Computational Complexity Theory

Lecture 10: NL = co-NL; Polynomial Hierarchy

Department of Computer Science,
Indian Institute of Science

Recap: PSPACE-completeness

- Recall, to define completeness of a complexity class, we need an appropriate notion of a *reduction*.
- What kind of reductions will be suitable is guided by *a complexity question*, like a comparison between the complexity class under consideration & another class.
- Is P = PSPACE ? …use poly-time Karp reduction!

- Definition. A language L' is *PSPACE-hard* if for every L in PSPACE, $L \leq_p L'$. Further, if L' is in PSPACE then L' is *PSPACE-complete*.

Recap: PSPACE-complete problem

- Definition. A *quantified Boolean formula (QBF)* is a formula of the form

$$Q_1x_1\ Q_2x_2\ \ldots\ Q_nx_n\ \ \varphi(x_1, x_2, \ldots, x_n)$$

Quantifiers ∃ or ∀

Just a formula on Boolean variables

- A QBF is either *true* or *false* as all variables are quantified. This is unlike a formula we've seen before where variables were unquantified/free.

Recap: PSPACE-complete problem

- Definition. TQBF is the set of ***true*** quantified Boolean formulas.

- Theorem. TQBF is PSPACE-complete.

Recap: PSPACE-complete problem

- Definition. TQBF is the set of ***true*** quantified Boolean formulas.

- Theorem. TQBF is PSPACE-complete.

- Theorem. *(Shamir 1990; Lund, Fortnow, Karloff, Nisan 1990)* IP = PSPACE.
- IP or ***Interactive Proof*** is a grand generalization of NP proof.

Recap: NL-completeness

- Recall again, to define completeness of a complexity class, we need an appropriate notion of a *reduction*.
- What kind of reductions will be suitable is guided by *a complexity question*, like a comparison between the complexity class under consideration & another class.
- Is L = NL ? …poly-time (Karp) reductions are much too powerful for L.
- We need to define a suitable *'log-space'* reduction.

Recap: Log-space reductions

$$(x, i) \xrightarrow{\text{Log-space TM}} f(x)_i$$

- **Issue:** A log-space TM may not have enough space to write down the whole output $f(x)$ in one shot.
- **Solution:** Have the log-space TM output a bit of $f(x)$.

- **Definition:** A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is *implicitly log-space computable* if
  1. $|f(x)| \leq |x|^c$ for some constant c,
  2. The following two languages are in $\mathbf{L}$:

$$L_f = \{(x, i) : f(x)_i = 1\} \quad \text{and} \quad L'_f = \{(x, i) : i \leq |f(x)|\}$$

Recap: Log-space reductions

$$(x, i) \xrightarrow{\text{Log-space TM}} f(x)_i$$

- Issue: A log-space TM may not have enough space to write down the whole output $f(x)$ in one shot.
- Solution: Have the log-space TM output a bit of $f(x)$.

- Definition: A language L_1 is *log-space reducible* to a language L_2, denoted $L_1 \leq_l L_2$, if there's an implicitly log-space computable function f such that

$$x \in L_1 \iff f(x) \in L_2$$

Recap: Log-space reductions

$$(x, i) \xrightarrow{\text{Log-space TM}} f(x)_i$$

- **Issue:** A log-space TM may not have enough space to write down the whole output $f(x)$ in one shot.
- **Solution:** Have the log-space TM output a bit of $f(x)$.

- **Claim:** If $L_1 \leq_l L_2$ and $L_2 \leq_l L_3$ then $L_1 \leq_l L_3$.

- **Claim:** If $L_1 \leq_l L_2$ and $L_2 \in \mathsf{L}$ then $L_1 \in \mathsf{L}$.

Recap: NL-completeness

- Definition: A language L is NL-complete if L ∈ NL and for every L' ∈ NL, L' is log-space reducible to L.

  PATH = {(G,s,t) : G is a digraph having a path from s to t}.

- Theorem: PATH is NL-complete.

- Reachability in DAGs, checking if a digraph is strongly connected, and 2SAT are also NL-complete.

An alternate characterization of NL

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with no path from s to t}

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with no path from s to t}

- Definition.(first attempt) Suppose L is a language, and there's a *log-space verifier* M & a function q s.t.

$$x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u) = 1$$

Should we define q(|x|) as a log function, meaning q(|x|) = O(log |x|) ?

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

  $\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Definition.(first attempt) Suppose L is a language, and there's a *log-space verifier* M & a function q s.t.

  $$x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u) = 1$$

Should we define q(|x|) as a log function, meaning q(|x|) = O(log |x|) ?

…**No, that's too restrictive.** That will imply L ∈ L.

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Definition.(first attempt) Suppose L is a language, and there's a *log-space verifier* M & a *<u>poly-function</u>* q s.t.

$$x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u) = 1$$

Is it so that L ∈ NL iff L has such a log-space verifier of the above kind?

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Definition.(first attempt) Suppose L is a language, and there's a *log-space verifier* M & a *poly-function* q s.t.

$$x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u) = 1$$

Is it so that $L \in$ NL iff L has such a log-space verifier of the above kind?
Unfortunately not!! Exercise: $L \in$ NP iff L has such a log-space verifier.

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Definition.(first attempt) Suppose L is a language, and there's a *log-space verifier* M & a *poly-function* q s.t.

$$x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u) = 1$$

Solution: Make the certificate ***<u>read-one</u>*** as described next…

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

  $\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Definition. A tape is called a *read-once tape* if the head moves from left to right and <u>never turns back</u>.

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with no path from s to t}

- Definition. A language L has *read-once certificates* if there's a *log-space verifier* M & a *poly-function* q s.t.

$$x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u) = 1,$$

where u is given on a read-once input tape of M.

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

  $\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Theorem. L ∈ NL iff L has read-once certificates.

