Advanced Complexity

TD n°2 : SPACE and NL

Charlie Jacomme

September 20, 2017

Exercise 1 : Warm up

Show that the following problems are NL-complete :

1. Deciding if a non-deterministic automaton $\mathcal{A}$ accepts a word w.
2. Deciding if a directed graph is strongly connected is NL-complete.
3. Deciding if a directed graph has a cycle.

Solution:

1. The problem is in NL as we can simply guess the path in the automaton corresponding to the word and arrive in an accepting state. To be in logspace, we do not however guess the whole path at once, but guess transition by transition while keeping a counter to the current state. Given a graph G, a starting node and an ending node, we label all the edges with the lettre ϵ in order to create an automaton $\mathcal{A}$, with initial state (resp. final) the starting (resp. ending) node. Now, REACH is equivalent to whether $\mathcal{A}$ accepts ϵ, we thus reduced REACH to our problem, which is then NL-complete.
2. The problem is in $co-\mathsf{NL}$ (=NL), as we can simply guess two nodes and check that they are not connected. We once again reduce REACH, so we are given (G, s, t). We construct G' by copying G, and for every node i, we add an edge from i to s, and one from t to i. This reduction is logspace as we only need one counter for the loop on the nodes. And finally, G' is strongly connected if and only if there is a path from s to t.
3. The problem is in NL, given G we guess an edge (x, y) of the cycle and run REACH on (G, x, y). We, one last time, reduce REACH. Given (G, s, t), we may create G' by first adding an edge between t and s, creating a cycle inside G' if s and t are connected in G. This is not enough, because G may have other cycles and the equivalence would not hold. Thus, we must first eliminate all the cycles in G. Let m be the number of nodes in G. We create m copies of G, which can be seen as m levels. For every edge from i to j in G, we draw an edge from node i at each level to node j at the next level. Additionally, we draw an edge from each node i at each level to node i at the next level. We call s' the s of the first level, and t' the t of the last level. Now, there is a path from s to t in G if and only if there is path from s' to t' in G'. Moreover, in G' path are only "going up" into the levels, so there cannot be any circle. Thus if we add an edge from t' to s', we now have that there is a path from s to t in G if and only if there is a cycle in G'.

Exercise 2 : Restrictions of the SAT problem

1. Let 3-SAT be the restriction of SAT to clauses consisting of at most three literals (called 3-clauses). In other words, the input is a finite set S of 3-clauses, and the question is whether S is satisfiable. Show that 3-SAT is NP-complete for logspace reductions (assuming SAT is).

2. Let 2-SAT be the restriction of SAT to clauses consisting of at most two literals (called 2-clauses). Show that 2-SAT is in P, using proofs by resolution.
3. Show that 2-UNSAT (i.e, the unsatisfiability of a set of 2-clauses) is NL-complete.
4. Conclude that 2-SAT is NL-complete.

Solution:

1. First, the problem is in NP as a sub case of SAT. We now must be able to transform any instance of SAT into an instance of 3-SAT. The idea is that we can replace a clause $L_1 \vee L_2 \vee C$ (C non empty) by the clauses $L_1 \vee L_2 \vee x$ and $\neg x \vee C$ with x fresh. Indeed, if $L_1 \vee L_2 \vee C$ can be satisfied, then either $L_1 \vee L_2$ can be satisfied, and then $\neg x \vee C$ with x set to false, either $L_1 \vee L_2$ cannot be satisfied, thus C can and we can set x to true. Conversely, if both clauses are true , if x is true then C is true, and if C is false, L_1 or L_2 is true. We do this for all the clauses in the formula, and it yields a 3-SAT formula with the same satisfiability. To conclude, we show that this transformation can be done in log space. We first read the input to obtain the number of variables N, and write $N+1$ on a tape B. Then, we treat each clauses one after the other, by writing the first variable to a tape B1, the second to a tape B2 and the third to B3. If the clause if over, we write it down directly on the output, if it is not, we write $B_1 \vee \mathcal{B}_2 \vee z$ where z is obtained from B. Then, we write $\neq z$, on B1, increment B, and the following variable goes to B2, et caetera et caetera. This requires 3 logspace tapes for the variables. The counter in B will not exceed $N + n/2 = O(n)$ (with n the number of literal, we at most create one fresh variable for every two literals), so B is also logspace.
2. From the formula S, we construct a graph G where the nodes are all the variables in S and their negation. For every clause $L \vee L'$ we create an edge from $\neg L$ to L' and one from $\neg L'$ to L. If there is an empty clause, we immediately conclude that S is not satisfiable. Else, if there is a path from x to $\neg x$ and one from $\neg x$ to x, then S is insatisfiable. Indeed, an edge represent an implication, which must be true in any model of S, and thus x and $\neg x$ would need to gave the same value in any model of S.

