Computational Complexity

Lecture 9
Alternation
(Continued)
ATM

∃

∀

2
ATM

Alternating Turing Machine
ATM

- Alternating Turing Machine

- At each step, execution can fork into two (like NTM or co-NTM)
ATM

- Alternating Turing Machine
- At each step, execution can fork into two (like NTM or co-NTM)
- Two kinds of configurations: $\exists$ and $\forall$
ATM

- Alternating Turing Machine

- At each step, execution can fork into two (like NTM or co-NTM)

- Two kinds of configurations: \( \exists \) and \( \forall \)

- A \( \exists \) configuration is accepting if either child is accepting
Alternating Turing Machine

At each step, execution can fork into two (like NTM or co-NTM)

Two kinds of configurations: $\exists$ and $\forall$

$\exists$ configuration is accepting if either child is accepting

$\forall$ configuration is accepting only if both children are accepting
ATM

- Alternating Turing Machine
  - At each step, execution can fork into two (like NTM or co-NTM)
  - Two kinds of configurations: $\exists$ and $\forall$
    - A $\exists$ configuration is accepting if either child is accepting
    - A $\forall$ configuration is accepting only if both children are accepting
  - ATM accepts if start config accepts according to this rule
ATIME, ASPACE
ATIME, ASPACE

\[ \text{ATIME}(T) \subseteq \text{DSPACE}(T^2) \]
ATIME, ASPACE

\( \text{ATIME}(T) \subseteq \text{DSPACE}(T^2) \)

\( \text{AP} = \text{PSPACE} \)
ATIME, ASPACE

- $\text{ATIME}(T) \subseteq \text{DSPACE}(T^2)$
- $\text{AP} = \text{PSPACE}$
- $\text{ASPACE}(S) = \text{DTIME}(2^{O(S)})$
ATIME, ASPACE

\[ \text{ATIME}(T) \subseteq \text{DSPACE}(T^2) \]

\[ \text{AP} = \text{PSPACE} \]

\[ \text{ASPACE}(S) = \text{DTIME}(2^{O(S)}) \]

\[ \text{AL} = \text{P and APSPACE} = \text{EXP} \]
\text{DTIME}(2^{O(S)}) \subseteq \text{ASPACE}(S)
\[ \text{DTIME}(2^{O(S)}) \subseteq \text{ASPACE}(S) \]

To decide, is configuration after \( t \) steps accepting
\[ \text{DTIME}(2^{O(S)}) \subseteq \text{ASPACE}(S) \]

- To decide, is configuration after \( t \) steps accepting

- \( C(i,j,x) : \) if after \( i \) steps, \( j^{\text{th}} \) cell of config is \( x \)
DTIME($2^{O(S)}$) $\subseteq$ ASPACE(S)

To decide, is configuration after $t$ steps accepting

\[ C(i,j,x) : \text{if after } i \text{ steps, } j^{th} \text{ cell of config is } x \]

\[ C(i,j,x) : \exists a, b, c \text{ s.t. } x = F(a, b, c) \text{ and } C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) \]
DTIME($2^{O(S)}$) ⊆ ASPACE(S)

To decide, is configuration after $t$ steps accepting

C(i,j,x) : if after $i$ steps, $j^{th}$ cell of config is $x$

C(i,j,x): $\exists a, b, c$ st $x = F(a, b, c)$ and $C(i-1,j-1,a)$, $C(i-1,j,b)$, $C(i-1,j+1,c)$

Base case: $C(0,j,x)$ easy to check from input
DTIME($2^{O(S)}$) ⊆ ASPACE(S)

- To decide, is configuration after $t$ steps accepting
  - $C(i,j,x)$: if after $i$ steps, $j^{th}$ cell of config is $x$
  - $C(i,j,x)$: $\exists a,b,c \ s.t \ x=F(a,b,c)$ and $C(i-1,j-1,a)$, $C(i-1,j,b)$, $C(i-1,j+1,c)$

  - Base case: $C(0,j,x)$ easy to check from input

  - Naive recursion: Extra $O(S)$ space to store $i,j$ at each level for $2^{O(S)}$ levels!
ATM for TM simulation
ATM for TM simulation

