NL=coNL

Theorem: NL=coNL

Proof.
It suffices to show the coNL-complete problem
  nonPATH={<G,s,t> | G is a directed graph with no path from s to t}
is in NL.
[Since coNL \subseteq NL \implies coNL=NL.]

The idea of the proof is to show all nodes reachable from the source node s of a digraph G doesn’t contain the target node t.
We do this by verifying nondeterministically the counts (and sets of nodes) reachable at distance 0 to |G|. We accept if and only if node t isn’t reachable.

First, (just for illustration purposes) we show that we can solve an easier problem nonPATHCOUNT in NL. This is the same as nonPATH except we are given the count c of nodes reachable from node s.

```
Alg: nonPATHCOUNT(G,s,t,c)
d=1
for u in G\s do
{
  nondeterministically go to next u   (i.e. don’t think u is reachable)

  v1=s
  for i=1 to |G|
  {
    nondeterministically pick node v2
    if "(v1,v2) isn’t an edge" reject
    if (v2==t) reject
    if (v2==u) { d=d+1; next u }
    v1=v2
  }
  reject
}
if (d==c) accept else reject
```

We now write a bootstrap version of nonPATHCOUNT to actually figure out the desired count value c. We do this by generating counts for each distance from s (1 up to |G|-1).

```
Alg: nonPATH(G,s,t)
c=1                      // one node s in level 0
for j=1 to |G|-1         // incrementally compute levels away from s
{
  y=0                    // to store next level count
  for v in G do
  {
    x=1                  // to verify current level count
    for u in G do
    {
       nondeterministically go to next u
       v1=s
       for i=0 to j-1    // j is next maximum path length
       {
         if (v1==u)
         {
           x=x+1
           if v in N[u] { y=y+1; next v }
           next u
         }
         nondeterministically pick node v2
         if "(v1,v2) isn’t an edge" reject
         if (v2==t) reject
         v1=v2
       }
    }
    if (x!=c) reject  // didn’t verify earlier count
  }
  c=y                 // next level count
}
accept                // verified final count without seeing t
```