15. Immerman & Szelepcsényi's Theorem

<u>Goal</u>: Answer Kuroda's second question from 1964 (see last lecture) positively:

The context-sensitive languages are closed under complement!

<u>History</u>:
- The result was obtained independently 1988 and 1987 by
  - ↳ Neil Immerman (big fish already then, Univ. of Massachusetts Amherst)
  - ↳ Robert Szelepcsényi (student in Bratislava, Slovakia).
- Both received the Gödel-Prize 1995.
- The result brought the <u>method of inductive counting</u> to complexity theory.

<u>Theorem (Immerman & Szelepcsényi '88 '87)</u>:

If $L \subseteq \Sigma^*$ is context-sensitive, so is $\overline{L}$.

<u>Goal</u>: Let $L = \mathcal{L}(G)$ with $G = (N, \Sigma, P, S)$ a type-1 grammar.

We construct an NLBA that, upon input $w \in \Sigma^*$, accepts if there is <u>no</u> <u>derivation</u> $S \Rightarrow_G^* w$

To this end, we consider $Graph_{|w|}$, the graph of all sentential forms of length $\leq |w|$ together with the derivation relation.

Formally,
$$Graph_{|w|} := (\underbrace{(\Sigma \cup N)^{\leq |w|}}_{\text{nodes}}, \underbrace{\{(\alpha, \beta) \mid \alpha \Rightarrow_G \beta\}}_{\text{edges}}).$$

There is no derivation $S \Rightarrow_G^* w$ iff there is no path from S to w in $Graph_{|w|}$.

Hence, the key of the proof is to show that <u>non-reachability</u> in a graph of exponential size can be solved <u>non-deterministically</u> with <u>linear space</u> (we discussed that we can be slightly more liberal than only using the space given by the input).

Summary: • We have to develop an algorithm (a TM) that

• given a graph G (here $Graph_{|w|}$)

and two nodes s (here S) and t (here w)

• shows the absence of a path from s to t.

• Moreover, the algorithm should only need space (tape) logarithmic in the size of G.

In our setting, being logarithmic in the size of $G = Graph_{|w|}$ means being linear in $|w|$ (see Box *)

Hence, the algorithm is not only a TM but indeed an NLBA.

Idea: • To check that t is not reachable from s in G, enumerate all nodes that are reachable from s and check that t is not among them.

* $Graph_{|w|}$ has $c^{|w|}$ nodes and at most $(c^{|w|})^2 = c^{2|w|}$ edges, where $c = |N| + |\Sigma| + 1$. Then $\log c^{2|w|} = 2|w| \log c$.

• 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 $\boxed{N = \text{number of nodes reachable from } s}$.

We show how to compute N non-deterministically in logarithmic (in $|G|$) space below.

• Given N, the following non-deterministic algorithm

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

↳ works in space $O(\log n)$.

```
bool unreach(G,s,t)
begin
  // Given N = number of 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";      // Easy in non-
                                               // deterministic
                                               // logarithmic space:
                                               // Just store last
                                               // node and steps.
      if "guessed path does not lead to v" then    // Wrong path or
        return false;                              // wrong guess
      else if v = t then                           // Reachable
        return false;
      else                                         // Another reachable node ≠ t found
        count++;
      end if
    end if
  end for
  if count < N then        // Guessed incorrectly about reachability
    return false;          // for some v.
  else                     // Unreachable
    return true;
  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 in G.

Proof:

The algorithm makes sure it enumerates all nodes reachable from s by comparing count to N.

The algorithm accepts iff t was not one of the N nodes reachable from s. $\square$

It remains to compute

$N =$ number of nodes reachable from s.

The key idea, nowadays called method of inductive counting, is to inductively compute the values

$R(i) :=$ number of nodes reachable from s in $\leq i$ steps.

Then $N = R(n)$ (we do not have to repeat a node to solve reachability).

```
| false | N   numberReach(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 step
        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      // Guessed incorrectly
    return false;           about reachability
                            for some u.
end if
end for                     // Loop for v
end for                     // Loop for i
return R(n);
end
```

Remark:

At any point in time, Algorithm numberReach(-) 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 with logarithmic space.

Lemma:

numberReach(G,s) computes the number of nodes reachable from s in G.

Proof:

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

$$R(i) = \text{number of nodes reachable from } s \text{ in } \leq i \text{ steps.}$$

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

Induction step ($i \rightsquigarrow i+1$): Assume the equality holds for $R(i)$ and consider $R(i+1)$.
The algorithm increments $R(i+1)$ on v iff v is reachable from s 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 step from any of them.
We are sure to check all nodes at distance $\leq i$ by comparing count to $R(i)$. □

Summary:

- To check that t is not reachable from s in G, we first run algorithm numberReach(G,s) to compute N. Then we run unreach(G,s,t) with that N.
- Since both (non-deterministic) algorithms run in logarithmic space (tape), the total space required by the procedure is $\mathcal{O}(\log |G|)$.
- We argued on Pages 1 and 2 that this entails the existence of an NLBA for $\overline{L}$. $\square$