Complement Space Complexity

Nabil Mustafa

Computational Complexity

Complement Space Classes

Definition

$$\mathbf{coPSPACE} = \{L : \overline{L} \in \mathbf{PSPACE}\ \}$$

Complement Space Classes

Definition

$$\mathbf{coPSPACE} = \{L : \overline{L} \in \mathbf{PSPACE}\}$$

$$\mathbf{coL} = \{L : \overline{L} \in \mathbf{L}\}$$

Complement Space Classes

Definition

$$\textbf{coPSPACE} = \{L : \overline{L} \in \textbf{PSPACE}\}$$

$$\textbf{coL} = \{L : \overline{L} \in \textbf{L}\}$$

Claim: $\textbf{coPSPACE} = \textbf{PSPACE}$

Complement Space Classes

Definition

$$\mathbf{coPSPACE} = \{L : \overline{L} \in \mathbf{PSPACE}\ \}$$

$$\mathbf{coL} = \{L : \overline{L} \in \mathbf{L}\ \}$$

Claim: $\mathbf{coPSPACE} = \mathbf{PSPACE}$

Claim: $\mathbf{coL} = \mathbf{L}$

Complement Space Classes

Definition

$$\mathbf{coPSPACE} = \{L : \overline{L} \in \mathbf{PSPACE}\}$$

$$\mathbf{coL} = \{L : \overline{L} \in \mathbf{L}\}$$

Claim: $\mathbf{coPSPACE} = \mathbf{PSPACE}$

Claim: $\mathbf{coL} = \mathbf{L}$

- All deterministic classes are closed under complement.

Complement Space Classes

Definition

$$\mathbf{coPSPACE} = \{L : \overline{L} \in \mathbf{PSPACE}\}$$

$$\mathbf{coL} = \{L : \overline{L} \in \mathbf{L}\}$$

Claim: $\mathbf{coPSPACE} = \mathbf{PSPACE}$

Claim: $\mathbf{coL} = \mathbf{L}$

- All deterministic classes are closed under complement.

Definition

$$\mathbf{coNSPACE} = \{L : \overline{L} \in \mathbf{NSPACE}\}$$

Complement Space Classes

Definition

$$\textbf{coPSPACE} = \{L : \overline{L} \in \textbf{PSPACE}\}$$

$$\textbf{coL} = \{L : \overline{L} \in \textbf{L}\}$$

Claim: $\textbf{coPSPACE} = \textbf{PSPACE}$

Claim: $\textbf{coL} = \textbf{L}$

- All deterministic classes are closed under complement.

Definition

$$\textbf{coNSPACE} = \{L : \overline{L} \in \textbf{NSPACE}\}$$

Claim: $\textbf{coNSPACE} = \textbf{NSPACE}$

Complement Space Classes

Definition

$$\textbf{coPSPACE} = \{L : \overline{L} \in \textbf{PSPACE}\}$$

$$\textbf{coL} = \{L : \overline{L} \in \textbf{L}\}$$

Claim: $\textbf{coPSPACE} = \textbf{PSPACE}$

Claim: $\textbf{coL} = \textbf{L}$

- All deterministic classes are closed under complement.

Definition

$$\textbf{coNSPACE} = \{L : \overline{L} \in \textbf{NSPACE}\}$$

Claim: $\textbf{coNSPACE} = \textbf{NSPACE}$

- By Savitch's theorem

Complement Space Classes

Definition

$$\mathbf{coPSPACE} = \{L : \overline{L} \in \mathbf{PSPACE}\}$$

$$\mathbf{coL} = \{L : \overline{L} \in \mathbf{L}\}$$

Claim: $\mathbf{coPSPACE} = \mathbf{PSPACE}$

Claim: $\mathbf{coL} = \mathbf{L}$

- All deterministic classes are closed under complement.

Definition

$$\mathbf{coNSPACE} = \{L : \overline{L} \in \mathbf{NSPACE}\}$$

Claim: $\mathbf{coNSPACE} = \mathbf{NSPACE}$

- By Savitch's theorem – $\mathbf{NSPACE} = \mathbf{PSPACE}$

The Class **coNL**

Definition

$$\textbf{coNL} = \{L : \overline{L} \in \textbf{NL}\}$$

The Class **coNL**

Definition

$$\textbf{coNL} = \{L : \overline{L} \in \textbf{NL}\}$$

Immerman-Szelepscenyi Theorem

$$\textbf{coNL} = \textbf{NL}$$

The Class **coNL**

Definition

$$\textbf{coNL} = \{L : \overline{L} \in \textbf{NL}\}$$

Immerman-Szelepscenyi Theorem

$$\textbf{coNL} = \textbf{NL}$$

The following two claims are exactly similar to **coNP** & **NP** proofs:

Claim: A language $\overline{L}$ is **coNL** complete iff L is **NL** complete

The Class **coNL**

Definition

$$\mathbf{coNL} = \{L : \overline{L} \in \mathbf{NL}\}$$

Immerman-Szelepscenyi Theorem

$$\mathbf{coNL} = \mathbf{NL}$$

The following two claims are exactly similar to **coNP** & **NP** proofs:

Claim: A language $\overline{L}$ is **coNL** complete iff L is **NL** complete

Claim: If L is **coNL** complete, and $L \in \mathbf{NL}$, then $\mathbf{coNL} = \mathbf{NL}$

The Class **coNL**

Definition

$$\textbf{coNL} = \{L : \overline{L} \in \textbf{NL}\}$$

Immerman-Szelepscenyi Theorem

$$\textbf{coNL} = \textbf{NL}$$

The following two claims are exactly similar to **coNP** & **NP** proofs:

Claim: A language $\overline{L}$ is **coNL** complete iff L is **NL** complete

Claim: If L is **coNL** complete, and $L \in$ **NL**, then **coNL** $=$ **NL**

Claim: `UNREACHABILITY` is **coNL** complete

The Class **coNL**

Definition

$$\textbf{coNL} = \{L : \overline{L} \in \textbf{NL}\}$$

Immerman-Szelepscenyi Theorem

$$\textbf{coNL} = \textbf{NL}$$

The following two claims are exactly similar to **coNP** & **NP** proofs:

