Lecture 32

NC and AC: Subclasses of $P_{/poly}$

Depth of a Circuit

Depth of a Circuit

Definition: The depth of a circuit is the length of the longest directed path from an input node

Depth of a Circuit

Definition: The depth of a circuit is the length of the longest directed path from an input node to the output node.

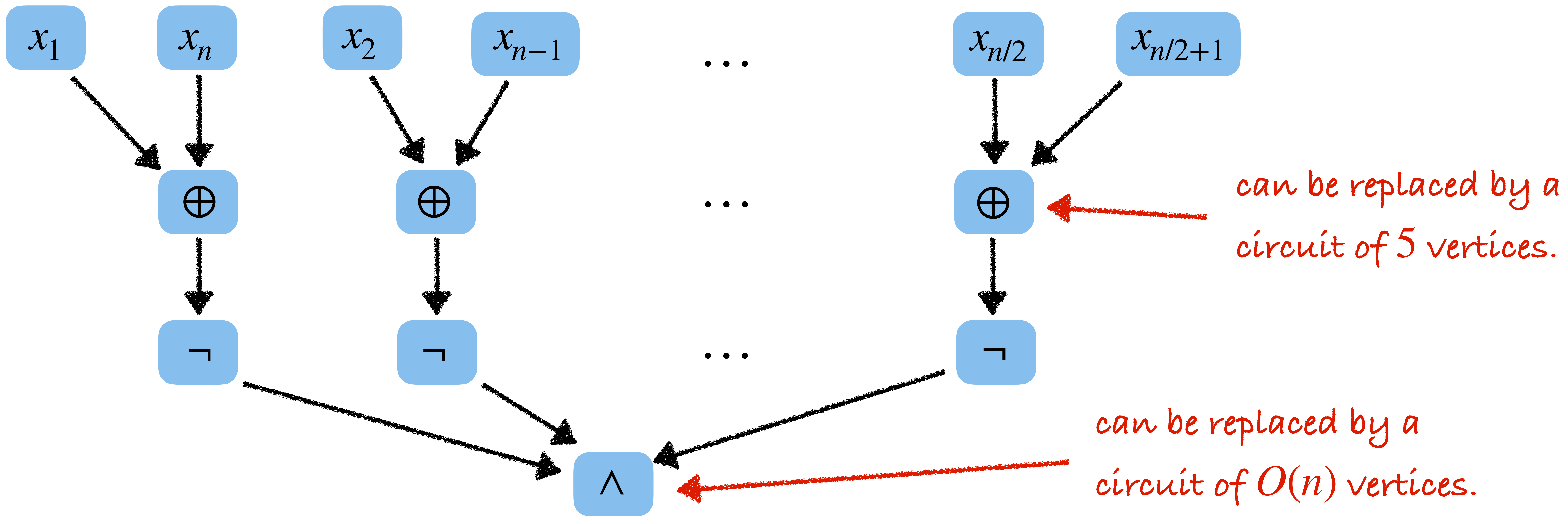
Depth of a Circuit

Definition: The depth of a circuit is the length of the longest directed path from an input node to the output node.