   If there is no such path, with $x_1, ..., x_n$ the variables of G, we define by induction on $1 \leq i \leq n$ the valuation of x_i and a graph G_i as follow : $G_0 = G$, and for $1 \leq i \leq n$, if there is a path from $\neg x_i$ to x_i in G_{i-1} then we set x_i to true and $G_i ? G_{i-1}$. Else, x_i is false and G_i is G_{i-1} with an edge between x_i to $\neg x_i$. By induction, we now show that G_i never contains a path going through a variable and its negation. It is true for G_0 by hypothesis, and when $G_i = G_{i-1}$ it is obvious. Else G_i is G_{i-1} with an edge between x_i to $\neg x_i$, knowing that there is no cycle containing x_i and $\neg x_i$. If there was a j such that there was a cycle containing x_j and $\neg x_j$, it would need to use the new arc, and if we assume that the cycle is minimal, then by removing the new arc we obtain a path from $\neg x_i$ to x_i in G_{i-1} which is a contradiction.

   Using this construction, we now show that the valuation satisfies S. Considering the construction, for every i there is either a path from x_i to $\neg x_i$ or the inverse, but not both at the same time. And the second case occurs when x_i is set to false, so the first one occurs when x_i is set to true. Thus, for a litteral L, L is set to true if and only if there is a path from L to $\neg L$, and respectively to false. For each edges (L, L') of G, if L was true and L' false, then we would have a path in G_n from $\neg L$ to L and one from L' to $\neg L$, which yields a path π from $\neg L$ to $\neg L'$ groing through L. But by construction, when (L, L') is an arc of G, so is $(\neg L', \neg L)$, which is then in G_n. With π and this edge, we then have a cycle in G_n passing through L and $\neg L$ which is a contradiction. Thus, for every edges (L, L') of G , the implication $L \Rightarrow L'$ is satisfied, i.e all the clauses of S are satisfied.
3. We can produce G in lospace, so the unsatisfiability reduces to finding a cycle containing an x and an $\neg x$. It is in NL by guessing x and calling $REACH(x, \neg x)$ and

$REACH(\neg x, x)$. For the completness, we reduce REACH and are given (G, s, t). For every edge (u, v), we create the clause $\neg u \vee v$ and then we create the clauses s and $\neg t$. We obtain a set S of clauses. If REACH(G, s, t), then the path yelds an implication from s to t, so t must be true but we have $\neg t$, S is unsatisfiable. Else, we set all the variables accessible from s to true, and the others to false. s and $\neg t$ are verified, and $u \Rightarrow v$ also as v is accessible from s if u is, i.e, we obtain a model for S, which is then satisfiable.

4. $2 - SAT$ is in $co - \mathsf{NL}$, and so in NL. Moreover, for any language $L \in \mathsf{NL}$, $\overline{L} \in \mathsf{NL}$. So $\overline{L}$ can be reduced to the the negation of 2-SAT (it is NL-complete). And then, this logspace reduction is a logspace reduction from L to 2-SAT.

Exercise 3 : Space hierarchy theorem

Using a diagonal argument, prove that for two space-constructible functions f and g such that $f(n) = o(g(n))$ (and as always $f, g \geq log$) we have $\mathsf{SPACE}(f(n)) \subsetneq \mathsf{SPACE}(g(n))$.

Solution:

We define a language which can be recognized in $O(f(n))$ but not in $g(n)$.

$$L = \{(M, w) | M \ reject \ (M, w) \ using \ space \leq f(|(M, w)|\}$$

— We show that $L \in SPACE(f(n))$ by constructing the corresponding TM. On an input x, we compute $f(x)$ and mark down an end of tape marker at position f(x), so that we reject if we use to much space. If x is not of the form (M, w), we reject, else we simulate M on w for at most $2^{f(x)}$ steps. If we go over the timeout, we reject. Else, if w is accepted, we reject, and if w is rejected, we accept.

— Show that $L \notin SPACE(f(n))$. Let's assume there is a machine M' recognizing L in space $f(n)$. For a sufficiently long w, M' uses less than $f(|(M', w)|)$ space on input M'. If $(M', w) \in L$, M' must both accept and reject (M', w). The other case is also a contradiction, thus it is impossible.