Claim: A language $\overline{L}$ is **coNL** complete iff L is **NL** complete

Claim: If L is **coNL** complete, and $L \in$ **NL**, then **coNL** $=$ **NL**

Claim: UNREACHABILITY is **coNL** complete

Claim: If UNREACHABILITY $\in$ **NL**, then **coNL** $=$ **NL**

UNREACHABILITY is in **NL**

Claim

$$\mathit{UNREACHABILITY} \in \mathbf{NL}$$

UNREACHABILITY is in **NL**

Claim

$$\mathtt{UNREACHABILITY} \in \mathbf{NL}$$

- Given a graph $G = (V, E)$, $|V| = n$, and two vertices u and v

UNREACHABILITY is in **NL**

Claim

$$\textit{UNREACHABILITY} \in \textbf{NL}$$

- Given a graph $G = (V, E)$, $|V| = n$, and two vertices u and v
  - UNREACHABILITY : Is v unreachable from u in G?

UNREACHABILITY is in **NL**

Claim

$$\mathit{UNREACHABILITY} \in \mathbf{NL}$$

- Given a graph $G = (V, E)$, $|V| = n$, and two vertices u and v
  - UNREACHABILITY : Is v unreachable from u in G?
- Construct a **NTM** N such that

UNREACHABILITY is in **NL**

Claim

$$\mathit{UNREACHABILITY} \in \textbf{NL}$$

- Given a graph $G = (V, E)$, $|V| = n$, and two vertices u and v
  - UNREACHABILITY : Is v unreachable from u in G?
- Construct a **NTM** N such that
  - N uses $O(\log n)$ space

UNREACHABILITY is in **NL**

Claim

$$\mathit{UNREACHABILITY} \in \mathbf{NL}$$

- Given a graph $G = (V, E)$, $|V| = n$, and two vertices u and v
  - UNREACHABILITY : Is v unreachable from u in G?
- Construct a **NTM** N such that
  - N uses $O(\log n)$ space
  - If there does not exist a path from u to v, then some sequence of transition rules will halt with an 'accept'.

UNREACHABILITY is in **NL**

Claim

$$\mathit{UNREACHABILITY} \in \mathbf{NL}$$

- Given a graph $G = (V, E)$, $|V| = n$, and two vertices u and v
  - UNREACHABILITY : Is v unreachable from u in G?
- Construct a **NTM** N such that
  - N uses $O(\log n)$ space
  - If there does not exist a path from u to v, then some sequence of transition rules will halt with an 'accept'.
  - If there exists a path from u to v, then *all* sequence of transition rules should halt with a 'reject'.

UNREACHABILITY is in **NL**

Claim

$$\mathit{UNREACHABILITY} \in \mathbf{NL}$$

- Given a graph $G = (V, E)$, $|V| = n$, and two vertices u and v
  - UNREACHABILITY : Is v unreachable from u in G?
- Construct a **NTM** N such that
  - N uses $O(\log n)$ space
  - If there does not exist a path from u to v, then some sequence of transition rules will halt with an 'accept'.
  - If there exists a path from u to v, then *all* sequence of transition rules should halt with a 'reject'.
- Once again, it's not as simple as saying: run the **NTM** M for REACHABILITY , and just invert its answer.

A Counting Problem $\texttt{CC}$

A Counting Problem

$\texttt{CC}$: Given $G = (V, E)$ and a vertex u, count the number of vertices reachable from v in G.

A Counting Problem $\mathtt{CC}$

A Counting Problem

$\mathtt{CC}$: Given $G = (V, E)$ and a vertex u, count the number of vertices reachable from v in G.

- Have not formally described **TM** for non-boolean functions

A Counting Problem $\mathtt{CC}$

A Counting Problem

$\mathtt{CC}$: Given $G = (V, E)$ and a vertex u, count the number of vertices reachable from v in G.

- Have not formally described **TM** for non-boolean functions
- For a **TM** it is obvious:
  - Put the output on the work-tape before halting.
  - Write-only output tape space does not matter.

A Counting Problem CC

A Counting Problem

CC : Given $G = (V, E)$ and a vertex u, count the number of vertices reachable from v in G.

- Have not formally described **TM** for non-boolean functions
- For a **TM** it is obvious:
  - Put the output on the work-tape before halting.
  - Write-only output tape space does not matter.
- For a **NTM** it is less clear:

A Counting Problem $\mathtt{CC}$

A Counting Problem

$\mathtt{CC}$: Given $G = (V, E)$ and a vertex u, count the number of vertices reachable from v in G.

- Have not formally described **TM** for non-boolean functions
- For a **TM** it is obvious:
  - Put the output on the work-tape before halting.
  - Write-only output tape space does not matter.
- For a **NTM** it is less clear:
  - The **NTM** ends in many ways, with possibly different answers

A Counting Problem CC

A Counting Problem

CC : Given $G = (V, E)$ and a vertex u, count the number of vertices reachable from v in G.

- Have not formally described **TM** for non-boolean functions
- For a **TM** it is obvious:
  - Put the output on the work-tape before halting.
  - Write-only output tape space does not matter.
- For a **NTM** it is less clear:
  - The **NTM** ends in many ways, with possibly different answers
  - So, what could be a consistent definition of the output?

A Counting Problem $\texttt{CC}$

A Counting Problem

$\texttt{CC}$: Given $G = (V, E)$ and a vertex u, count the number of vertices reachable from v in G.