Example: Depth of *PALIN*'s circuit was

Depth of a Circuit

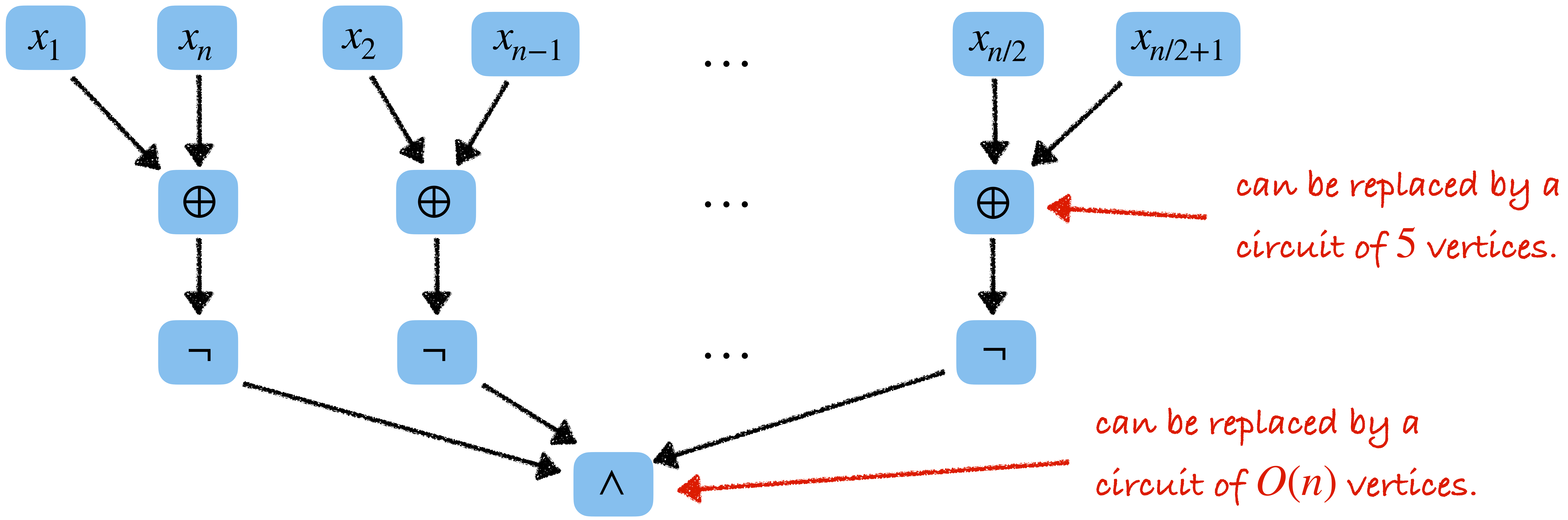
Definition: The depth of a circuit is the length of the longest directed path from an input node to the output node.



Example: Depth of *PALIN*'s circuit was

Depth of a Circuit

Definition: The depth of a circuit is the length of the longest directed path from an input node to the output node.



Example: Depth of *PALIN*'s circuit was $O(\log n)$.

NC **and** AC

NC and AC

Definition: For every $d \geq 0$, a language L is in $\mathbf{NC}^d$

NC **and** AC

Definition: For every $d \geq 0$, a language L is in **NC**^{d} if L can be decided by a family of circuits

NC and AC

Definition: For every $d \geq 0$, a language L is in **NC^d** if L can be decided by a family of circuits $\{C_n\}_{n \in \mathbb{N}}$, where C_n has $poly(n)$**-size** and $O(\log^d n)$ **depth**.

NC and AC

Definition: For every $d \geq 0$, a language L is in **NC**^{d} if L can be decided by a family of circuits $\{C_n\}_{n\in\mathbb{N}}$, where C_n has $poly(n)$**-size** and $O(\log^d n)$ **depth**.

Definition: **NC** $= \cup_{i\geq 0}$ **NC**^{i} (**NC** stands for Nick's Class)

NC and AC

Definition: For every $d \geq 0$, a language L is in **NC**^{d} if L can be decided by a family of circuits $\{C_n\}_{n \in \mathbb{N}}$, where C_n has $poly(n)$**-size** and $O(\log^d n)$ **depth**.

Definition: **NC** $= \cup_{i \geq 0}$ **NC**^{i} (**NC** stands for Nick's Class)

Definition: For every $d \geq 0$, a language L is in **AC**^{d}

NC and AC

Definition: For every $d \geq 0$, a language L is in NC^d if L can be decided by a family of circuits $\{C_n\}_{n\in\mathbb{N}}$, where C_n has $poly(n)$-**size** and $O(\log^d n)$ **depth**.

Definition: $\mathsf{NC} = \cup_{i\geq 0} \mathsf{NC}^i$ (**NC** stands for Nick's Class)

Definition: For every $d \geq 0$, a language L is in AC^d if L can be decided by a family of circuits

NC and AC

Definition: For every $d \geq 0$, a language L is in **NC**^{d} if L can be decided by a family of circuits $\{C_n\}_{n \in \mathbb{N}}$, where C_n has $poly(n)$-**size** and $O(\log^d n)$ **depth**.

