8. Immerman & Szelepcsényi's Theorem

Goal: Show that for $s(n) \geq \log n$
the class $NSPACE(s(n))$ is closed under complement.

Importance:
- Shows $NL = \text{co-}NL$
  - ↳ This is like $NP = \text{co-}NP$ but for space.
  - ↳ Does not follow from Savitch.
- Solves the second LBA problem posed by Kuroda '64:
  - ↳ Kuroda showed that
    non-deterministic, linear-bounded automata
    (NTM with linear space bound)

    accept precisely

    the context-sensitive languages.
  - ↳ Kuroda posed two questions:

    (1) Are the languages accepted by NLBA
    precisely the languages accepted by DLBA?

    In our terms:

    $$NSPACE(O(n)) = DSPACE(O(n)) \,?$$

    (2) Are the languages accepted by NLBA
    closed under complement:

    $$NSPACE(O(n)) = \text{co-}NSPACE(O(n)) \,?$$
  - ↳ Kuroda showed that

    $\neg(2) \Rightarrow \neg(1)$.

    This did not help much as
    Immerman and Szelepcsényi proved (2) to hold!
  - ↳ Problem (1) is still open.

History: • The result was proved independently 1988 and 1987
by ↳ Neil Immerman (big fish already those days,
Univ. of Massachusetts Amherst)
↳ Róbert Szelepcsényi (student in Bratislava, Slovakia).
• Both received the Gödel-Prize 1995.
• The result brought the method of inductive counting to complexity theory.

Theorem (Immerman & Szelepcsényi 1988'87):
For $s(n) \geq \log n$, we have
$$NSPACE(s(n)) = co\text{-}NSPACE(s(n)).$$

The key of the proof is to show that non-reachability in a graph can be solved in non-deterministic logarithmic space.

Let $\overline{PATH}$ be the problem
Given: A directed graph G with nodes s and t.
Problem: Show that there is no path from s to t.

Theorem:
$\overline{PATH} \in NL$.

Proof:
Idea: • To check that t is not reachable from s, enumerate all nodes that are reachable from s and check that t is not among them.
• Sounds too easy? How to enumerate all nodes in logarithmic space? Ask differently? How to ensure that all nodes reachable from s were enumerated?
Clever idea: Counting?

In detail: • Assume we are given

$N =$ # nodes reachable from s.
(number of)

We show below how to compute N in NL.

• Given N, the following algorithm

↳ checks that t is not reachable from s
in $G = (V, \to)$ with $|V| = n$,

↳ works in NL.

```
bool unreach (G, s, t)
begin
  // Given N = # nodes reachable from s.
  count := 0;
  for every node v do
    "make a non-deterministic guess
          whether v is reachable from s";

    if guess = true then
      "non-deterministically try to guess a path
                    from s to v of length ≤ n";

      if "guessed path does not lead to v" then
                                  // Wrong path or wrong guess.
        return false;
      else if v = t then
        return false;                     // Reachable.
      else                                // Another reachable node ≠ t found.
        count ++;
      end if
    end if
  end for
```

```
  if count < N then              // Guessed incorrectly
     return false;                  about reachability for some v.
  else
     return true;                // Unreachable.
  end if
end
```

Algorithm unreach runs in non-deterministic logarithmic space.
Indeed, N and count can be at most n,
so they can be written down in binary at length $\log n$.

Lemma (Correctness):
Algorithm unreach(G, s, t) has a computation that returns true
iff t is not reachable from s.

Proof:
The algorithm makes sure it enumerates all nodes reachable from s
by comparing count with N.
The algorithm accepts iff t was not one of the N nodes reachable from s. $\square$

It remains to compute
$N = \#$ nodes reachable from s.
The key idea, nowadays called method of inductive counting,
is to inductively compute the values
$R(i) := \#$ nodes reachable from s in $\leq i$ steps.
Then $N = R(n)$.

```
Ifalse/u IN  #reach(G, s)
begin
  R(0) := 1;            // s is reachable from s in 0 steps.
  for i = 1, ..., n do
    R(i) := 0;          // Initialize R(i)
(*)   for every node v do
        // Try all nodes u reachable from s in ≤ i-1 steps.
        // Check if v is reachable from such a u in ≤ 1 steps.
        count := 0;
        for every node u do
          "make a non-deterministic guess
              whether u is reachable from s in ≤ i-1 steps"
          if guess = true then
            "non-deterministically try to guess a path
                 from s to u of length ≤ i-1"
            if "guessed path does not lead to u" then
              return false;
            else
              count++;        // If u is reachable, count it in.
              if u = v or u → v then
                R(i)++;
                goto (*);     // Go to next iteration
                              //   of "for v" loop.
              end if
            end if
          end if
        end for               // Loop for u.
        if count < R(i-1) then return false;   // Guessed incorrectly
                                               //   about reachability
                                               //   for some v.
```

end for // Loop for v.
end for // Loop for i.
return $R(n)$;
end

__Remark:__

At any point in time, algorithm #reach needs to remember only two successive values $R(i-1)$ and $R(i)$.
So it can reuse space when computing $R(1), \dots, R(n)$,
and can be made run in NL.

__Lemma:__

#reach computes the number of nodes reachable from s.

__Proof:__

We proceed by induction on i and show that upon termination of the iteration for i:

$R(i) = \#$ nodes reachable in $\leq i$ steps

Base case ($i=0$): $R(0) = 1$ is correct.

Induction step ($i \to i+1$): Assume the equality holds for $R(i)$,
and consider $R(i+1)$.
The algorithm increments $R(i+1)$ on a node v
iff v is reachable in $\leq i+1$ steps.
To see this, note that $R(i+1)$ is __not__ __incremented__
only if __all__ nodes at distance $\leq i$ from s were tried
and v is __not__ __reachable__ in ≤ 1 steps from any of them.
We are sure to check all nodes at distance $\leq i$
by comparing count with $R(i-1)$. $\square$

Summary:

To check that t is not reachable from s,
we first run algorithm #reach to compute N.
Then we run unreach with that N.
Since both (non-deterministic) algorithms
run in logarithmic space,
the total space required by the procedure is $O(\log n)$. $\square$