- Have not formally described **TM** for non-boolean functions
- For a **TM** it is obvious:
  - Put the output on the work-tape before halting.
  - Write-only output tape space does not matter.
- For a **NTM** it is less clear:
  - The **NTM** ends in many ways, with possibly different answers
  - So, what could be a consistent definition of the output?
- A **NTM** N computes a non-boolean function f iff
  - All sequences that halt with an 'accept' must have the same output string
  - All other sequences must halt with a 'reject' state

Proof of **coNL** = **NL**

Claim

If $\mathit{CC} \in$ ***NL*** , *then* $\mathit{UNREACHABILITY} \in$ ***NL***

Proof of **coNL** = **NL**

Claim

If $\mathtt{CC} \in \mathbf{NL}$ *, then* $\mathtt{UNREACHABILITY} \in \mathbf{NL}$

Set tsum = number of nodes reachable from u

Proof of **coNL** = **NL**

Claim

If $\mathtt{CC} \in$ ***NL*** *, then* $\mathtt{UNREACHABILITY} \in$ ***NL***

Set tsum = number of nodes reachable from u
Set counter = 0

Proof of **coNL** = **NL**

Claim

If $\mathtt{CC} \in$ ***NL*** *, then* $\mathtt{UNREACHABILITY} \in$ ***NL***

Set tsum = number of nodes reachable from u
Set counter = 0
For each vertex $t \neq v$ in G

Proof of **coNL** = **NL**

Claim

If $\mathit{CC} \in \mathbf{NL}$ *, then* $\mathit{UNREACHABILITY} \in \mathbf{NL}$

Set tsum = number of nodes reachable from u

Set counter = 0

For each vertex $t \neq v$ in G

- Guess if t is reachable from u

Proof of **coNL** = **NL**

Claim

If $\mathtt{CC} \in \mathbf{NL}$ *, then* $\mathtt{UNREACHABILITY} \in \mathbf{NL}$

Set tsum = number of nodes reachable from u

Set counter = 0

For each vertex $t \neq v$ in G

- Guess if t is reachable from u
- If the guess is that t *is* reachable

Proof of **coNL** = **NL**

Claim

If $\mathtt{CC} \in \mathbf{NL}$ *, then* $\mathtt{UNREACHABILITY} \in \mathbf{NL}$

Set tsum = number of nodes reachable from u

Set counter = 0

For each vertex $t \neq v$ in G

- Guess if t is reachable from u
- If the guess is that t *is* reachable
  - Guess a path from u to t

Proof of **coNL** = **NL**

Claim

If $\texttt{CC} \in$ ***NL*** *, then* $\texttt{UNREACHABILITY} \in$ ***NL***

Set tsum = number of nodes reachable from u

Set counter = 0

For each vertex $t \neq v$ in G

- Guess if t is reachable from u
- If the guess is that t *is* reachable
  - Guess a path from u to t
  - Verify that the above path is correct.

Proof of **coNL** = **NL**

Claim

If $\mathtt{CC} \in \mathbf{NL}$ *, then* $\mathtt{UNREACHABILITY} \in \mathbf{NL}$

Set tsum = number of nodes reachable from u

Set counter = 0

For each vertex $t \neq v$ in G

- Guess if t is reachable from u
- If the guess is that t *is* reachable
  - Guess a path from u to t
  - Verify that the above path is correct.
  - If verification fails at any stage, halt with 'reject'

Proof of **coNL** = **NL**

Claim

If $\mathtt{CC} \in \textbf{NL}$ *, then* $\mathtt{UNREACHABILITY} \in \textbf{NL}$

Set tsum = number of nodes reachable from u

Set counter = 0

For each vertex $t \neq v$ in G

- Guess if t is reachable from u
- If the guess is that t *is* reachable
  - Guess a path from u to t
  - Verify that the above path is correct.
  - If verification fails at any stage, halt with 'reject'
  - If verification succeeds, increment counter.

Proof of **coNL** = **NL**

Claim

If $\texttt{CC} \in \textbf{NL}$ *, then* $\texttt{UNREACHABILITY} \in \textbf{NL}$

Set tsum = number of nodes reachable from u

Set counter = 0

For each vertex $t \neq v$ in G

- Guess if t is reachable from u
- If the guess is that t *is* reachable
  - Guess a path from u to t
  - Verify that the above path is correct.
  - If verification fails at any stage, halt with 'reject'
  - If verification succeeds, increment counter.

If counter = tsum, 'accept'

Proof of **coNL** = **NL**

Claim

If $\mathit{CC} \in \mathbf{NL}$ *, then* $\mathit{UNREACHABILITY} \in \mathbf{NL}$

Set tsum = number of nodes reachable from u
Set counter = 0
For each vertex $t \neq v$ in G

- Guess if t is reachable from u
- If the guess is that t *is* reachable
  - Guess a path from u to t
  - Verify that the above path is correct.
  - If verification fails at any stage, halt with 'reject'
  - If verification succeeds, increment counter.

If counter = tsum, 'accept'
Otherwise, 'reject'

Proof of **coNL** = **NL**

Claim

*If $\mathtt{CC} \in$ **NL** , then $\mathtt{UNREACHABILITY} \in$ **NL***

Set tsum = number of nodes reachable from u

Set counter = 0

For each vertex $t \neq v$ in G

- Guess if t is reachable from u
- If the guess is that t *is* reachable
  - Guess a path from u to t
  - Verify that the above path is correct.
  - If verification fails at any stage, halt with 'reject'
  - If verification succeeds, increment counter.

If counter = tsum, 'accept'

Otherwise, 'reject'

Claim

The above algorithm uses $O(\log n)$ non-deterministic space

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an 'accept'
  - If v is reachable, *all* sequences should result in a reject

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an 'accept'
  - If v is reachable, *all* sequences should result in a reject
- Assume v is not reachable from u.

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an 'accept'
  - If v is reachable, *all* sequences should result in a reject
- Assume v is not reachable from u.
  - $\exists$ sequence of guesses that identify vertices connected to u

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an 'accept'
  - If v is reachable, *all* sequences should result in a reject
- Assume v is not reachable from u.
  - $\exists$ sequence of guesses that identify vertices connected to u
  - $\exists$ guesses which verify correctly the connecting path from u

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an 'accept'
  - If v is reachable, *all* sequences should result in a reject
- Assume v is not reachable from u.
  - $\exists$ sequence of guesses that identify vertices connected to u
  - $\exists$ guesses which verify correctly the connecting path from u
  - Therefore counter gets incremented exactly *tsum* times.

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an ‘accept’
  - If v is reachable, *all* sequences should result in a reject
- Assume v is not reachable from u.
  - $\exists$ sequence of guesses that identify vertices connected to u
  - $\exists$ guesses which verify correctly the connecting path from u
  - Therefore counter gets incremented exactly *tsum* times.
  - The **NTM** halts with an accept.