Definition: **NC** $= \cup_{i \geq 0}$ **NC**^{i} (**NC** stands for Nick's Class)

Definition: For every $d \geq 0$, a language L is in **AC**^{d} if L can be decided by a family of circuits $\{C_n\}_{n \in \mathbb{N}}$, where C_n has unbounded fan-in, $poly(n)$-**size** and $O(\log^d n)$ **depth**.

NC and AC

Definition: For every $d \geq 0$, a language L is in **NC**^{d} if L can be decided by a family of circuits $\{C_n\}_{n\in\mathbb{N}}$, where C_n has *poly*(n)**-size** and $O(\log^d n)$ **depth**.

Definition: **NC** $= \cup_{i\geq 0}$ **NC**^{i} (**NC** stands for Nick's Class)

Definition: For every $d \geq 0$, a language L is in **AC**^{d} if L can be decided by a family of circuits $\{C_n\}_{n\in\mathbb{N}}$, where C_n has unbounded fan-in, *poly*(n)**-size** and $O(\log^d n)$ **depth**.

Definition: **AC** $= \cup_{i\geq 0}$ **AC**^{i} (**AC** stands for Alternating's Class)

NC and AC

Definition: For every $d \geq 0$, a language L is in **NC^d** if L can be decided by a family of circuits $\{C_n\}_{n\in\mathbb{N}}$, where C_n has $poly(n)$**-size** and $O(\log^d n)$ **depth**.

Definition: **NC** $= \cup_{i\geq 0}$ **NC^i** (**NC** stands for Nick's Class)

Definition: For every $d \geq 0$, a language L is in **AC^d** if L can be decided by a family of circuits $\{C_n\}_{n\in\mathbb{N}}$, where C_n has unbounded fan-in, $poly(n)$**-size** and $O(\log^d n)$ **depth**.

Definition: **AC** $= \cup_{i\geq 0}$ **AC^i** (**AC** stands for Alternating's Class)

Definition: **uniform-NC** and **uniform-AC** require circuits to be logspace uniform.

Motivation for NC and AC

Motivation for NC and AC

Why should we study **NC** and **AC**:

Motivation for NC and AC

Why should we study **NC** and **AC**:

- Showing $\mathbf{NP} \not\subseteq \mathbf{NC}$ might be easier than $\mathbf{NP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$ and may give insight to prove $\mathbf{NP} \not\subseteq \mathbf{P}_{/\mathbf{poly}}$.

Motivation for NC and AC

Why should we study **NC** and **AC**:

- Showing $\mathsf{NP} \nsubseteq \mathsf{NC}$ might be easier than $\mathsf{NP} \nsubseteq \mathsf{P}_{/\mathsf{poly}}$ and may give insight to prove $\mathsf{NP} \nsubseteq \mathsf{P}_{/\mathsf{poly}}$.
- **NC** and **AC** correspond to efficient parallel computation.

Observation on NC and AC

Observation on NC and AC

- Output of an NC^0 circuit depends on the constant number of input bits.

Observation on NC and AC

- Output of an NC^0 circuit depends on the constant number of input bits.

Example: $L = \{1^n \mid n \geq 1\}$ is not in NC^0.

Observation on NC and AC

- Output of an NC^0 circuit depends on the constant number of input bits.

Example: $L = \{1^n \mid n \geq 1\}$ is not in NC^0.

- $\mathsf{NC}^i \subseteq \mathsf{AC}^i \subseteq \mathsf{NC}^{i+1}$.

Observation on NC and AC

- Output of an $\mathbf{NC}^0$ circuit depends on the constant number of input bits.

Example: $L = \{1^n \mid n \geq 1\}$ is not in $\mathbf{NC}^0$.

