s}}} (\overline{x_a}, x_b) \right) \wedge \left(\bigwedge_{\substack{(a,b)\in E(G) \\ a \text{ is not connected to s}}} (\overline{x_a}, x_b) \right).$$

Clearly, since x_a is chosen true if a is connected to s, $\left(\bigwedge_{\substack{(a,b)\in E(G) \\ a \text{ is connected to s}}} (\overline{x_a}, x_b) \right)$ is true, since each clause contain exactly one false and one true literal. Similarly $\left(\bigwedge_{\substack{(a,b)\in E(G) \\ a \text{ is not connected to s}}} (\overline{x_a}, x_b) \right)$ is true since it cotains one true and one false literal, and lastly $(\overline{x_s}, x_t)$ is true since it cotains one false and one true literal.

On the contrary if there is no path from s to t, ϕ_G cannot be satisfied, since $\left(\bigwedge_{\substack{(a,b)\in E(G) \\ a \text{ is connected to s}}} (\overline{x_a}, x_b) \right)$ forces all literals corresponding to vertices connect to s to be equal..... All literals true satifies ϕ_G???

Clearly if two vertices, y, z are connected outside the part of G connected to s and t, ϕ_G contain clause ()

On input $\langle G, s, t\rangle$ we construct the $2cnf$-formula ϕ, with one varibles x_i for each vertex, v_i. Now construct the formula consiting of $(x_s \vee x_i)$ for each v_i that is connected to s, $(x_s \vee x_i)$ □

8.24

We show that $EQ_{REX} = \{\langle R, S\rangle \mid R \text{ and } S \text{ are equivalent regular expressions}\}$ is in PSPACE.

Proof. Consider the non-deterministic polynomial space TM $M =$"On input $\langle R, S\rangle$

1. Check that R and S are regular expressions, if not *reject*.

2. Construct the NFA, N, that accepts $L(R)\Delta L(S) = L(R)\cap L(S)^{\complement}\cup L(S)\cap L(R)^{\complement}$. (This can be done in polynomial space by following the proofs that DFAs and regular expressions are equivalent).

3. Non-determinstically construct all string with length less than or equal to the number of states in N, and simulate N on them. If for some string, N accepts, *reject*, else *accept*.

Clearly if N in the above accepts a string, it must accept at least one string of length less than or equal to the number of states in N since if N visits any state twice we can cut away the part of the string that loops N, thus resulting in a string that makes N visit every state at most once. Therefore, M accepts if and only if $L(R)\Delta L(S) = \emptyset$ or equivalently $L(R) = L(S)$. We notice also that simulating N on a string, uses no space except for the space to store N and the input, since N has no memory. This shows that EQ_{REX} is in PSPACE. □

8.27

We show if every NP-hard language is PSPACE-hard, then NP=PSPACE.

Proof. This follows since, any NP-complete language, say $HAMPATH$, is in PSPACE, by NP$\subset$PSPACE. Therefore, assuming that every NP-hard language is PSPACE-hard, $HAMPATH$ is also PSPACE-hard. Thus we may reduce any PSPACE language, A, to $HAMPATH$ in polynomial time, followed by a polynomial time verfier for $HAMPATH$, this all in all would constitute a polynomial time verifier for A, showing that $A \in$ NP for all $A \in$ PSPACE, or equivalently PSPACE$\subset$NP. This shows that PSPACE= NP □

8.34

Let B be the language of properly nested parenthesis and brackets. We showt that B is in L.

Proof. Simply consider the log-space TM, $M =$"On input B where the alphabet consists of $\{(,),[,]\}$

1. Check that the first input symbol is not a close parethesis or bracket, if it is *reject*.
2. For $i = 1$ to n, where n is input length.
3. If the i^{th} input symbol is an open bracket or parethesis:
   (i) Save the i^{th} input symbol to the work tape .
   (ii) Starting with the i^{th} symbol, scan across the input from left to rigth keeping count on the work tape of how many *open*s, $\{(, [\}$ and *close*s $\{),]\}$ are met, if the two numbers are ever equal, check that the i^{th} input symbol and the last symbol scanned form a proper pair, *i.e.* () or [] if not *reject*, if yes go to 4.
4. Clear the working tape for everything except n, and continue with the loop.
5. *Accept*

