Advanced Complexity

TD n°3

Charlie Jacomme

September 27, 2017

Exercise 1 : 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 using space $O(g(n))$ but not in $f(n)$.

$$L = \{(M, w) | M \ reject \ w \ using \ space \leq g(|(M, w)| \ and |\Gamma| = 4\}$$

Where $\Gamma$is the alphabet of the Turing Machine

- We show that $L \in \mathsf{SPACE}(g(n))$ by constructing the corresponding TM. This is where we need the fact that the alphabet is bounded. Indeed, we want to construct one fixed machine that recognizes L for any TM M, and if we allow M to have an arbitrary size of alphabet, the fixed machine might need a lot of space in order to represent the alphabet of M, and it might go over $O(g(n))$. On an input x, we compute $g(x)$ 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 $|Q| \times 4^{g(x)} \times n$ steps. If we go over the timeout, we reject. Else, if w is accepted, we reject, and if w is rejected, we accept. This can be done in $\mathsf{SPACE}(O(g(n)))$, and we conclude with the speed up theorem.
- Show that $L \notin SPACE(f(n))$. Let's assume there is a machine M' recognizing L in space $f(n)$. We can create from M' a machine M'' which will recognize L in space $O(f(n))$ and with an alphabet of size 4. For a sufficiently long w, M'' uses less than $g(|(M'', w)|)$ 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 there cannot be a machine recognizing L in space $f(n)$.

Exercise 2 : Polylogarithmic space

Let $\mathsf{polyL} = \cup_{k \in \mathbb{N}} \mathsf{SPACE}(log^k(n))$. Show that $\mathsf{polyL} \neq \mathsf{P}$.

Solution:

P has a complete problem under logarithmic space many-one reductions but polyL does not due to the space hierarchy theorem. The space hierarchy theorem guarantees that $\mathsf{DSPACE}(log^k(n)) \subsetneq \mathsf{DSPACE}(log^{k+1}(n))$ for all integers $k > 0$. If polyL had a complete problem, call it A, it would be an element of $\mathsf{DSPACE}(log^k(n))$ for some integer $k > 0$. Suppose problem B is an element of $\mathsf{DSPACE}(log^{k+1}(n))$ $\mathsf{DSPACE}(logk(n))$. The assumption that A is complete implies the following $O(log^k(n))$ space algorithm for B : reduce B to A in logarithmic space, then decide A in $O(log^k(n))$ space. This implies that B is an element of $\mathsf{DSPACE}(log^k(n))$ and we have a contradiction.

Exercise 3 : Padding argument

1. Show that if $\mathsf{DSPACE}(n^c) \subseteq \mathsf{NP}$ for some $c > 0$, then $\mathsf{PSPACE} \subseteq \mathsf{NP}$.
2. Deduce that $\mathsf{DSPACE}(n^c) \neq \mathsf{NP}$.

Solution:

1. Assume $\mathsf{DSPACE}(n^c) \subseteq \mathsf{NP}$ and let there be $L \in PSPACE$. For some k, we have $L \in \mathsf{DSPACE}(n^k)$. Then, $\tilde{L} = \{(x, 1^{x^{k/c}}) | x \in L\} \in \mathsf{DSPACE}(n^c) \subseteq \mathsf{NP}$. Thus $\tilde{L} \in NP$. As we can reduce L to $\tilde{L}$ by transforming x into $(x, 1^{x^{k/c}})$ in logspace, we do have that $L \in NP$. This is true for any L, so it yields the result.
2. Assume $\mathsf{DSPACE}(n^c) = \mathsf{NP}$, then $PSPACE = NP = \mathsf{DSPACE}(n^c)$ which is a contradiction to the space hierarchy theorem.

Exercise 4 : My very first PSPACE-complete problem

Show that the following problem is PSPACE-complete (not assuming anything about QBF) :

— INPUT : a Turing Machine M and a word w and a number t written in unary

— QUESTION : does M accepts w within space t ?

Solution:

— The problem is PSPACE : given an input (M, w, t), we can build M' which simulates M on w and rejects if M uses more than t cells, and returns the result of M otherwise. As t is part of the input, we are effectively in PSPACE.