- $\mathsf{NC}^i \subseteq \mathsf{AC}^i \subseteq \mathsf{NC}^{i+1}$.

trivial

Observation on NC and AC

- Output of an NC^0 circuit depends on the constant number of input bits.

Example: $L = \{1^n \mid n \geq 1\}$ is not in NC^0.

- $\mathsf{NC}^i \subseteq \mathsf{AC}^i \subseteq \mathsf{NC}^{i+1}$.

trivial

a vertex with unbounded fan-in can be replaced by an $O(\log n)$ depth circuit.

Observation on NC **and** AC

- Output of an $\mathbf{NC}^0$ circuit depends on the constant number of input bits.

Example: $L = \{1^n \mid n \geq 1\}$ is not in $\mathbf{NC}^0$.

- $\mathbf{NC}^i \subseteq \mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$. Thus, $\mathbf{NC} = \mathbf{AC}$.

trivial

a vertex with unbounded fan-in can be replaced by an $O(\log n)$ depth circuit.

Observation on NC and AC

- Output of an $\mathbf{NC}^0$ circuit depends on the constant number of input bits.

Example: $L = \{1^n \mid n \geq 1\}$ is not in $\mathbf{NC}^0$.

- $\mathbf{NC}^i \subseteq \mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$. Thus, $\mathbf{NC} = \mathbf{AC}$.

trivial

a vertex with unbounded fan-in can be replaced by an $O(\log n)$ depth circuit.

- $\mathbf{NC}^0 \subset \mathbf{AC}^0 \subset \mathbf{NC}^1$.

Observation on NC and AC

- Output of an NC^0 circuit depends on the constant number of input bits.

Example: $L = \{1^n \mid n \geq 1\}$ is not in NC^0.

- $\mathsf{NC}^i \subseteq \mathsf{AC}^i \subseteq \mathsf{NC}^{i+1}$. Thus, $\mathsf{NC} = \mathsf{AC}$.

trivial

a vertex with unbounded fan-in can be replaced by an $O(\log n)$ depth circuit.

- $\mathsf{NC}^0 \subset \mathsf{AC}^0 \subset \mathsf{NC}^1$.

L of above example

Observation on NC and AC

- Output of an $\mathbf{NC}^0$ circuit depends on the constant number of input bits.

Example: $L = \{1^n \mid n \geq 1\}$ is not in $\mathbf{NC}^0$.

- $\mathbf{NC}^i \subseteq \mathbf{AC}^i \subseteq \mathbf{NC}^{i+1}$. Thus, $\mathbf{NC} = \mathbf{AC}$.

trivial

a vertex with unbounded fan-in can be replaced by an $O(\log n)$ depth circuit.

- $\mathbf{NC}^0 \subset \mathbf{AC}^0 \subset \mathbf{NC}^1$.

L of above example

PARITY is in $\mathbf{NC}^1$ but not in $\mathbf{AC}^0$.

uniform-NC^1 **vs** L

uniform-NC^1 **vs** L

Theorem: uniform-$NC^1 \subseteq$ L.

uniform-NC^1 **vs** L

Theorem: uniform-$NC^1 \subseteq L$.

Proof:

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in uniform-NC^1.

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in uniform-NC^1.

We will construct a logspace algorithm A for L.

uniform-NC^1 vs L

Theorem: uniform-$\mathsf{NC}^1 \subseteq \mathsf{L}$.

Proof: Let L be in uniform-NC^1.

We will construct a logspace algorithm A for L.

$A(x)$:

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

uniform-NC^1 vs L

Theorem: uniform-NC^1 $\subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$B(V, x)$:

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$B(V, x)$:

if V is an input vertex

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

uniform-NC^1 vs L

Theorem: uniform-$\mathsf{NC}^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