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Theorem. L ∈ NL iff L has read-once certificates.
- Proof. Suppose L ∈ NL. Let N be an NTM that decides L. Think of a verifier M that on input (x, u) simulates N on input x by using u as the nondeterministic choices of N. Clearly |u| = poly(|x|)...

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Theorem. L ∈ NL iff L has read-once certificates.
- Proof. (contd.) …as $G_{N,x}$ has poly(|x|) configurations. M scans u from left to right without moving its head backward. So, u is a read-once certificate satisfying,

$$x \in L \quad \Longleftrightarrow \quad \exists u \in \{0,1\}^{poly(|x|)} \text{ s.t. } M(x,u) = 1$$

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

  $\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Theorem. L ∈ NL iff L has read-once certificates.
- Proof. (contd.) Suppose L has read-once certificates, and M be a log-space verifier s.t.

  $x \in L \iff \exists u \in \{0,1\}^{q(|x|)}$ s.t. $M(x,u) = 1$.

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Theorem. L ∈ NL iff L has read-once certificates.
- Proof. (contd.) Now, think of an NTM N that on input x starts simulating M. It guesses the bits of u as and when required during the simulation. As u is read-once for M, there's no need for N to store u.

Certificate definition of NL

- Like NP, it will be useful to have a *certificate-verifier* kind of definition for the class NL.
- We'll see how it helps in proving NL = co-NL i.e., in showing $\overline{\text{PATH}} \in$ NL.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with <u>no</u> path from s to t}

- Theorem. L ∈ NL iff L has read-once certificates.
- Proof. (contd.) So, N is a log-space NTM deciding L.

NL = co-NL

Class co-NL

- Definition. A language L is in co-NL if $\overline{L} \in$ NL. L is co-NL-complete if L ∈ co-NL and for every L' ∈ co-NL, L' is log-space reducible to L.

  $\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with no path from s to t}
- Obs. $\overline{\text{PATH}}$ is co-NL-complete under log-space reduction.

Class co-NL

- Definition. A language L is in co-NL if $\overline{L} \in$ NL. L is co-NL-complete if $L \in$ co-NL and for every $L' \in$ co-NL, L' is log-space reducible to L.

$\overline{\text{PATH}}$ = {(G,s,t): G is a digraph with no path from s to t}

- Obs. $\overline{\text{PATH}}$ is co-NL-complete under log-space reduction.

- Obs. If a language L' log-space reduces to a language in NL then $L' \in$ NL. (*Homework*) So, if $\overline{\text{PATH}} \in$ NL then NL = co-NL.

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- Proof. It is sufficient to show that there's a *log-space verifier* M & a *poly-function* q s.t.

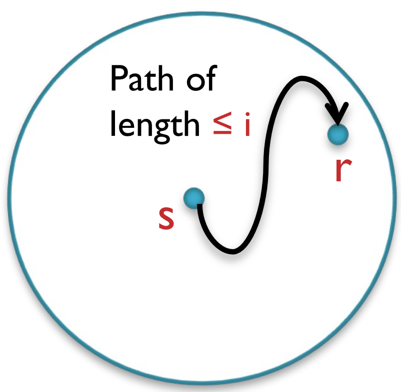
  $x \in \overline{\text{PATH}} \iff \exists u \in \{0,1\}^{q(|x|)}$ s.t. $M(x,u) = 1$,

  where u is given on a read-once input tape of M.

- Let us focus on forming a read-once certificate u that convinces a verifier that there's no path from s to t…

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- Proof. $x = (G,s,t)$. Let m be the number of nodes in G. Let k_i = no. of nodes reachable from s by a path of length at most i in G.



k_i nodes

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- Proof. $x = (G,s,t)$. Let m be the number of nodes in G. Let k_i = no. of nodes reachable from s by a path of length at most i in G.

Read-once certificate u is of the form $(u_1, u_2, \ldots, u_m, v)$, where u_i's and v are strings s.t.

(1) reading until $(u_1, u_2, \ldots u_i)$ in a read-once fashion, M knows correctly the value of k_i.

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- Proof. $x = (G,s,t)$. Let m be the number of nodes in G. Let k_i = no. of nodes reachable from s by a path of length at most i in G.

Read-once certificate u is of the form $(u_1, u_2, \ldots, u_m, v)$, where u_i's and v are strings s.t.

(1) reading until $(u_1, u_2, \ldots u_i)$ in a read-once fashion, M knows correctly the value of k_i. So, after reading $(u_1, u_2, \ldots u_m)$, M knows k_m, the number of nodes reachable from s.

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- Proof. $x = (G,s,t)$. Let m be the number of nodes in G. Let k_i = no. of nodes reachable from s by a path of length at most i in G.

Read-once certificate u is of the form $(u_1, u_2, \ldots, u_m, v)$, where u_i's and v are strings s.t.

(1) reading until $(u_1, u_2, \ldots u_i)$ in a read-once fashion, M knows correctly the value of k_i. So, after reading $(u_1, u_2, \ldots u_m)$, M knows k_m, the number of nodes reachable from s.

(2) v then convinces M (which already knows k_m) that t is not one of the k_m vertices reachable from s.