— Hardness : Let L be a language and M a TM which recognizes L in PSPACE. $M \in$ PSPACE, so there is a polynomial p which can bound the size of the runs of M. So on input w, we have that $w \in L$ iff M accepts w within space $1^{p(|w|)}$, and as we can construct $1^{p(|w|)}$ in logspace, we have an effective reduction from L to our problem.

Exercise 5 : PSPACE and games

The Geography game is played as follow :

— The game starts with a given name of a city, for instance *Cachan* ;

— the first player gives the name of a city whose first letter coincides with the last letter of the previous city, for instance *Nice* ;

— the second player gives then another city name, always starting with the last letter of the previous city, for instance *Evry* ;

— the first player plays again, and so on – with the restriction that no player is allowed to give the name of a city already used in the game ;

— the loser is the first player who does not find a new city name to continue.

This game can be described using a directed graph whose vertices represent cities and where an edge (X, Y) means that the last letter of the city X is the same as the first letter of the city Y. This graph has also a vertex marked as the initial vertex of the game (the initial city). Each player choses a vertex of the graph, the first player choses first, and the two players alternate their moves. At each move, the sequence of vertices chosen by the two players must form a simple path in the graph, starting from the distinguished initial vertex.

Player 1 wins the game if, after some number of moves, Player 2 has no valid move (that is no move that forms a simple path with the sequence of previous moves).

`GEOGRAPHY` is the following problem :

— INPUT : a directed graph G and an initial vertex s.

— QUESTION : is the player 1 sure to win the game on G starting at s ?

Show that `GEOGRAPHY` is PSPACE-complete by :

1. Showing that `GEOGRAPHY` is PSPACE
2. That the satisfiability of a QSAT formula of the form $\exists x_1 \forall x_2 \ldots \exists x_n \bigwedge (C_i)$ where C_i is a clause can be expressed as a `GEOGRAPHY` instance.

Solution:

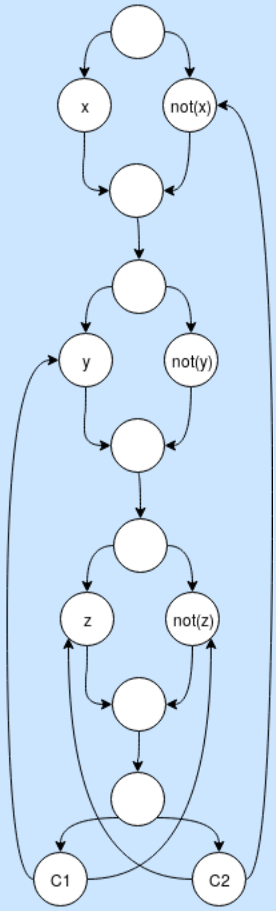
1. Actually, any such game can be solved in PSPACE. Given the input, we run a recursive algorithm $win(G, (s_1, ...s_k))$ which says if, given the graph G and the previous

moves $(s_1, ...s_k)$ the player has a winning strategy. We define :

$$win(G, (s_1, ...s_k)) := \bigvee_{s_k \to s \in G, s \neq s_i} \neg win(G, (s_1, ..., s_k, s))$$

At every recursive call we only add one node to the recursion stack, thus the size of the recursive stack is at most the size of the graph, we do are in |G|.

2. The figure gives an example for $\exists x \forall y \exists z.(\neg y \vee z) \wedge (x \vee \neg z)$ Each variable is replaced by a diamond-like "choice-gadget" and all those gadgets are put sequentially. Any path in the graph will pick a valuation for the variables. The $\exists$ variables are made to be chosen at player 1 turn, and $\forall$ at player 2 turn. Then, at the end of the valuation, we add a node for every clause. Intuitively, the player 2 is allowed to chose a clause that he thinks might be false according to the valuation. Finally, from every clause, we have an edge from this node to the inverse of the litteral in the clause. Thus, if the clause is satisfied by the valuation, we have for instance x in the clause and x set to true, then player 1 can take the path which leads to $\neg x$ and block player 2. If the formula is satisfiable, player 1 will always be able to make the good choices, and if it is not, the player 2 may block him. We do have a reduction, which is in logspace.



,