$B(V, x)$:

if V is an input vertex

return appropriate x_i

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

$B(V, x)$:

if V is an input vertex

return appropriate x_i

let V_1, $V_2 =$ vertices with an edge to V

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

$B(V, x)$:

if V is an input vertex

return appropriate x_i

let $V_1, V_2 =$ vertices with an edge to V

$op =$ boolean operator of V

uniform-NC^1 vs L

Theorem: uniform-NC^1 $\subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

$B(V, x)$:

if V is an input vertex

return appropriate x_i

let V_1, $V_2 =$ vertices with an edge to V

$op =$ boolean operator of V

return $B(V_1, x)\ op\ B(V_2, x)$

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

$B(V, x)$:

if V is an input vertex

return appropriate x_i

let V_1, $V_2 =$ vertices with an edge to V

$op =$ boolean operator of V

return $B(V_1, x)\ op\ B(V_2, x)$

Space Complexity of B:

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

$B(V, x)$:

if V is an input vertex

return appropriate x_i

let V_1, $V_2 =$ vertices with an edge to V

$op =$ boolean operator of V

return $B(V_1, x)\ op\ B(V_2, x)$

Space Complexity of B: Total $O(\log n)$ recursion levels and at every level it is storing

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

$B(V, x)$:

if V is an input vertex

return appropriate x_i

let V_1, $V_2 =$ vertices with an edge to V

$op =$ boolean operator of V

return $B(V_1, x)\ op\ B(V_2, x)$

Space Complexity of B: Total $O(\log n)$ recursion levels and at every level it is storing constant information.

uniform-NC^1 vs L

Theorem: uniform-$NC^1 \subseteq$ L.

Proof: Let L be in **uniform-NC^1**.

We will construct a logspace algorithm A for L.

$A(x)$:

let $C =$ circuit for $|x|$ length input.

$V =$ output vertex of C

output $B(V, x)$

$B(V, x)$:

if V is an input vertex

return appropriate x_i

let V_1, $V_2 =$ vertices with an edge to V

$op =$ boolean operator of V

return $B(V_1, x)$ op $B(V_2, x)$

Space Complexity of B: Total $O(\log n)$ recursion levels and at every level it is storing constant information.

■

uniform-AC^1 **vs** NL

uniform-AC^1 **vs** NL

Theorem: NL $\subseteq$ uniform-AC^1.

uniform-AC^1 **vs** NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof:

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

uniform-AC^1 vs NL

Theorem: **NL** $\subseteq$ **uniform-AC^1**.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$A_l[i, j] = 1$ iff $\exists$ a path of length at most 2^l from i to j in $G_{M,x}$.

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$A_l[i, j] = 1$ iff $\exists$ a path of length at most 2^l from i to j in $G_{M,x}$.

How can we compute A_{l+1} from A_l?

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$A_l[i, j] = 1$ iff $\exists$ a path of length at most 2^l from i to j in $G_{M,x}$.

How can we compute A_{l+1} from A_l?

$A_{l+1}[i, j] \ =$

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$A_l[i, j] = 1$ iff $\exists$ a path of length at most 2^l from i to j in $G_{M,x}$.

How can we compute A_{l+1} from A_l?

$$A_{l+1}[i, j] = \vee_{1 \le k \le N} (A_l[i, k] \wedge A_l[k, j])$$

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$$A_l[i,j] = 1 \text{ iff } \exists \text{ a path of length at most } 2^l \text{ from } i \text{ to } j \text{ in } G_{M,x}.$$

How can we compute A_{l+1} from A_l?

$$A_{l+1}[i,j] \; = \; \vee_{1 \le k \le N} \left(A_l[i,k] \wedge A_l[k,j]\right)$$

Building the circuit for L:

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$$A_l[i, j] = 1 \text{ iff } \exists \text{ a path of length at most } 2^l \text{ from } i \text{ to } j \text{ in } G_{M,x}.$$

How can we compute A_{l+1} from A_l?

$$A_{l+1}[i, j] \;=\; \vee_{1 \le k \le N} (A_l[i, k] \wedge A_l[k, j])$$

Building the circuit for L:

- Have one layer for every $l \in [0, \log N]$.

