

# CMPT 710/407 - Complexity Theory: Lecture 12

Valentine Kabanets

October 18, 2007

## 1 NL = coNL

Next we turn to an even more amazing result in complexity which proves the closure of NL under complementation. (This is like  $\text{NP} = \text{coNP}$  for space-bounded machines!) The question whether nondeterministic space is closed under complementation was open for 23 years; it was first stated in 1964 by Kuroda in relation to the class of context sensitive languages, which Kuroda proved to be exactly the class  $\text{NSpace}(n)$ . In 1987, Neil Immerman, an American researcher, and Robert Szelepcsényi, a Slovakian undergraduate student, independently proved that  $\text{NSpace}(s(n)) = \text{coNSpace}(s(n))$ , for any proper complexity function  $s(n) \geq \log n$ . This was a big shock to the CS community for two reasons: (1) it was widely believed that  $\text{NL} \neq \text{coNL}$ , and (2) the proofs by Immerman and Szelepcsényi were quite simple.

Since  $ST - CON$  is NL-complete, in order to prove  $\text{NL} = \text{coNL}$ , it suffices to prove that  $ST - CON \in \text{coNL}$ , i.e., that it can be checked in nondeterministic logspace whether  $t$  is *not* reachable from  $s$ .

**Theorem 1** (Immerman-Szelepcsényi).  $ST - CON \in \text{coNL}$

*Proof.* Idea: To check if  $t$  is not reachable from  $s$ , enumerate all nodes that *are* reachable from  $s$  and check that  $t$  is *not* among them.

This sounds too easy. The trick is to do this enumeration of *all* nodes in logspace, and ensuring that indeed *all* nodes reachable from  $s$  were enumerated. We need some clever idea to do this. The clever idea is to *count*.

Let us imagine for a moment that we are given a number  $N = \#$  of nodes reachable from node  $s$ . (Later we'll show how to compute this  $N$  in NL.) The following NL algorithm check if  $t$  is *not* reachable from  $s$  in a given directed graph  $G = (V, E)$ , where  $|V| = n$ .

**Algo**  $Unreach(G, s, t)$

% given  $N = \#$  nodes reachable from  $s$

$count = 0$ ;

**for** every node  $v$

        “make a nondeterministic guess whether  $v$  is reachable from  $s$ ”

**if** guess is Yes **then**

            “nondeterministically try to guess a path from  $s$  to  $v$  of length at most  $n$ ”;

```

    if “guessed path does not lead to  $v$ ” then Reject end if
    if  $v = t$  then Reject
    else  $count = count + 1$ ;
    end if
  end if
end for
if  $count < N$  then Reject
else Accept % if  $count = N$ 
end if
end Algo

```

Clearly the algorithm *Unreach* runs in nondeterministic logspace; observe that  $N$  and  $count$  can be at most  $n$ , and so they can be written as binary numbers of length at most  $\log n$ .

**Claim 2.** *Algorithm  $Unreach(G, s, t)$  has an accepting computation iff  $t$  is not reachable from  $s$ .*

*Proof of Claim.* The algorithm makes sure that it enumerates all nodes reachable from  $s$ , by comparing  $count$  with  $N$ . The algorithm accepts iff node  $t$  was not one of these  $N$  nodes reachable from  $s$ .  $\square$

To compute  $N = \#$  nodes reachable from  $s$ , we will iteratively compute (re-using space) the values  $R(i) = \#$  nodes reachable from  $s$  in at most  $i$  steps. Then we obtain  $N = R(n)$ .

```

Algo #Reach( $G, s, t$ )
   $R(0) = 1$  %  $s$  is reachable from  $s$  in 0 steps
  for  $i = 1..n$ 
     $R(i) = 0$  % initialize  $R(i)$ 
    for every node  $v$ 
      % try all nodes  $u$  reachable from  $s$  in  $\leq (i - 1)$  steps, and
      % check if  $v$  is reachable in  $\leq 1$  steps from any such  $u$ 
       $count = 0$ ;
      for every node  $u$ 
        “make nondeterministic guess whether  $u$  is reachable from  $s$  in  $\leq (i - 1)$  steps”;
        if “guess is Yes” then
          “nondeterministically try to guess a path from  $s$  to  $u$  of length  $\leq (i - 1)$ ”;
          if “guessed path does not lead to  $u$ ” then Reject end if
           $count = count + 1$ ; % if  $u$  is reachable, count it in
          if  $u = v$  OR  $(u, v) \in E$ 
            then  $R(i) = R(i) + 1$ ;
            break; % go to next iteration of “for  $v$ ” loop
          end if
        end if
      end for
    end for
  end for
  if  $count < R(i - 1)$  then Reject