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an 'accept'
  - If v is reachable, *all* sequences should result in a reject
- Assume v is not reachable from u.
  - $\exists$ sequence of guesses that identify vertices connected to u
  - $\exists$ guesses which verify correctly the connecting path from u
  - Therefore counter gets incremented exactly *tsum* times.
  - The **NTM** halts with an accept.
  - Correctness: If, not counting v, there are already *tsum* connected vertices, this implies that v is not reachable from u.

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an 'accept'
  - If v is reachable, *all* sequences should result in a reject
- Assume v is not reachable from u.
  - $\exists$ sequence of guesses that identify vertices connected to u
  - $\exists$ guesses which verify correctly the connecting path from u
  - Therefore counter gets incremented exactly *tsum* times.
  - The **NTM** halts with an accept.
  - Correctness: If, not counting v, there are already *tsum* connected vertices, this implies that v is not reachable from u.
- Assume v is reachable from u.

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an 'accept'
  - If v is reachable, *all* sequences should result in a reject
- Assume v is not reachable from u.
  - $\exists$ sequence of guesses that identify vertices connected to u
  - $\exists$ guesses which verify correctly the connecting path from u
  - Therefore counter gets incremented exactly *tsum* times.
  - The **NTM** halts with an accept.
  - Correctness: If, not counting v, there are already *tsum* connected vertices, this implies that v is not reachable from u.
- Assume v is reachable from u.
  - No matter what we guess, we never incorrectly say a vertex t is reachable when it is not

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an 'accept'
  - If v is reachable, *all* sequences should result in a reject
- Assume v is not reachable from u.
  - $\exists$ sequence of guesses that identify vertices connected to u
  - $\exists$ guesses which verify correctly the connecting path from u
  - Therefore counter gets incremented exactly $tsum$ times.
  - The **NTM** halts with an accept.
  - Correctness: If, not counting v, there are already $tsum$ connected vertices, this implies that v is not reachable from u.
- Assume v is reachable from u.
  - No matter what we guess, we never incorrectly say a vertex t is reachable when it is not
  - Therefore, even if we guess the connected vertices correctly, counter will become at most $tsum - 1$.

Proof of **coNL** = **NL**

- Remember our objective:
  - If v is not reachable from u, at least one sequence should result in an 'accept'
  - If v is reachable, *all* sequences should result in a reject
- Assume v is not reachable from u.
  - $\exists$ sequence of guesses that identify vertices connected to u
  - $\exists$ guesses which verify correctly the connecting path from u
  - Therefore counter gets incremented exactly $tsum$ times.
  - The **NTM** halts with an accept.
  - Correctness: If, not counting v, there are already $tsum$ connected vertices, this implies that v is not reachable from u.
- Assume v is reachable from u.
  - No matter what we guess, we never incorrectly say a vertex t is reachable when it is not
  - Therefore, even if we guess the connected vertices correctly, counter will become at most $tsum - 1$.
  - Correctness: Therefore, we always reject.

$\mathtt{CC} \in \mathbf{NL}$

Claim

$$\mathit{CC} \in \textbf{\textit{NL}}$$

$\texttt{CC} \in \mathbf{NL}$

Claim

$$\texttt{CC} \in \mathbf{NL}$$

Proof.

We know that REACHABILITY is in **NL**

$\texttt{CC} \in$ **NL**

Claim

$$\texttt{CC} \in \textbf{NL}$$

Proof.

We know that $\texttt{REACHABILITY}$ is in **NL**
$\texttt{CC}$ is just summing up $\texttt{REACHABILITY}$ over all vertices:

$\mathtt{CC} \in \mathbf{NL}$

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

We know that $\mathtt{REACHABILITY}$ is in **NL**

$\mathtt{CC}$ is just summing up $\mathtt{REACHABILITY}$ over all vertices:

- Set counter $= 0$

$\texttt{CC} \in \mathbf{NL}$

Claim

$$\texttt{CC} \in \mathbf{NL}$$

Proof.

We know that $\texttt{REACHABILITY}$ is in **NL**

$\texttt{CC}$ is just summing up $\texttt{REACHABILITY}$ over all vertices:

- Set counter $= 0$
- For each vertex t in G

$\texttt{CC} \in \mathbf{NL}$

Claim

$$\texttt{CC} \in \mathbf{NL}$$

Proof.

We know that $\texttt{REACHABILITY}$ is in **NL**
$\texttt{CC}$ is just summing up $\texttt{REACHABILITY}$ over all vertices:

- Set counter $= 0$
- For each vertex t in G
  - Guess a path from u to t

$\texttt{CC} \in \mathbf{NL}$

Claim

$$\texttt{CC} \in \mathbf{NL}$$

Proof.

We know that $\texttt{REACHABILITY}$ is in **NL**
$\texttt{CC}$ is just summing up $\texttt{REACHABILITY}$ over all vertices:

- Set counter $= 0$
- For each vertex t in G
  - Guess a path from u to t
  - Verify that that path is correct.

$\texttt{CC} \in \mathbf{NL}$

Claim

$$\texttt{CC} \in \mathbf{NL}$$

Proof.

We know that $\texttt{REACHABILITY}$ is in **NL**
$\texttt{CC}$ is just summing up $\texttt{REACHABILITY}$ over all vertices:

- Set counter $= 0$
- For each vertex t in G
  - Guess a path from u to t
  - Verify that that path is correct.
  - If verification fails at any stage, halt with 'reject'

$\mathtt{CC} \in \mathbf{NL}$

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

We know that REACHABILITY is in **NL**

CC is just summing up REACHABILITY over all vertices:

- Set counter $= 0$
- For each vertex t in G
  - Guess a path from u to t
  - Verify that that path is correct.
  - If verification fails at any stage, halt with 'reject'
  - If verification succeeds, increment counter.

CC $\in$ **NL**

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

We know that REACHABILITY is in **NL**
CC is just summing up REACHABILITY over all vertices:

- Set counter $= 0$
- For each vertex t in G
  - Guess a path from u to t
  - Verify that that path is correct.
  - If verification fails at any stage, halt with 'reject'
  - If verification succeeds, increment counter.
- Output counter

$\mathtt{CC} \in$ **NL**

Claim

$$\mathtt{CC} \in \textbf{\textit{NL}}$$

Proof.