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$$A_l[i, j] = 1 \text{ iff } \exists \text{ a path of length at most } 2^l \text{ from } i \text{ to } j \text{ in } G_{M,x}.$$

How can we compute A_{l+1} from A_l?

$$A_{l+1}[i, j] \ = \vee_{1 \le k \le N} (A_l[i, k] \wedge A_l[k, j])$$

Building the circuit for L:

- Have one layer for every $l \in [0, \log N]$.
- In the lth layer, keep one vertex for every (i, j)th entry of A_l.

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$$A_l[i, j] = 1 \text{ iff } \exists \text{ a path of length at most } 2^l \text{ from } i \text{ to } j \text{ in } G_{M,x}.$$

How can we compute A_{l+1} from A_l?

$$A_{l+1}[i, j] \; = \vee_{1 \le k \le N} (A_l[i, k] \wedge A_l[k, j])$$

Building the circuit for L:

- Have one layer for every $l \in [0, \log N]$.
- In the lth layer, keep one vertex for every (i, j)th entry of A_l.
- Arrange wires/edges from lth layer to $(l + 1)$th layer according to this formula.

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$$A_l[i,j] = 1 \text{ iff } \exists \text{ a path of length at most } 2^l \text{ from } i \text{ to } j \text{ in } G_{M,x}.$$

How can we compute A_{l+1} from A_l?

$$A_{l+1}[i,j] = \vee_{1 \leq k \leq N} (A_l[i,k] \wedge A_l[k,j])$$

Building the circuit for L:

- Have one layer for every $l \in [0, \log N]$.
- In the lth layer, keep one vertex for every (i,j)th entry of A_l.
- Arrange wires/edges from lth layer to $(l+1)$th layer according to this formula.

uniform-AC^1 vs NL

Theorem: **NL** $\subseteq$ **uniform-AC^1**.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$A_l[i,j] = 1$ iff $\exists$ a path of length at most 2^l from i to j in $G_{M,x}$.

How can we compute A_{l+1} from A_l?

$$A_{l+1}[i,j] = \vee_{1 \le k \le N} (A_l[i,k] \wedge A_l[k,j])$$

edges for 0th layer require some work

Building the circuit for L:

- Have one layer for every $l \in [0, \log N]$.
- In the lth layer, keep one vertex for every (i,j)th entry of A_l.
- Arrange wires/edges from lth layer to $(l+1)$th layer according to this formula.

uniform-AC^1 vs NL

Theorem: **NL** $\subseteq$ **uniform-AC^1**.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$$A_l[i,j] = 1 \text{ iff } \exists \text{ a path of length at most } 2^l \text{ from } i \text{ to } j \text{ in } G_{M,x}.$$

How can we compute A_{l+1} from A_l?

$$A_{l+1}[i,j] \;=\; \vee_{1 \le k \le N} \left(A_l[i,k] \wedge A_l[k,j]\right)$$

edges for 0th layer require some work

Building the circuit for L:

- Have one layer for every $l \in [0, \log N]$.
- In the lth layer, keep one vertex for every (i,j)th entry of A_l.
- Arrange wires/edges from lth layer to $(l+1)$th layer according to this formula.
- Keep the (start config, acc. config.) vertex in the last layer as output vertex.

uniform-AC^1 vs NL

Theorem: NL $\subseteq$ uniform-AC^1.

Proof: Let L be in **NL** and M be a logspace NTM that decides it.

Let A_l denote the $N \times N$ matrix, where $N =$ # of configurations of M on an input x, s.t.

$$A_l[i, j] = 1 \text{ iff } \exists \text{ a path of length at most } 2^l \text{ from } i \text{ to } j \text{ in } G_{M,x}.$$

How can we compute A_{l+1} from A_l?

$$A_{l+1}[i, j] \; = \vee_{1 \le k \le N} (A_l[i, k] \wedge A_l[k, j])$$

edges for 0th layer require some work

Building the circuit for L:

- Have one layer for every $l \in [0, \log N]$.
- In the lth layer, keep one vertex for every (i, j)th entry of A_l.
- Arrange wires/edges from lth layer to $(l + 1)$th layer according to this formula.
- Keep the (start config, acc. config.) vertex in the last layer as output vertex. ■