```

```

        end if
    end for
end for
return  $R(n)$ ;
end Algo

```

**Remark:** The algorithm *#Reach* needs to remember only two successive values  $R(i)$  and  $R(i + 1)$  at any point in time. So, it re-uses space when computing  $R(1), \dots, R(n)$ . Thus, the algorithm can be made to run in NL.

**Claim 3.** *Algorithm #Reach computes the number of nodes reachable from  $s$ .*

*Proof of Claim.* The proof is by induction on  $i$ . For  $i = 0$ ,  $R(0) = 1$  is obviously correct.

For the induction step, assume that  $R(i)$  is equal to the number of nodes reachable from  $s$  in at most  $i$  steps. We need to prove that  $R(i + 1)$  is equal to the number of nodes reachable from  $s$  in at most  $(i + 1)$  steps. To prove this, notice that the algorithm increments  $R(i + 1)$  on a node  $v$  iff  $v$  is reachable from  $s$  in at most  $(i + 1)$  steps. This is because  $R(i + 1)$  is *not* incremented only if all nodes at distance  $\leq i$  from  $s$  were tried, and  $v$  is *not* reachable in  $\leq 1$  steps from any one of them.  $\square$

Thus, to check if  $t$  is not reachable from  $s$ , we first run the algorithm *#Reach* to compute  $N$ , then run the algorithm *Unreach* with that  $N$ . The total space this nondeterministic procedure takes is  $O(\log n)$ , because each of the two algorithms is logspace.  $\square$

## 2 PSPACE-Completeness

Recall the NP-complete problem SAT: Is a given Boolean formula  $\phi(x_1, \dots, x_n)$  satisfiable? The same question can be stated equivalently as: Is the formula  $\exists x_1 \dots \exists x_n \phi(x_1, \dots, x_n)$  true? Also recall the coNP-complete problem TAUT: Is a given Boolean formula  $\psi(x_1, \dots, x_n)$  a tautology (i.e., true on all assignments)? This can be stated as: Is the formula  $\forall x_1 \dots \forall x_n \psi(x_1, \dots, x_n)$  true?

What if we start alternating the quantifiers? Let us define the language TQBF (True Quantified Boolean Formulas) as the set of true formulas of the form

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n \phi(x_1, x_2, \dots, x_n),$$

where  $Q_i \in \{\exists, \forall\}$  is a quantifier,  $\phi$  is a Boolean formula in CNF (conjunctive normal form), and the sequence of quantifiers  $Q_1 \dots Q_n$  alternates between  $\exists$  and  $\forall$ . For example, all odd  $Q_{2k+1}$  can be  $\exists$ , and all even  $Q_{2k}$  can be  $\forall$ .

We can assume, without loss of generality, that  $Q_1 = \exists$ : if not, then we can always add an  $\exists y$  before the formula for some variable  $y$  that does not occur in the formula. In this case, one can think of a QBF as a *game* between two players: Player  $\exists$  and Player  $\forall$ . First, Player  $\exists$  sets the value of  $x_1$ . Then Player  $\forall$  sets the value of  $x_2$ . Then Player  $\exists$  sets the value of  $x_3$ , and so on. After  $n$  moves, the winner is declared using the following criterion: Player  $\exists$  wins iff the assignments to  $x_1 \dots x_n$  constructed during the game is *satisfying* for

the formula  $\phi$ . It is easy to see that a given QBF is true iff Player  $\exists$  has a *winning strategy*, i.e., for any choice of moves by Player  $\forall$ , Player  $\exists$  can play so as to guarantee its win.