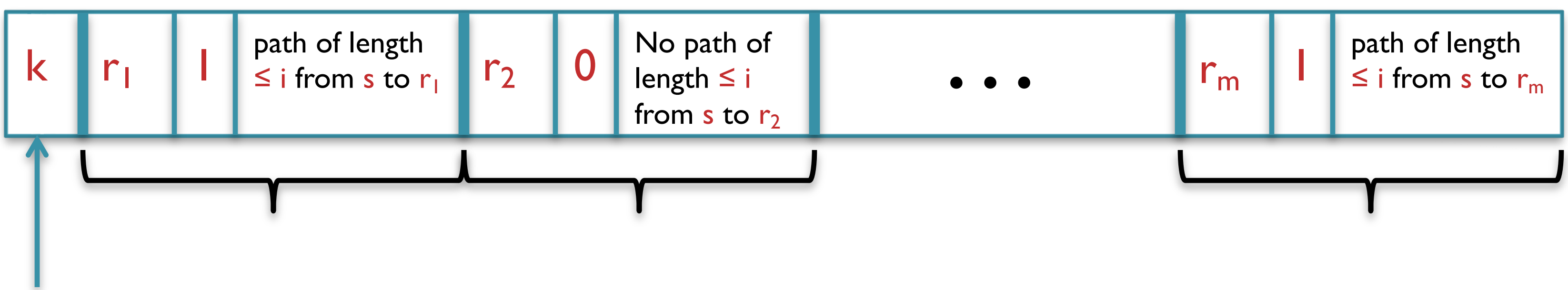
NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- Proof. We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- Proof. We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

u_i looks like:



The claimed value of k_i.
O(log m) bits required.

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{PATH} \in NL$.
- Proof. We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

u_i looks like:

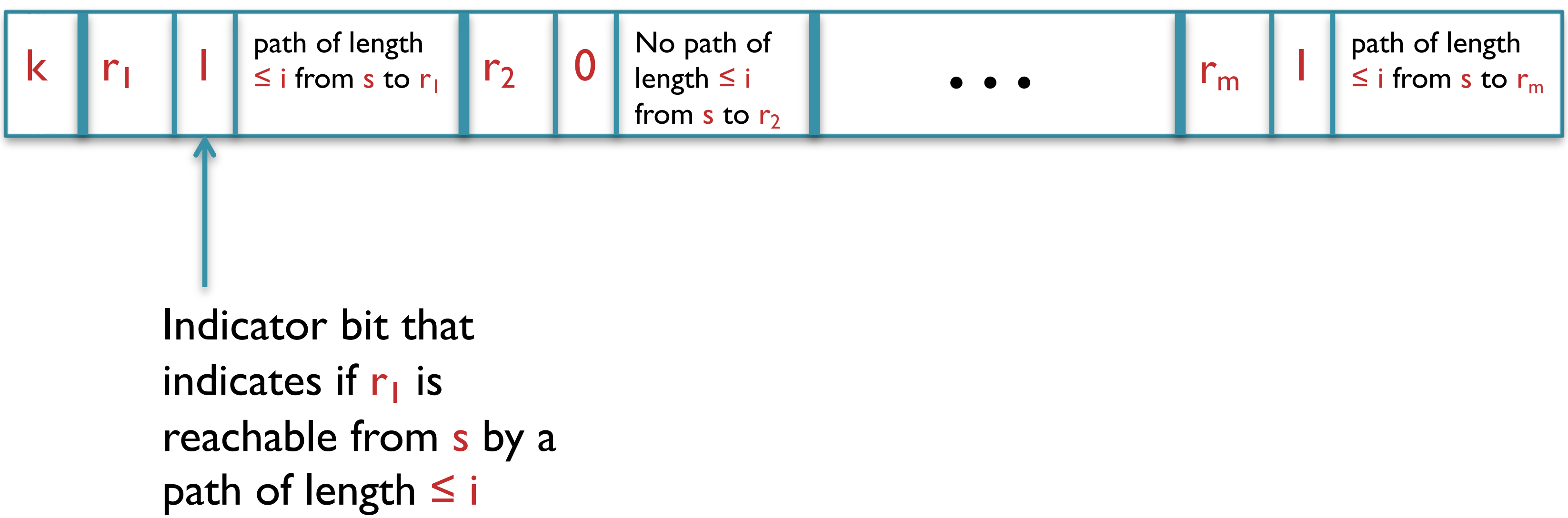
k	r_1	1	path of length $\leq i$ from s to r_1	r_2	0	No path of length $\leq i$ from s to r_2	• • •	r_m	1	path of length $\leq i$ from s to r_m

Index of a vertex.
$O(\log m)$ bits required.

NL = co-NL

- **Theorem.** (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- **Proof.** We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

u_i looks like:



NL = co-NL

- **Theorem.** (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- **Proof.** We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots . < r_m$. Then,

u_i looks like:

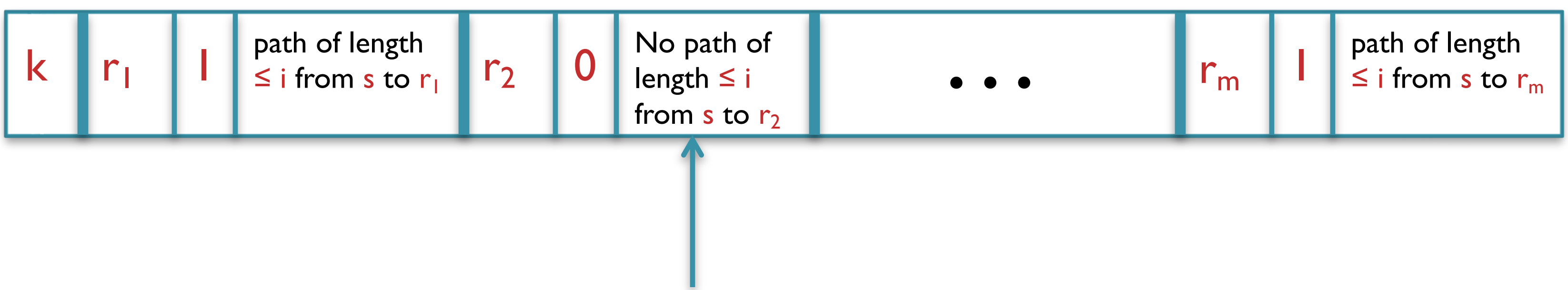
k	r_1	1	path of length $\leq i$ from s to r_1	r_2	0	No path of length $\leq i$ from s to r_2	. . .	r_m	1	path of length $\leq i$ from s to r_m

If indicator bit is 1 then give a path from s to r_1 of length $\leq i$. $O(m \log m)$ bits required for this.