We know that $\mathtt{REACHABILITY}$ is in **NL**
$\mathtt{CC}$ is just summing up $\mathtt{REACHABILITY}$ over all vertices:

- Set counter $= 0$
- For each vertex t in G
  - Guess a path from u to t
  - Verify that that path is correct.
  - If verification fails at any stage, halt with 'reject'
  - If verification succeeds, increment counter.
- Output counter

Problem?

CC $\in$ NL

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

We know that REACHABILITY is in **NL**

CC is just summing up REACHABILITY over all vertices:

- Set counter $= 0$
- For each vertex t in G
  - Guess a path from u to t
  - Verify that that path is correct.
  - If verification fails at any stage, halt with 'reject'
  - If verification succeeds, increment counter.
- Output counter

Problem? All paths reject! Only works if every vertex is reachable. □

$\mathtt{CC} \in \mathbf{NL}$

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

CC $\in$ **NL**

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

CC is just summing up REACHABILITY over all vertices carefully:

CC $\in$ NL

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

CC is just summing up REACHABILITY over all vertices carefully:

- Set counter $= 0$
- For each vertex t in G

$\texttt{CC} \in \mathbf{NL}$

Claim

$$\texttt{CC} \in \mathbf{NL}$$

Proof.

$\texttt{CC}$ is just summing up $\texttt{REACHABILITY}$ over all vertices carefully:

- Set counter $= 0$
- For each vertex t in G
  - Guess if t is reachable from u

$\mathtt{CC} \in \mathbf{NL}$

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

$\mathtt{CC}$ is just summing up $\mathtt{REACHABILITY}$ over all vertices carefully:

- Set counter $= 0$
- For each vertex t in G
  - Guess if t is reachable from u
  - If the guess is that t *is* reachable

$\mathtt{CC} \in \mathbf{NL}$

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

$\mathtt{CC}$ is just summing up $\mathtt{REACHABILITY}$ over all vertices carefully:

- Set counter $= 0$
- For each vertex t in G
  - Guess if t is reachable from u
  - If the guess is that t *is* reachable
    - Guess a path from u to t

$\texttt{CC} \in \mathbf{NL}$

Claim

$$\texttt{CC} \in \mathbf{NL}$$

Proof.

$\texttt{CC}$ is just summing up $\texttt{REACHABILITY}$ over all vertices carefully:

- Set counter $= 0$
- For each vertex t in G
  - Guess if t is reachable from u
  - If the guess is that t *is* reachable
    - Guess a path from u to t
    - Verify that that path is correct.

$\texttt{CC} \in \mathbf{NL}$

Claim

$$\texttt{CC} \in \textbf{\textit{NL}}$$

Proof.

$\texttt{CC}$ is just summing up $\texttt{REACHABILITY}$ over all vertices carefully:

- Set counter $= 0$
- For each vertex t in G
  - Guess if t is reachable from u
  - If the guess is that t *is* reachable
    - Guess a path from u to t
    - Verify that that path is correct.
    - If verification fails at any stage, halt with 'reject'

CC $\in$ NL

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

CC is just summing up REACHABILITY over all vertices carefully:

- Set counter $= 0$
- For each vertex t in G
  - Guess if t is reachable from u
  - If the guess is that t *is* reachable
    - Guess a path from u to t
    - Verify that that path is correct.
    - If verification fails at any stage, halt with 'reject'
    - If verification succeeds, increment counter.

CC $\in$ NL

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

CC is just summing up REACHABILITY over all vertices carefully:

- Set counter $= 0$
- For each vertex t in G
  - Guess if t is reachable from u
  - If the guess is that t *is* reachable
    - Guess a path from u to t
    - Verify that that path is correct.
    - If verification fails at any stage, halt with 'reject'
    - If verification succeeds, increment counter.
- Output counter

$\mathtt{CC} \in \mathbf{NL}$

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

$\mathtt{CC}$ is just summing up $\mathtt{REACHABILITY}$ over all vertices carefully:

- Set counter $= 0$
- For each vertex t in G
  - Guess if t is reachable from u
  - If the guess is that t *is* reachable
    - Guess a path from u to t
    - Verify that that path is correct.
    - If verification fails at any stage, halt with 'reject'
    - If verification succeeds, increment counter.
- Output counter

Problem?

$\mathtt{CC} \in \mathbf{NL}$

Claim

$$\mathtt{CC} \in \mathbf{NL}$$

Proof.

$\mathtt{CC}$ is just summing up $\mathtt{REACHABILITY}$ over all vertices carefully:

- Set counter $= 0$
- For each vertex t in G
  - Guess if t is reachable from u
  - If the guess is that t *is* reachable
    - Guess a path from u to t
    - Verify that that path is correct.
    - If verification fails at any stage, halt with 'reject'
    - If verification succeeds, increment counter.
- Output counter

Problem? All paths with subsets of reachable vertices guessed correctly give different answers (including the correct one!) □

$\mathtt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

$\mathtt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

$\texttt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

$\texttt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Compute and check if $t \in T_{i+1}$

$\texttt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- For each vertex s in G
  - Compute and check if $s \in T_i$
  - If $s \in T_i$ and $(s, t) \in E$, increment T_{i+1}

$\texttt{CC} \in$ **NL**

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0$
- For each vertex s in G
  - Guess the exact subset V' of vertices in T_i
  - counter: Maintains size of V'

$\mathtt{CC} \in$ **NL**

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0$
- For each vertex s in G
  - Guess the exact subset V' of vertices in T_i
  - counter: Maintains size of V'
  - If counter $\neq T_i$, know we made a mistake

$\mathtt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0$
- For each vertex s in G
  - Guess the exact subset V' of vertices in T_i
  - counter: Maintains size of V'
  - If counter $\neq T_i$, know we made a mistake
  - Also verify that each vertex in V' is in T_i

$\mathtt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0$
- For each vertex s in G
  - Guess the exact subset V' of vertices in T_i
  - counter: Maintains size of V'
  - If counter $\neq T_i$, know we made a mistake
  - Also verify that each vertex in V' is in T_i
  - V' is correct. Simply check if an edge to t from it.

