Theorem[Immerman/Szelepcsény]: \( \text{NL} = \text{coNL} \)

January 10, 2009

Our aim is to show \((s, t)\)-NON-CONNECTIVITY is in NL, which implies the theorem. Let us start with some definitions.

**Definition 1.** For any directed graph \( G = (V, E) \) and a vertex \( s \in V \) designated as the start vertex of \( G \), denote

\[
\text{reachable}(G) = \{ v \in V | s \rightarrow v \}
\]

where “\( \rightarrow \)” denotes a directed path in \( G \).

Assume \( t \in V \) is the designated target vertex in \( G \), and define \( G_{-t} = (V, E - V \times \{t\}) \) — namely, the graph that results from removing from \( G \) all edges leading to \( t \). Of course, the above definition applies to it too: \( \text{reachable}(G_{-t}) \) is the set of all vertexes in \( G \) reachable from \( s \) without passing through \( t \).

Now, let \( \text{reachable}_l(G) = \{ v \in V | s \rightarrow_l v \} \) where “\( u \rightarrow_l v \)” denotes there is a path from \( u \) to \( v \) in \( G \) of length \( \leq l \).

**Claim 0.1.** For any directed graph \( G = (V, E) \) and a designated start vertex \( s \) and target vertex \( t \), \( \text{reachable}(G_{-t}) \subseteq \text{reachable}(G) \).

**Proof.** For \( v \in \text{reachable}(G_{-t}) \), by definition, there is a path \( s \rightarrow v \) in \( G_{-t} \), which is also a path in \( G \).

**Lemma 0.2.** For any graph \( G \),

\[
|\text{reachable}(G_{-t})| \neq |\text{reachable}(G)| \text{ iff } s \rightarrow t \text{ in } G
\]

**Proof.** First, note that by definition of \( G_{-t} \), \( t \notin \text{reachable}(G_{-t}) \).

If \( s \rightarrow t \) then \( t \in \text{reachable}(G) \) and by the claim \( |\text{reachable}(G_{-t})| < |\text{reachable}(G)| \).

If \( |\text{reachable}(G_{-t})| = |\text{reachable}(G)| \) it must be that \( t \notin \text{reachable}(G) \) as well.

Therefore, to demonstrate there is no path \( s \rightarrow t \) in \( G \), it is enough to show that

\[
|\text{reachable}(G_{-t})| = |\text{reachable}(G)|
\]

Hence, to show that our problem is in NL, it is enough to give an NL-witness to this fact. Recall that an NL-witness is one that can be verified by an L TM, which reads the witness bit by bit
(cannot go back on the witness tape). Consequently, it suffices to show how to construct an NL-witness for \( \text{reachable}(G) = r \) for a general \( G \) and for the appropriate \( r \). The NL-witness for the above claim can first attest that \( \text{reachable}(G) = r \) and then that \( \text{reachable}(G_\setminus t) = r \) — for the same \( r \). (An L TM can easily read the graph \( G \) however work as if seeing \( G_\setminus t \)). The L TM can register \( r \) from the first part of the witness, and compare it with the second part of the witness.

Our remaining goal is to exhibit such an NL-witness to the fact that \( \text{reachable}(G) = r \).

Observe that \( \text{reachable}_{|V|}(G) = \text{reachable}(G) \).

The Witness

The NL-witness is constructed inductively: assuming \( W \# r_1 \# \) is an NL-witness that \( \text{reachable}_{l}(G) = r_l \), extend that witness to become an NL-witness attesting that \( \text{reachable}_{l+1}(G) = r_{l+1} \).

Note that throughout, \( W, W_i \) and \( W_j \) are variables for presentation purpose (not to be read as actual letters), each representing a string.

**Base case:** \#1\# is a trivial proof that \( \text{reachable}_{0}(G) = 1 \).

**Induction step:** To extend \( W \# r_1 \# \) into an NL-witness for \( l + 1 \), append to it \( |V| \) strings, each of the form

\[ b_i \$ W_i \$ \]

where \( b_i = 1 \) is interpreted as \( i \in \text{reachable}_{l+1} \) while \( 0 \) that it is not (we assume the set of vertexes is \( \{1, \ldots, |V|\} \)). Each \( W_i \) should be a string representing a witness that \( b_i \) indicates correctly whether \( i \) is or is not reachable by at most \( l + 1 \) steps from \( s \).

**In case** \( b_i = 1 \): \( W_i \) is simply a path of length \( \leq l + 1 \) from \( s \) to \( i \) (represented according to whichever convention as a 0/1 string).

**In case** \( b_i = 0 \): \( W_i \) is constructed by appending \( |V| \) strings, each of the form

\[ c_j * Z_j * \]

\( c_j \) is a 0/1 bit where \( c_j \) should be 1 iff \( s \rightarrow_l j \) (namely, \( j \in \text{reachable}_{l} \)). \( Z_j \) is then interpreted as a witness that \( c_j \) is the correct indication as to whether \( j \) is reachable from \( s \) within \( l \) steps.

If \( c_j = 1 \): again, \( Z_j \) can simply be a path of length \( \leq l \) from \( s \) to \( j \). Note however that if there is an edge in \( G \) from \( j \) to \( i \), then \( s \rightarrow_l j \) implies \( s \rightarrow_{l+1} i \) and the witness is not well-constructed (recall it is trying to prove \( b_i \) indeed should be 0).

It is however not a good idea to proceed, in case \( c_j = 0 \), recursively, as this would blowup the size of the witness to being exponential in the size of the graph.

Instead, in case \( c_j = 0 \), \( Z_j \) is an empty string.

How can then the L TM verifier make sure all \( b_j \)'s are correct? Here is the crux of the entire construction and proof: It only needs to count the number of \( j \)'s for which \( b_j = 1 \), and verify it is correct. It can do that by comparing that number to \( r_l \)!!

Let us now describe the L TM verifier. Note that read-letter, read-bit, read-number and verify-path are procedure calls that either read a character, a bit, a \( \log(|V|) \)-bit number, or verify a path between \( s \) to a vertex of some given length. They all reject unless their input is well constructed and valid, and read the witness bit-by-bit as necessary.
verify()
rl=1
for (l=1..|V|)
  if (read-letter() <> '#') reject
  if (read-number() <> rl) reject
  if (read-letter() <> '#') reject
r=0
for (i=1..|V|)
  bi = read-bit()
  if (read-letter() <> '$') reject
  if (bi=1)
    verify-path(l+1, i)
    increase r by 1
  else verify-no-path(l, i, rl)
  if (read-letter() <> '$') reject
end
rl=r
end
return(accept)

verify-no-path(l, i, rl)
rl' = 0
for (j=1..|V|)
  cj = read-bit();
  if (read-letter() <> '*') reject
  if (cj=1) then
    if (edge (j, i) in G) reject
    verify-path(l, j);
    increase rl' by 1
  if (read-letter() <> '*') reject
end
if rl' <> rl reject
return(accept)