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- Proof. We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

u_i looks like:

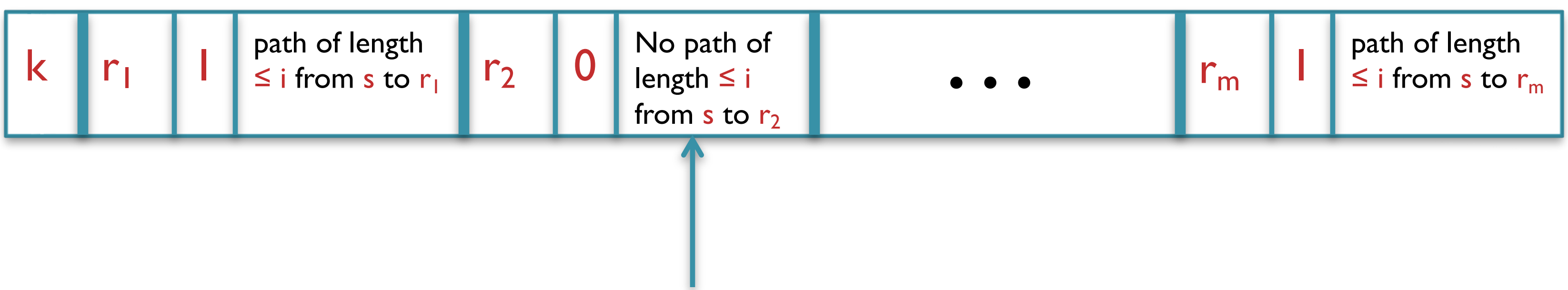


If indicator bit is 0 then give a certificate for absence of paths from s to r_2 of length $\leq i$. (how?)

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- Proof. We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

u_i looks like:



If indicator bit is 0 then give a certificate for absence of paths from s to r_2 of length $\leq i$. (how?)

NL = co-NL

- **Theorem.** (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- **Proof.** We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

u_i looks like:

k	r_1	1	path of length $\leq i$ from s to r_1	r_2	0	No path of length $\leq i$ from s to r_2	. . .	r_m	1	path of length $\leq i$ from s to r_m

If indicator bit is 0 then give a certificate for absence of paths from s to r_2 of length $\leq i$. (how?)

If such certificates can be given using poly(m) bits then $|u_i| = \text{poly}(m)$

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- Proof. We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

u_i looks like:

k	r_1	1	path of length $\leq i$ from s to r_1	r_2	0	No path of length $\leq i$ from s to r_2	. . .	r_m	1	path of length $\leq i$ from s to r_m

- While reading u_i, M's work tape remembers the following info:

    1. k_{i-1} and k,
    2. the last read index of a vertex r_j

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- Proof. We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

u_i looks like:

k	r_1	1	path of length $\leq i$ from s to r_1	r_2	0	No path of length $\leq i$ from s to r_2	. . .	r_m	1	path of length $\leq i$ from s to r_m

- While reading u_i, M's work tape remembers the following info:

The moment M encounters a new vertex index r, it checks immediately if $r > r_j$. This ensures that M is not fooled by repeating info about the same vertex in u_i.

1. k_{i-1} and k,
2. the last read index of a vertex r_j

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- Proof. We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

u_i looks like:

k	r_1	1	path of length $\leq i$ from s to r_1	r_2	0	No path of length $\leq i$ from s to r_2	. . .	r_m	1	path of length $\leq i$ from s to r_m

- While reading u_i, M's work tape remembers the following info:

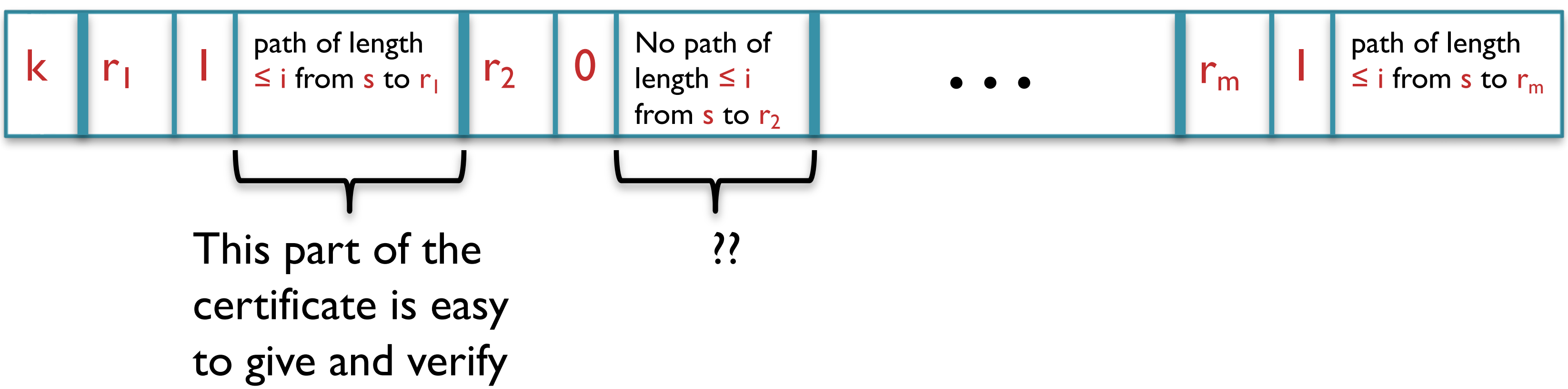
  1. k_{i-1} and k,
  2. the last read index of a vertex r_j

While reading u_i, M keeps a count of the number of indicator bits that are 1 and finally checks if this number is k.