$\mathtt{CC} \in$ **NL**

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0, r = 0$
- For each vertex s in G
  - Guess if $s \in T_i$

$\mathtt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0, r = 0$
- For each vertex s in G
  - Guess if $s \in T_i$
  - If the guess is that $s \in T_i$
    - Guess a path of length i from u to s
    - Verify that the path is correct

$\mathtt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0, r = 0$
- For each vertex s in G
    - Guess if $s \in T_i$
    - If the guess is that $s \in T_i$
        - Guess a path of length i from u to s
        - Verify that the path is correct
        - If verification fails at any stage, halt with 'reject'

$\mathtt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0, r = 0$
- For each vertex s in G
  - Guess if $s \in T_i$
  - If the guess is that $s \in T_i$
    - Guess a path of length i from u to s
    - Verify that the path is correct
    - If verification fails at any stage, halt with 'reject'
    - If verification succeeds, counter++.

$\texttt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0, r = 0$
- For each vertex s in G
  - Guess if $s \in T_i$
  - If the guess is that $s \in T_i$
    - Guess a path of length i from u to s
    - Verify that the path is correct
    - If verification fails at any stage, halt with 'reject'
    - If verification succeeds, counter++. If $(s, t) \in E$, set $r = 1$

CC $\in$ **NL**

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0, r = 0$
- For each vertex s in G
  - Guess if $s \in T_i$
  - If the guess is that $s \in T_i$
    - Guess a path of length i from u to s
    - Verify that the path is correct
    - If verification fails at any stage, halt with 'reject'
    - If verification succeeds, counter++. If $(s, t) \in E$, set $r = 1$
- If counter $\neq T_i$, halt with 'reject'

$\texttt{CC} \in \mathbf{NL}$

Let T_i be the count of vertices reachable from u in at most i steps.

Claim

If we know T_i, we can compute T_{i+1}.

Set $T_{i+1} = 0$

For each vertex t in G

- Set $counter = 0, r = 0$
- For each vertex s in G
  - Guess if $s \in T_i$
  - If the guess is that $s \in T_i$
    - Guess a path of length i from u to s
    - Verify that the path is correct
    - If verification fails at any stage, halt with 'reject'
    - If verification succeeds, counter++. If $(s, t) \in E$, set $r = 1$
- If counter $\neq T_i$, halt with 'reject'
- If $r = 1$, increment T_{i+1}

$\mathtt{CC} \in \mathbf{NL}$

- Note that T_{n-1} is the answer to problem $\mathtt{CC}$.

$\texttt{CC} \in \mathbf{NL}$

- Note that T_{n-1} is the answer to problem $\texttt{CC}$.
  - Construct T_i inductively

$\texttt{CC} \in \mathbf{NL}$

- Note that T_{n-1} is the answer to problem $\texttt{CC}$.
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.

$\texttt{CC} \in$ NL

- Note that T_{n-1} is the answer to problem $\texttt{CC}$.
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i

$\texttt{CC} \in \mathbf{NL}$

- Note that T_{n-1} is the answer to problem $\texttt{CC}$.
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i
  - Can't afford to keep all T_j, only T_i to compute T_{i+1}

$\texttt{CC} \in \textbf{NL}$

- Note that T_{n-1} is the answer to problem $\texttt{CC}$.
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i
  - Can't afford to keep all T_j, only T_i to compute T_{i+1}
  - Reuse the space for computing each T_{i+1}

$\texttt{CC} \in$ **NL**

- Note that T_{n-1} is the answer to problem $\texttt{CC}$.
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i
  - Can't afford to keep all T_j, only T_i to compute T_{i+1}
  - Reuse the space for computing each T_{i+1}
- The algorithm described is correct.

CC $\in$ **NL**

- Note that T_{n-1} is the answer to problem CC .
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i
  - Can't afford to keep all T_j, only T_i to compute T_{i+1}
  - Reuse the space for computing each T_{i+1}
- The algorithm described is correct.
  - Each path that halts with 'accept' gives the same answer.

$\texttt{CC} \in \textbf{NL}$

- Note that T_{n-1} is the answer to problem $\texttt{CC}$.
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i
  - Can't afford to keep all T_j, only T_i to compute T_{i+1}
  - Reuse the space for computing each T_{i+1}
- The algorithm described is correct.
  - Each path that halts with 'accept' gives the same answer.
  - Pretty easy to see that we check for each type of failure.

$\texttt{CC} \in \mathbf{NL}$

- Note that T_{n-1} is the answer to problem $\texttt{CC}$.
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i
  - Can't afford to keep all T_j, only T_i to compute T_{i+1}
  - Reuse the space for computing each T_{i+1}
- The algorithm described is correct.
  - Each path that halts with 'accept' gives the same answer.
  - Pretty easy to see that we check for each type of failure.
  - Proof given nicely in Papadimitriou's book.

CC $\in$ NL

- Note that T_{n-1} is the answer to problem CC .
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i
  - Can't afford to keep all T_j, only T_i to compute T_{i+1}
  - Reuse the space for computing each T_{i+1}
- The algorithm described is correct.
  - Each path that halts with 'accept' gives the same answer.
  - Pretty easy to see that we check for each type of failure.
  - Proof given nicely in Papadimitriou's book.
- Space used is $O(\log n)$

$\mathtt{CC} \in \mathbf{NL}$

- Note that T_{n-1} is the answer to problem $\mathtt{CC}$.
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i
  - Can't afford to keep all T_j, only T_i to compute T_{i+1}
  - Reuse the space for computing each T_{i+1}
- The algorithm described is correct.
  - Each path that halts with 'accept' gives the same answer.
  - Pretty easy to see that we check for each type of failure.
  - Proof given nicely in Papadimitriou's book.
- Space used is $O(\log n)$
  - Re-use space over T_i, so consider space for each T_i

$\texttt{CC} \in \mathbf{NL}$

- Note that T_{n-1} is the answer to problem $\texttt{CC}$.
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i
  - Can't afford to keep all T_j, only T_i to compute T_{i+1}
  - Reuse the space for computing each T_{i+1}
- The algorithm described is correct.
  - Each path that halts with 'accept' gives the same answer.
  - Pretty easy to see that we check for each type of failure.
  - Proof given nicely in Papadimitriou's book.
- Space used is $O(\log n)$
  - Re-use space over T_i, so consider space for each T_i
  - Use constant number of vertex indices – $O(\log n)$