How can we decide if a given QBF is true? The following simple recursive algorithm *Truth* does the job.

```

Algo Truth( $\Phi$ )
if  $\Phi$  is quantifier-free then return its value
end if
Let  $\Psi = Q_1x_1 \dots Q_nx_n\phi(x_1, \dots, x_n)$ .
 $b_0 = \text{Truth}(Q_2x_2 \dots x_n\phi(0, x_2, \dots, x_n))$ ;
 $b_1 = \text{Truth}(Q_2x_2 \dots x_n\phi(1, x_2, \dots, x_n))$ ; % re-using space
if  $Q_1 = \exists$  then return  $b_0 \vee b_1$ 
   else return  $b_0 \wedge b_1$ 
end if
end Algo

```

The algorithm *Truth* runs in polynomial space: the depth of the recursion is  $n$ , and the size of each stack record is  $\text{poly}(n)$ .

**Remark:** Using some tricks (as those we used to show that every  $O(\log n)$ -depth formula can be evaluated in  $O(\log n)$ -space), one can show that TQBF is in  $\text{Space}(n)$ .

It turns out that PSPACE is probably the best possible we can do for TQBF, as it is PSPACE-complete.

**Theorem 4.** *TQBF is PSPACE-complete.*

*Proof.* (1)  $TQBF \in \text{PSPACE}$ : we already showed that above.

(2)  $TQBF$  is PSPACE-hard. We need to reduce every language in PSPACE to TQBF. According to our definition of TQBF, the inner formula must be in CNF. However, it will be easier for us to reduce every language in PSPACE to a quantified formula in DNF (disjunctive normal form). This turns out to be sufficient since PSPACE is closed under complementation. (Check this.)

Let  $L$  be any language in PSPACE. Say  $L$  is decided by some TM  $M$  in  $\text{Space}(n^c)$  for some constant  $c$ . To decide if  $x \in L$ , we consider the configuration graph of  $M$  on  $x$ . Each configuration will be of size  $O(n^c)$ ; let us denote this size by  $m$ . There at most  $2^m$  different configurations. So,  $x \in L$  iff there is a path from the start configuration to the accept configuration of length at most  $2^m$ . We will construct a QBF that will express the existence of such a path.

In a sense, our proof is a restatement of the proof of Savitch's Theorem. Recall the function  $\text{Path}(a, b, i)$  that we used in the proof of Savitch's Theorem:  $\text{Path}(a, b, i)$  is True iff there is a path from node  $a$  to node  $b$  of length at most  $2^i$ . We will define a sequence of QBFs  $\psi_i$ ,  $i = 0, \dots, m$ , where  $\psi_i(A, B)$  is True iff the variables  $A$  and  $B$  encode two configurations  $a$  and  $b$  such that  $b$  is reachable from  $a$  in at most  $2^i$  steps; note that  $A$  and  $B$  are groups of  $m$  variables each. The required QBF will then be  $\psi_m(\text{Start}, \text{Accept})$ , where *Start* and *Accept* are the binary strings encoding the start configuration and accept configuration, respectively.

For  $i = 0$ ,  $\psi_0(X, Y)$  is a quantified Boolean formula on free variables  $X$  and  $Y$  that is True iff either  $X = Y$  or  $Y$  is reachable from  $X$  in 1 step. The formula  $\psi_0(X, Y)$  can be written as a *DNF* of size  $\text{poly}(m)$ . (*Exercise:* Explain why.)

Given  $\psi_i(X, Y)$ , we can try to define  $\psi_{i+1}(X, Y) = \exists Z[\psi_i(X, Z) \wedge \psi_i(Z, Y)]$ . However, this is a *bad* idea: the size of  $\psi_{i+1}$  doubles, and so, the size of  $\psi_m$  would be exponential. We need to do something more. To be continued next time...  $\square$