NL = co-NL

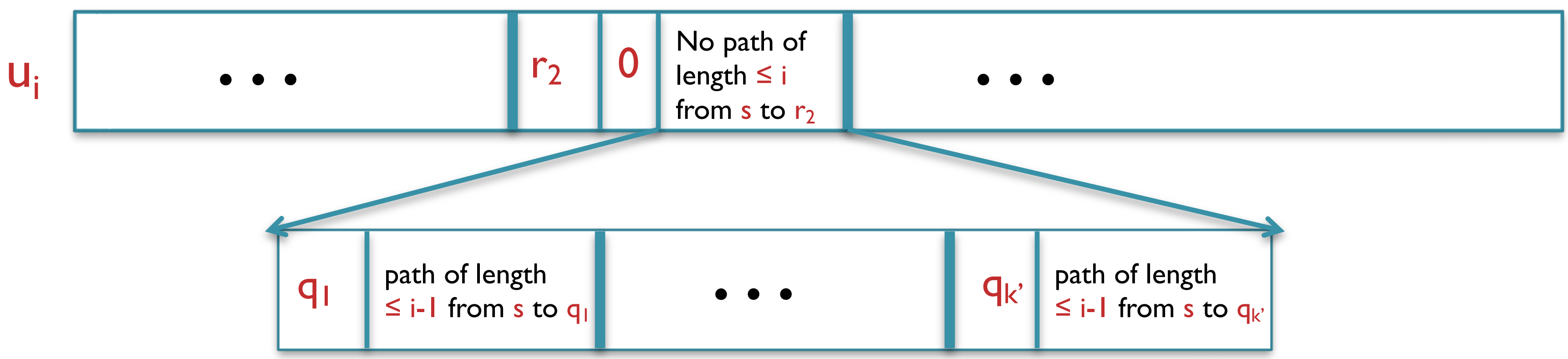
- **Theorem.** (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- **Proof.** We'll design u_i assuming that $u_1, \ldots, u_{i-1}$ have already been constructed and M knows k_{i-1}. Let $r_1, \ldots r_m$ be the nodes of G s.t. $r_1 < r_2 < \ldots < r_m$. Then,

u_i looks like:



NL = co-NL

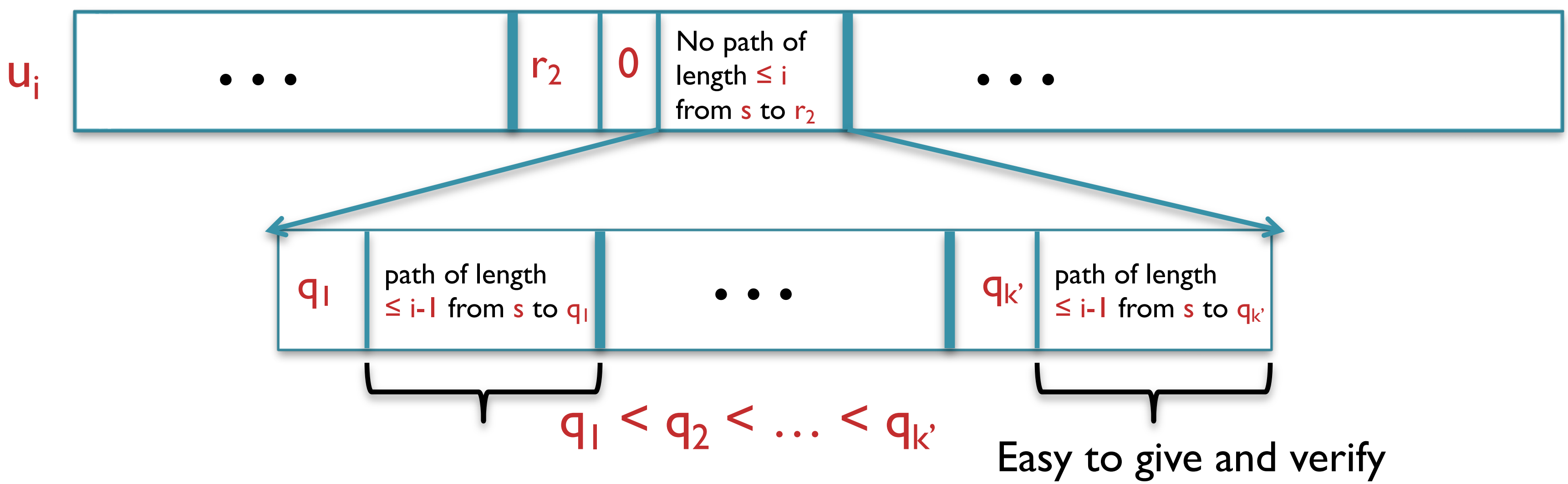
- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- Proof. Recall, M knows $k_{i-1} = k'$ (say) while reading u_i.