$\texttt{CC} \in$ **NL**

- Note that T_{n-1} is the answer to problem $\texttt{CC}$.
  - Construct T_i inductively
  - $T_0 = 1$ – just u itself.
  - To compute T_{i+1}, we only need T_i
  - Can't afford to keep all T_j, only T_i to compute T_{i+1}
  - Reuse the space for computing each T_{i+1}
- The algorithm described is correct.
  - Each path that halts with 'accept' gives the same answer.
  - Pretty easy to see that we check for each type of failure.
  - Proof given nicely in Papadimitriou's book.
- Space used is $O(\log n)$
  - Re-use space over T_i, so consider space for each T_i
  - Use constant number of vertex indices – $O(\log n)$
  - Use constant number of counters – $O(\log n)$

UNREACHABILITY $\in NL$

Nabil's Note: These are slides by Laiq. I have not checked them.

UNREACHABILITY

We now give the 'certificate'-based proof of the previous theorem.
We know that if UNREACHABILITY $\in NL$ then $NL = coNL$

UNREACHABILITY $\in NL$

Nabil's Note: These are slides by Laiq. I have not checked them.

UNREACHABILITY

We now give the 'certificate'-based proof of the previous theorem.
We know that if UNREACHABILITY $\in NL$ then $NL = coNL$

UNREACHABILITY

CLAIM: UNREACHABILITY $\in NL$
we only need to show $O(log - n)$ space Algorithm A such that

- $\exists$ polynomial sized certificate and
- $A(< G, s, t >, u) = 1$ *iff* t is not reachable from s in G.

UNREACHABILITY $\in NL$ - PROOF

- Let C_i be the set of vertices reachable from s in at most i steps

UNREACHABILITY $\in NL$ - PROOF

- Let C_i be the set of vertices reachable from s in at most i steps
- vertices are labeled from 1 to n so that each could be represented by log-n bits.

UNREACHABILITY $\in NL$ - PROOF

- Let C_i be the set of vertices reachable from s in at most i steps
- vertices are labeled from 1 to n so that each could be represented by log-n bits.
- The certificate only sequence of vertices from s to v (for $v \in C_i$)

UNREACHABILITY $\in NL$ - PROOF

- Let C_i be the set of vertices reachable from s in at most i steps
- vertices are labeled from 1 to n so that each could be represented by log-n bits.
- The certificate only sequence of vertices from s to v (for $v \in C_i$)
- Certificate could be checked using read-once Access

UNREACHABILITY $\in NL$ - PROOF

- Let C_i be the set of vertices reachable from s in at most i steps
- vertices are labeled from 1 to n so that each could be represented by log-n bits.
- The certificate only sequence of vertices from s to v (for $v \in C_i$)
- Certificate could be checked using read-once Access
  - first vertex of sequence should be s.

UNREACHABILITY $\in NL$ - PROOF

- Let C_i be the set of vertices reachable from s in at most i steps
- vertices are labeled from 1 to n so that each could be represented by log-n bits.
- The certificate only sequence of vertices from s to v (for $v \in C_i$)
- Certificate could be checked using read-once Access
  - first vertex of sequence should be s.
  - for $s > 0$, v_{j-1} and v_j should have an edge.

UNREACHABILITY $\in NL$ - PROOF

- Let C_i be the set of vertices reachable from s in at most i steps
- vertices are labeled from 1 to n so that each could be represented by log-n bits.
- The certificate only sequence of vertices from s to v (for $v \in C_i$)
- Certificate could be checked using read-once Access
  - first vertex of sequence should be s.
  - for $s > 0$, v_{j-1} and v_j should have an edge.
  - $v_k = v$ (last vertex should be v)

$\texttt{UNREACHABILITY} \in NL$ - PROOF

- Let C_i be the set of vertices reachable from s in at most i steps
- vertices are labeled from 1 to n so that each could be represented by log-n bits.
- The certificate only sequence of vertices from s to v (for $v \in C_i$)
- Certificate could be checked using read-once Access
  - first vertex of sequence should be s.
  - for $s > 0$, v_{j-1} and v_j should have an edge.
  - $v_k = v$ (last vertex should be v)
  - certificate is at most polynomial sized.

$\texttt{UNREACHABILITY} \in NL$ - PROOF

our Algorithm will use following 2 procedures

- Given $|C_i|$, certify if $v \notin C_i$ (1)

UNREACHABILITY $\in NL$ - PROOF

our Algorithm will use following 2 procedures

- Given $|C_i|$, certify if $v \notin C_i$ (1)
- Given $|C_i|$, certify if $|C_i| = c$ (2)

UNREACHABILITY $\in NL$ - PROOF

our Algorithm will use following 2 procedures

- Given $|C_i|$, certify if $v \notin C_i$ (1)
- Given $|C_i|$, certify if $|C_i| = c$ (2)

NOTE: (2) will be applied iteratively to find Max-Size of sets $C_1, C_2, C_3, ..., C_n$ and then will use (1) to certify that t $\notin C_n$

PROOF - UNREACHABILITY $\notin NL$

Procedure-1 Given $|C_i|$, certify if $v \notin C_i$

PROOF - UNREACHABILITY $\notin NL$

Procedure-1 Given $|C_i|$, certify if $v \notin C_i$

- Certificate here is the list of certificates for every $u \in C_i$ sorted in ascending order of Vertex labels

PROOF - UNREACHABILITY $\notin NL$

Procedure-1 Given $|C_i|$, certify if $v \notin C_i$

- Certificate here is the list of certificates for every $u \in C_i$ sorted in ascending order of Vertex labels
- The Algorithms checks
  - Each certificate is valid

PROOF - UNREACHABILITY $\notin NL$

Procedure-1 Given $|C_i|$, certify if $v \notin C_i$

- Certificate here is the list of certificates for every $u \in C_i$ sorted in ascending order of Vertex labels
- The Algorithms checks
  - Each certificate is valid
  - u's label (whose certificate is given) is larger than previous vertex's label
  - no certificate is provided for v

