CS 4510 : Automata and Complexity
$\mathsf{NL} = \mathsf{coNL}$

Subrahmanyam Kalyanasundaram

April 19, 2010

Today we see the result that nondeterministic logspace, NL is closed under complement. The result was proved by Neil Immerman and Róbert Szelepcsényi in 1987.

Theorem 1 $\mathsf{NL} = \mathsf{coNL}$.

Towards this end, we show that the coNL-complete language $\overline{PATH}$ is contained in NL. Why is this enough? We have seen that $PATH$ is an NL-complete language. This implies that the complement of the language $\overline{PATH}$ is the complete language of the complement class coNL.

We shall see today that $\overline{PATH} \in \mathsf{NL}$. This implies that $\mathsf{coNL} \subseteq \mathsf{NL}$. Also we have

$$\overline{PATH} \in \mathsf{NL} \Rightarrow PATH \in \mathsf{coNL} \Rightarrow \mathsf{NL} \subseteq \mathsf{coNL}$$

Together we get the required result $\mathsf{NL} = \mathsf{coNL}$.

$$\overline{PATH} = \{\langle G, s, t\rangle | G \text{ has no directed } s - t \text{ path}\}$$

We show $\overline{PATH} \in \mathsf{NL}$ in two parts. First, we look at a simpler problem. Given c, the number of vertices that in G can be reached from s, can you show that t is not reachable from s?. We shall see an NL machine which accepts, when t is not reachable from s, and which rejects when t is reachable from s. Second, we show how one can use an NL machine to compute the value of c correctly.

First part 1.

- Given G, s, t, c.
- A certificate that t is not reachable from s is the list of c vertices which are reachable from s, a list which does not contain t.
- The NL machine M needs to guess this list and verify.
- The NL machine goes through all vertices in G, and nondeterministically chooses if each one is reachable from s.

- When u is guessed to be reachable from s, M guesses a path from s to u and verifies the existence of this path. M rejects if the path does not exist.
- If t is guessed to be reachable, M rejects.
- M accepts if the list contains exactly c reachable vertices from s which have been verified.

Algorithm 1 NL algorithm to which accepts if t is not reachable from s, given c

---

Let $d = 0$.
for each vertex $u \in G$ **do**
    Nondeterministically either perform or skip the following steps:
    Call the function CHECKPATH$(G, s, u, |V|)$.
    If $u = t$, then reject.
    $d \leftarrow d + 1$
if $d = c$ **then**
    Accept.
else
    Reject.

Notice that the algorithm uses space only to store the counters d, u and to pass the parameters $s, u, |V|$ to CHECKPATH. This can be done using logarithmic space. Also, here CHECKPATH$(G, s, u, |V|)$ is the NL computation that accepts if there is a path from s to u in G: this happens if there is such a path of length $|V|$ or less. All we need to do is the following : guess a path nondeterministically from s to u of length $|V|$ or less, and check the validity of this path. This is done by maintaining a counter j, and for each $j = 1, \ldots, |V|$, guess a vertex w reachable from s by a path of length j. For $j + 1$, guess an edge w' from w, and reject if the edge (w', w) is not present in G. Finally, when $j = |V|$, reject if $w \neq u$. If $w = u$, the machine has guessed and verified the existence of a path of length $\leq |V|$ from s to u, and proceeds with the main algorithm.

Algorithm 2 CHECKPATH(G, s, u, k)

$w' = s$.
for $j = 1$ **to** k **do**
    Nondeterministically guess a w, and reject if $[((w', w) \notin E(G))\text{AND}(w \neq w')]$.
    if $w = u$ **then**
        Accept and return.
    $w' \leftarrow w$.
    $j \leftarrow j + 1$.
Reject.

Here we need to store only j, w, w', each of which requires logarithmic space.

Now part 2. How do we compute c in NL? Before that we need to define how a nondeterministic machine can compute a function.

> We say that a nondeterministic algorithm computes c, if it either rejects, or completes the computation and returns the correct value of c. In other words, every non-rejecting path computes c correctly.

The value of c is calculated by recursively computing $c_i = |A_i|$ for all $i = 0, \ldots, |V|$. Here A_i is the set of all vertices reachable from s using a path of length i or less. We calculate c_{i+1} from c_i. Notice that A_i can have $O(n)$ elements, so we cannot hope to store A_i fully. The clever method to solve this is to pass on just c_i to the next loop of the computation. $A_0 = \{s\}$, so $c_0 = 1$. The inductive step computes c_{i+1} from c_i.

The idea is similar to the first part. To find c_{i+1}, the machine checks for each candidate $v \in A_{i+1}$. For each v, there would be at least one computation which verifies it to be in A_{i+1}. How does M do it? For each v, M tries to reconstruct all the elements of A_i. $v \in A_{i+1}$ if there is one $u \in A_i$ such that $(u, v) \in E(G)$. Once again, M can never reconstruct the whole set A_i at once, because of the space constraint. Instead it has to do it serially, one by one, and then use the knowledge of c_i to check if the computation was correct. This is done by maintaining a count, and rejecting if M did not see enough elements of A_i. $v \in A_{i+1}$ if M discovers a path of length $\leq i + 1$ to v. If A_i has been verified to be correctly computed, and v has not yet been shown to be a neighbor of a vertex in A_i, we can conclude that $v \notin A_{i+1}$. Then we go to the next v.

Algorithm 3 To compute c, given G, s

Let $c_0 = 1$.
for $i = 0$ **to** $|V| - 1$ **do**
  Let $c_{i+1} = 1$.
  for each node $v \neq s$ in G **do**
    Let $d = 0$.
    for each node u in G **do**
      Nondeterministically either perform or skip the following steps.
      Call the function CHECKPATH(G, s, u, i).
      $d \leftarrow d + 1$.
      if $(u, v) \in E(G)$ **then**
        $c_{i+1} \leftarrow c_{i+1} + 1$
        Go to Stage 5 with the next v.
    if $d \neq c_i$ **then**
      Reject.
return $c_{|V|}$

This needs to store counters for i, d, u, v and also c_i and c_{i+1}. All of these numbers are bounded by n, so each one of these takes logarithmic space at most.