$q_1 < q_2 < \ldots < q_{k'}$

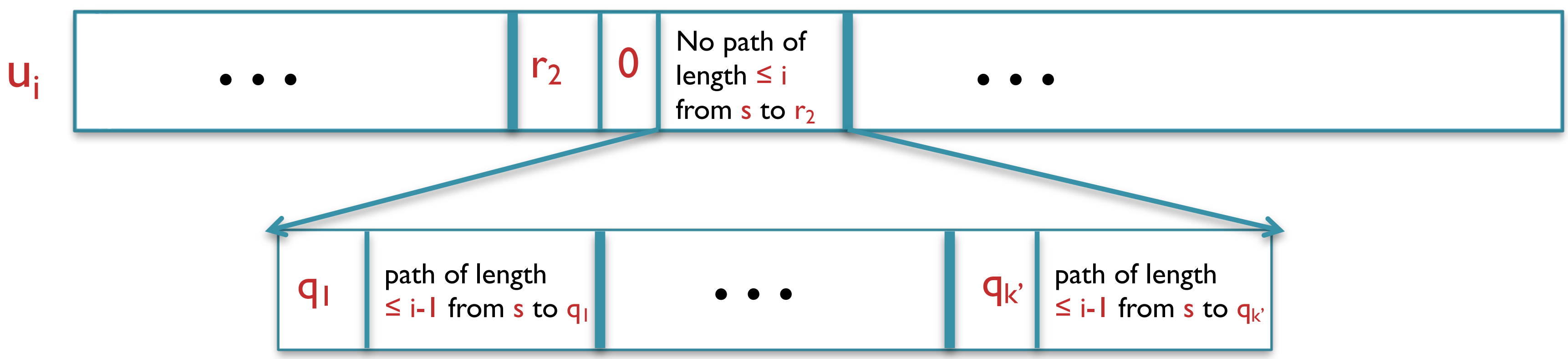
NL = co-NL

- **Theorem.** (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- **Proof.** Recall, M knows $k_{i-1} = k'$ (say) while reading u_i.



NL = co-NL

- **Theorem.** (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- **Proof.** Recall, M knows $k_{i-1} = k'$ (say) while reading u_i.

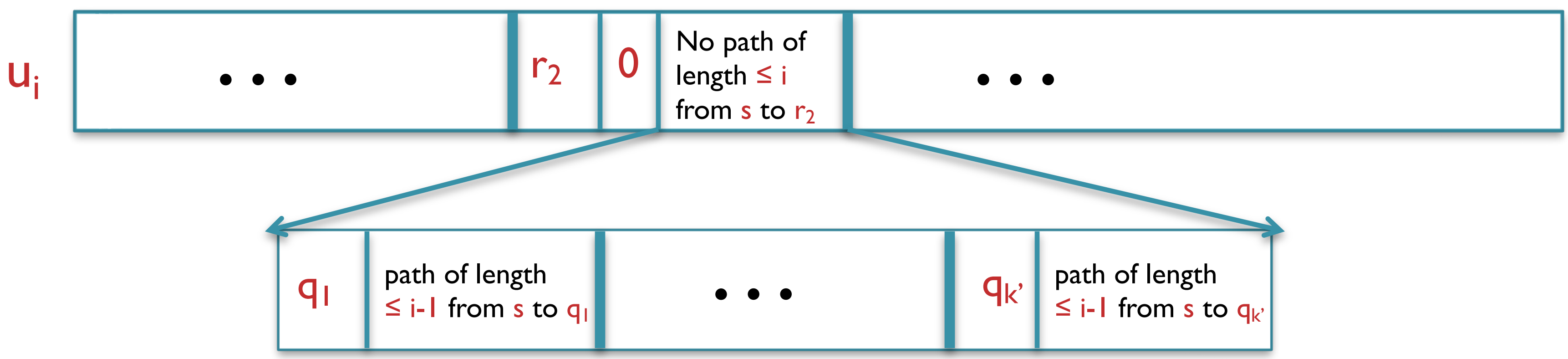


$q_1 < q_2 < \ldots < q_{k'}$

- While reading the 'No path…r_2' part of u_i, M remembers the last q_j read and checks that the next $q > q_j$. This ensures M is not fooled by repeating q's.

NL = co-NL

- **Theorem.** (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- **Proof.** Recall, M knows $k_{i-1} = k'$ (say) while reading u_i.

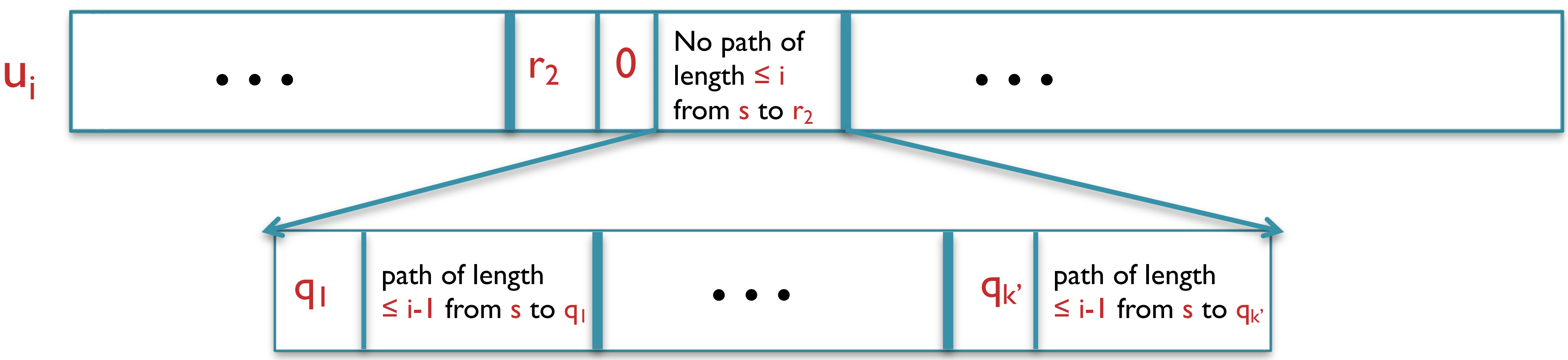


$q_1 < q_2 < \ldots < q_{k'}$

- For every $j \in [1, k_{i-1}]$, after verifying the path of length $\leq i-1$ from s to q_j, M checks that r_2 is <u>not</u> adjacent to q_j by looking at G's adjacency matrix.

NL = co-NL

- **Theorem.** (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- **Proof.** Recall, M knows $k_{i-1} = k'$ (say) while reading u_i.