Clearly this TM uses only log space, as it only needs to store the input length (in binary), any input character, and the two counters for open and close. Each of which can be stored in $O(\log(n))$ space. It is also clear that this TM checks that each open parenthesis or bracket has a proper pairing at the correct position, and it can get to its accepts state, if and only if this is true. This shows that B is in L. $\square$

Exam 2017, Question 3

Part 3.1

Fix $r > 0$, the

$$r - NEIGHBORHOOD = \{\langle G, k\rangle \mid \text{There exist } V' \subset V(G), \text{ such that } \left|V'\right| \leq k$$
$$\text{and for any edge } (u, v) \in E(G), \text{ we have } d_G(\{u, v\}, V') \leq r\},$$

where we define the distance for subsets of vertices, with a sligth abuse of notation, by $d_G(V_1, V_2) := \inf_{u\in V_1,\ v\in V_2} d_G(u, v)$, where the righthand side, $d_G(u, v)$, is the shortest distance, in number of edges, between u and v.
We now show that $r - NEIGHBORHOOD$ is in NP.

Proof. We construct a polynomial time verifier, that verifies $r-NEIGHBORHOOD$. Consider $V =$" On input $(\langle G, k\rangle, c)$

1. Check that c is a subset of $V(G)$ such that $|c| \leq k$, if not *reject*.
2. For each vertex, v, in G:
   (a) Construct all paths $P_{v,i}$ of length r starting at v (there are at most n^r where $g = |V(G)|$).
   (b) Check if any of the paths $P_{v,i}$ intersect c. If not *reject*.

3. *Accept.*

Clearly if c is an r-neighborhood cover of G of size at most k then V accepts $(\langle G, k\rangle, c)$, on the other hand, if it is not, then clearly V rejects along the way. Clearly step 1 is polynomial in time. Step 2a is polynomial since there are at most n^r possible paths of length r starting at each vertex, so they are constructed in polynomial time. Step 2b is polynomial in time, since there are at most n^r paths, each with $r+1$ vertices so at most $(r+1)n^r$ vertices to check. And since step 2 runs on all vertices but this gives at most a factor n on top of the polynomial runtime of step a and b, it runs in polynomial time. This shows $r - NEIGHBORHOOD \in$NP. □

Part 3.2

We again fix $r \geq 0$. Then $r - NEIGHBORHOOD$ is in PSPACE, since by part 3.1 it is in NP, and we now that NP $\subset$ NPSPACE = PSPACE, where the first inclusion follows by the fact that a polynomial time NTM does not have time to use more than polynomial space, and the equality by Savitch's theorem.

Part 3.3

We show that $2 - NEIGHBORHOOD$ is NP-complete.

Proof. We do this by reducing from $VERTEX - COVER$. Consider the polynomial time computable function, computed by TM $T =$"On input $\langle G, k\rangle$

1. Construct $\tilde{G}$ by: For each edge $(u, v) \in G$ construct a vertices $v^1_{(u,v)}$ and $v^1_{(u,v)}$ and replace the edge (u, v) with the two edges $(u, v^1_{(u,v)})$, $(v_{(u,v)}, v^2_{(u,v)})$ and $(v^2_{(u,v)}, v)$.

2. Output $\langle \tilde{G}, k\rangle$

Clearly if $\langle G, k\rangle \in VERTEX - COVER$ there exist a subset of vertices V' such that $|V'| = k$ and all edges has distance 0 to V'. Thus V' constitutes a $2 - NEIGHBORHOOD$ cover of $\tilde{G}$. On the other hand if $\tilde{G}$ has a $2 - NEIGHBORHOOD$ cover $\tilde{V}'$ we may merge $v^1_{(u,v)}$ with u and $v^2_{(u,v)}$ with v in $\tilde{G}$ for all edges (u, v) in G. Then $\tilde{V}'$ becomes a subset V' of $V(G)$ after the merging, and this subset clearly constitutes a vertex-cover, of a possibly lower degree than k, since the distance from $\tilde{V}$ to any edge was at most 2 before the merging, and thus after merging the distance between any of the leftover vertices are decreased by at least 2. Furhtermore, it is clear if G has an l-vertex cover for $l < k$ then G also have a k-vertex cover, by adding more vertices to the vertex cover. Therefore, T actually construct a polynomial time reduction from $VERTEX - COVER$ to $2 - NEIGHBORHOOD$, and hence by Theorems 7.44 and 7.36 we have that $2 - NEIGHBORHOOD$ is NP-complete. □