PROOF - UNREACHABILITY $\notin NL$

Procedure-1 Given $|C_i|$, certify if $v \notin C_i$

- Certificate here is the list of certificates for every $u \in C_i$ sorted in ascending order of Vertex labels
- The Algorithms checks
  - Each certificate is valid
  - u's label (whose certificate is given) is larger than previous vertex's label
  - no certificate is provided for v
  - total number of certificate $= |C_i|$

PROOF - $\texttt{UNREACHABILITY} \notin NL$

Procedure-1 Given $|C_i|$, certify if $v \notin C_i$

- Certificate here is the list of certificates for every $u \in C_i$ sorted in ascending order of Vertex labels
- The Algorithms checks
  - Each certificate is valid
  - u's label (whose certificate is given) is larger than previous vertex's label
  - no certificate is provided for v
  - total number of certificate $= |C_i|$
- if $v \notin C_i$ then certificate will be accepted

PROOF - UNREACHABILITY $\notin NL$

Procedure-1 Given $|C_i|$, certify if $v \notin C_i$

- Certificate here is the list of certificates for every $u \in C_i$ sorted in ascending order of Vertex labels
- The Algorithms checks
  - Each certificate is valid
  - u's label (whose certificate is given) is larger than previous vertex's label
  - no certificate is provided for v
  - total number of certificate $= |C_i|$
- if $v \notin C_i$ then certificate will be accepted
- if $v \in C_i$ then there won't be $|C_i|$ certificate for $|C_i|$ vertices other than v

PROOF - UNREACHABILITY $\notin$ *NL*

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $v \notin C_i$

- Similar to previous procedure

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $v \notin C_i$

- Similar to previous procedure
- Certificate here is the list of $|C_{i-1}|$ certificates certifying $u \in C_{i-1}$ for every $u \in C_{i-1}$

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $v \notin C_i$

- Similar to previous procedure
- Certificate here is the list of $|C_{i-1}|$ certificates certifying $u \in C_{i-1}$ for every $u \in C_{i-1}$
- The Algorithms checks

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $v \notin C_i$

- Similar to previous procedure
- Certificate here is the list of $|C_{i-1}|$ certificates certifying $u \in C_{i-1}$ for every $u \in C_{i-1}$
- The Algorithms checks
  - Each certificate is valid

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $v \notin C_i$

- Similar to previous procedure
- Certificate here is the list of $|C_{i-1}|$ certificates certifying $u \in C_{i-1}$ for every $u \in C_{i-1}$
- The Algorithms checks
  - Each certificate is valid
  - u's label (whose certificate is given) is larger than previous vertex's label

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $v \notin C_i$

- Similar to previous procedure
- Certificate here is the list of $|C_{i-1}|$ certificates certifying $u \in C_{i-1}$ for every $u \in C_{i-1}$
- The Algorithms checks
  - Each certificate is valid
  - u's label (whose certificate is given) is larger than previous vertex's label
  - verify no certificate is given for v or Neighbor of v

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $v \notin C_i$

- Similar to previous procedure
- Certificate here is the list of $|C_{i-1}|$ certificates certifying $u \in C_{i-1}$ for every $u \in C_{i-1}$
- The Algorithms checks
  - Each certificate is valid
  - u's label (whose certificate is given) is larger than previous vertex's label
  - verify no certificate is given for v or Neighbor of v
  - total number of certificates $= |C_i|$

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $v \notin C_i$

- Similar to previous procedure
- Certificate here is the list of $|C_{i-1}|$ certificates certifying $u \in C_{i-1}$ for every $u \in C_{i-1}$
- The Algorithms checks
  - Each certificate is valid
  - u's label (whose certificate is given) is larger than previous vertex's label
  - verify no certificate is given for v or Neighbor of v
  - total number of certificates $= |C_i|$
- since $v \in C_i$ *iff* $\exists\ u \in C_{i-1}$ such that $u = v$ or u is neighbor of v.

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $v \notin C_i$

- Similar to previous procedure
- Certificate here is the list of $|C_{i-1}|$ certificates certifying $u \in C_{i-1}$ for every $u \in C_{i-1}$
- The Algorithms checks
  - Each certificate is valid
  - u's label (whose certificate is given) is larger than previous vertex's label
  - verify no certificate is given for v or Neighbor of v
  - total number of certificates $= |C_i|$
- since $v \in C_i$ *iff* $\exists\ u \in C_{i-1}$ such that $u = v$ or u is neighbor of v.
- so the procedure provides correct results

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $v \notin C_i$

- Similar to previous procedure
- Certificate here is the list of $|C_{i-1}|$ certificates certifying $u \in C_{i-1}$ for every $u \in C_{i-1}$
- The Algorithms checks
  - Each certificate is valid
  - u's label (whose certificate is given) is larger than previous vertex's label
  - verify no certificate is given for v or Neighbor of v
  - total number of certificates $= |C_i|$
- since $v \in C_i$ *iff* $\exists\ u \in C_{i-1}$ such that $u = v$ or u is neighbor of v.
- so the procedure provides correct results

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $|C_i| = c$

- The certificate that $|C_i| = c$ will have n certificates

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $|C_i| = c$

- The certificate that $|C_i| = c$ will have n certificates
  - There is a certificate for every $u \in C_i$

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $|C_i| = c$

- The certificate that $|C_i| = c$ will have n certificates
  - There is a certificate for every $u \in C_i$
  - There is a certificate for every $u \notin C_i$

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $|C_i| = c$

- The certificate that $|C_i| = c$ will have n certificates
  - There is a certificate for every $u \in C_i$
  - There is a certificate for every $u \notin C_i$
- Algorithm will verify all certificates and count the certificates for $u \in C_i$

PROOF - UNREACHABILITY $\notin NL$

Given $|C_{i-1}|$, certify if $|C_i| = c$

- The certificate that $|C_i| = c$ will have n certificates
  - There is a certificate for every $u \in C_i$
  - There is a certificate for every $u \notin C_i$
- Algorithm will verify all certificates and count the certificates for $u \in C_i$
- Accepts if count = c