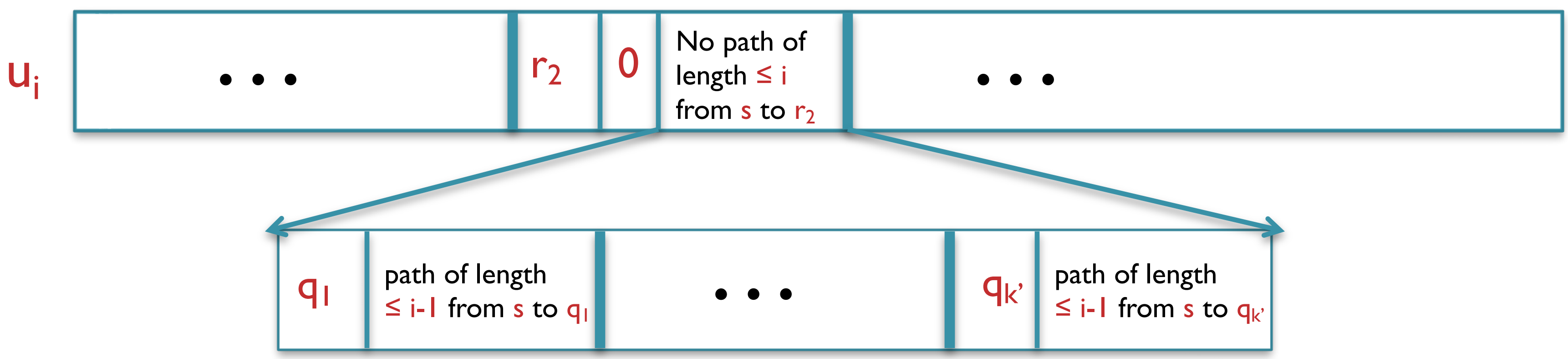


$$q_1 < q_2 < \ldots < q_{k'}$$

- At the end of reading the 'No path…r_2' part, M checks that the number of q's read is exactly k_{i-1}.

NL = co-NL

- **Theorem.** (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in$ NL.
- **Proof.** Recall, M knows $k_{i-1} = k'$ (say) while reading u_i.



$q_1 < q_2 < \ldots < q_{k'}$

- This convinces M that there is no path of length ≤ i from s to r_2. Length of the 'No path…r_2' part of u_i is $O(m^2 \log m)$.

NL = co-NL

- Theorem. (*Immerman-Szelepcsenyi 1987*) $\overline{\text{PATH}} \in \text{NL}$.
- Proof. So, after reading $(u_1, \ldots, u_m)$, the verifier M knows k_m, the number of vertices reachable from s.

- The v part of the certificate u is similar to the 'No path…r_2' part of u_i described before. The details here are easy to fill in (*homework*).

- We stress again that M is able to verify nonexistence of a path between s and t by <u>reading u once</u> from left to right and never moving its head backward.

NL = co-NL

- Hence, both PATH and

  $\text{PATH} \in \text{NL} \subseteq \text{SPACE}((\log n)^2)$

  by Savitch's theorem.

Polynomial Hierarchy

Problems between NP & PSPACE

- There are decision problems that don't appear to be captured by nondeterminism alone (i.e., with a **single** $\exists$ or $\forall$ quantifier), unlike problems in NP and co-NP.

- Example.

  Eq-DNF = {(φ,k): φ is a **DNF** and <u>there's</u> a DNF ψ of size $\leq$ k that is <u>equivalent</u> to φ}

- Two Boolean formulas on the same input variables are *equivalent* if their evaluations agree on every assignment to the variables.

Problems between NP & PSPACE

- There are decision problems that don't appear to be captured by nondeterminism alone (i.e., with a **single** ∃ or ∀ quantifier), unlike problems in NP and co-NP.

- Example.

  Eq-DNF = {(φ,k): φ is a **DNF** and there's a DNF ψ of size ≤ k that is equivalent to φ}

- Is Eq-DNF in NP? …if we give a DNF ψ as a certificate, it is not clear how to efficiently verify that ψ and φ are equivalent. (W.l.o.g. $k \leq$ size of φ .)

Class Σ_2

- Definition. A language L is in Σ_2 if there's a polynomial function q(.) and a poly-time TM M (the "verifier") s.t.

$$x \in L \iff \exists u \in \{0,1\}^{q(|x|)}\ \forall v \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u,v) = 1.$$

Class Σ_2

- Definition. A language L is in Σ_2 if there's a polynomial function q(.) and a poly-time TM M (the "verifier") s.t.

$$x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \; \forall v \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u,v) = 1.$$

- Obs. Eq-DNF is in Σ_2.
- Proof. Think of u as another DNF ψ and v as an assignment to the variables. Poly-time TM M checks if ψ has size $\leq k$ and $\varphi(v) = \psi(v)$.

Class Σ_2

- Definition. A language L is in Σ_2 if there's a polynomial function q(.) and a poly-time TM M (the "verifier") s.t.

  $x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \ \forall v \in \{0,1\}^{q(|x|)}$ s.t. $M(x,u,v) = 1$.

- Obs. Eq-DNF is in Σ_2.
- Proof. Think of u as another DNF ψ and v as an assignment to the variables. Poly-time TM M checks if ψ has size $\leq k$ and $\varphi(v) = \psi(v)$.

- Remark. *(Masek 1979)* Even if φ is given by its truth-table, the problem (i.e., DNF-MCSP) is NP-complete.

Class $\sum_2$

- Definition. A language L is in $\sum_2$ if there's a polynomial function q(.) and a poly-time TM M (the "verifier") s.t.

$$x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \ \forall v \in \{0,1\}^{q(|x|)} \text{ s.t. } M(x,u,v) = 1.$$

- Another example.

Succinct-SetCover = $\{(\varphi_1,\ldots\varphi_m,k)$: φ_i's are DNFs and there's an $S \subseteq [m]$ of size $\leq k$ s.t. $\vee_{i\in S}\, \varphi_i$ is a tautology$\}$

Class Σ_2

- Definition. A language L is in Σ_2 if there's a polynomial function q(.) and a poly-time TM M (the "verifier") s.t.

  $x \in L \Leftrightarrow \exists u \in \{0,1\}^{q(|x|)} \ \forall v \in \{0,1\}^{q(|x|)}$ s.t. $M(x,u,v) = 1$.

- Obs. *(Homework)* Succinct-SetCover is in Σ_2.

Class Σ_2

- Definition. A language L is in Σ_2 if there's a polynomial function q(.) and a poly-time TM M (the "verifier") s.t.

  $x \in L \iff \exists u \in \{0,1\}^{q(|x|)}\ \forall v \in \{0,1\}^{q(|x|)}$ s.t. $M(x,u,v) = 1$.

- Obs. *(Homework)* Succinct-SetCover is in Σ_2.

- Other natural problems in PH: *"Completeness in the Polynomial-Time Hierarchy: A Compendium"* by *Schaefer and Umans (2008)*.

Class Σ_2

- Definition. A language L is in Σ_2 if there's a polynomial function q(.) and a poly-time TM M (the "verifier") s.t.

  $x \in L \iff \exists u \in \{0,1\}^{q(|x|)} \ \forall v \in \{0,1\}^{q(|x|)}$ s.t. $M(x,u,v) = 1$.

- Obs. $P \subseteq NP \subseteq \Sigma_2$.

Class Σ_i

- Definition. A language L is in Σ_i if there's a polynomial function q(.) and a poly-time TM M (the "verifier") s.t.

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{q(|x|)} \ \forall u_2 \in \{0,1\}^{q(|x|)} \ Q_i u_i \in \{0,1\}^{q(|x|)}$$

$$\text{s.t. } M(x,u_1,\ldots,u_i) = 1,$$

where Q_i is $\exists$ or $\forall$ if i is odd or even, respectively.

- Obs. $\Sigma_i \subseteq \Sigma_{i+1}$ for every i.

Polynomial Hierarchy

- **Definition.** A language L is in Σ_i if there's a polynomial function $q(.)$ and a poly-time TM M (the "verifier") s.t.

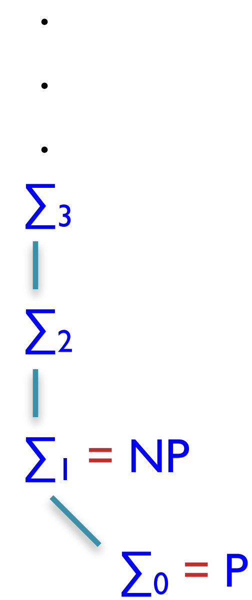
$$x \in L \iff \exists u_1 \in \{0,1\}^{q(|x|)} \ \forall u_2 \in \{0,1\}^{q(|x|)} \ Q_i u_i \in \{0,1\}^{q(|x|)}$$

$$\text{s.t. } M(x, u_1, \ldots, u_i) = 1,$$

where Q_i is $\exists$ or $\forall$ if i is odd or even, respectively.

- **Definition.** *(Meyer & Stockmeyer 1972)*

$$PH = \bigcup_{i \in \mathbf{N}} \Sigma_i .$$



Class Π_i

- Definition. Π_i = co-Σ_i = $\{ L : \bar{L} \in \Sigma_i \}$.

- Obs. A language L is in Π_i if there's a polynomial function q(.) and a poly-time TM M (the "verifier") s.t.

$$x \in L \iff \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)}\ \cdots\ Q_i u_i \in \{0,1\}^{q(|x|)}$$

$$\text{s.t. } M(x,u_1,\ldots,u_i) = 1,$$

where Q_i is $\forall$ or $\exists$ if i is odd or even, respectively.

Class Π_i

- Definition. Π_i = co-Σ_i = $\{ L : \overline{L} \in \Sigma_i \}$.

- Obs. A language L is in Π_i if there's a polynomial function q(.) and a poly-time TM M (the "verifier") s.t.

  $x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{q(|x|)}\ \exists u_2 \in \{0,1\}^{q(|x|)}\ \ Q_i u_i \in \{0,1\}^{q(|x|)}$

  s.t. $M(x,u_1,\ldots, u_i) = 1$,

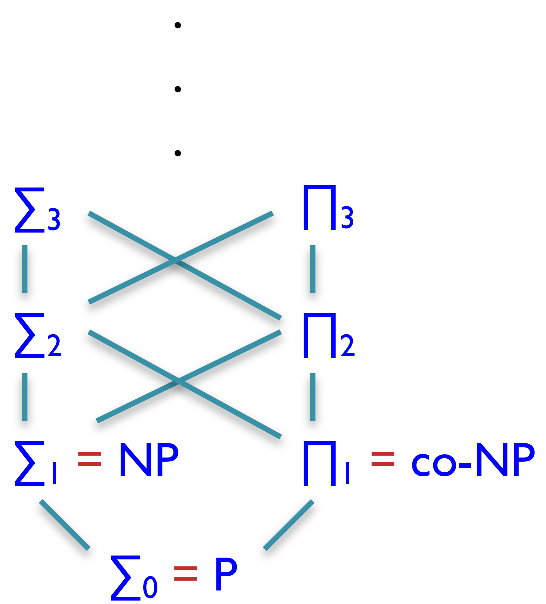
  where Q_i is $\forall$ or $\exists$ if i is odd or even, respectively.

- Obs. $\Sigma_i \subseteq \Pi_{i+1} \subseteq \Sigma_{i+2}$.

Polynomial Hierarchy

- Obs. PH $= \bigcup_{i \in \mathbf{N}} \Sigma_i = \bigcup_{i \in \mathbf{N}} \Pi_i$.

PH =



Polynomial Hierarchy

- Claim. PH $\subseteq$ PSPACE .
- Proof. Similar to the proof of TQBF $\in$ PSPACE.