ATM to check if $C(i,j,x)$
ATM for TM simulation

- ATM to check if $C(i, j, x)$

- $C(i, j, x)$: $\exists a, b, c$ st $x=F(a, b, c)$ and $C(i-1, j-1, a)$, $C(i-1, j, b)$, $C(i-1, j+1, c)$
ATM for TM simulation

- ATM to check if \( C(i,j,x) \)

- \( C(i,j,x): \exists a,b,c \text{ st } x=F(a,b,c) \text{ and } C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) \)

- Tail-recursion in parallel forks
ATM for TM simulation

- ATM to check if $C(i,j,x)$

  - $C(i,j,x)$: $\exists a,b,c \text{ st } x=F(a,b,c)$ and $C(i-1,j-1,a)$, $C(i-1,j,b)$, $C(i-1,j+1,c)$

- Tail-recursion in parallel forks

  - Check $x=F(a,b,c)$; then enter universal state, and non-deterministically choose one of the three conditions to check
ATM for TM simulation

- ATM to check if \( C(i,j,x) \)

- \( C(i,j,x) \): \( \exists a,b,c \) st \( x=F(a,b,c) \) and \( C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c) \)

- Tail-recursion in parallel forks
  
  - Check \( x=F(a,b,c) \); then enter universal state, and non-deterministically choose one of the three conditions to check
  
  - Overwrite \( C(i,j,x) \) with \( C(i-1,...) \) and reuse space
ATM for TM simulation

- ATM to check if $C(i,j,x)$

  $C(i,j,x): \exists a,b,c \text{ s.t. } x = F(a,b,c) \text{ and } C(i-1,j-1,a), C(i-1,j,b), C(i-1,j+1,c)$

- Tail-recursion in parallel forks

  Check $x = F(a,b,c)$; then enter universal state, and non-deterministically choose one of the three conditions to check

  Overwrite $C(i,j,x)$ with $C(i-1,...)$ and reuse space

  Stay within the same $O(S)$ space at each level!
ATM for TM simulation

- ATM to check if $C(i,j,x)$

- $C(i,j,x)$: $\exists a,b,c$ st $x = F(a,b,c)$ and $C(i-1,j-1,a)$, $C(i-1,j,b)$, $C(i-1,j+1,c)$

- Tail-recursion in parallel forks
  - Check $x = F(a,b,c)$; then enter universal state, and non-deterministically choose one of the three conditions to check
  - Overwrite $C(i,j,x)$ with $C(i-1,\ldots)$ and reuse space
  - Stay within the same $O(S)$ space at each level!
Zoo
Zoo
Non-Uniform Computation

Lecture 10
Non-Uniform Computational Models: Circuits
Non-Uniform Computation
Non-Uniform Computation

Uniform: Same program for all (the infinitely many) inputs
Non-Uniform Computation

- Uniform: Same program for all (the infinitely many) inputs
- Non-uniform: A different “program” for each input size
Non-Uniform Computation

- **Uniform:** Same program for all (the infinitely many) inputs
- **Non-uniform:** A different “program” for each input size
  - Then complexity of building the program and executing the program
Non-Uniform Computation

Uniform: Same program for all (the infinitely many) inputs

Non-uniform: A different “program” for each input size

Then complexity of building the program and executing the program

Sometimes will focus on the latter alone
Non-Uniform Computation

- Uniform: Same program for all (the infinitely many) inputs
- Non-uniform: A different “program” for each input size
  - Then complexity of building the program and executing the program
  - Sometimes will focus on the latter alone
  - Not entirely realistic if the program family is uncomputable or very complex to compute
Non-uniform advice
Non-uniform advice

Program: TM M and advice strings \{A_n\}
Non-uniform advice

- Program: TM M and advice strings \( \{A_n\} \)
- M given \( A_{|x|} \) along with x
Non-uniform advice

Program: TM $M$ and advice strings $\{A_n\}$

- $M$ given $A_{|x|}$ along with $x$
- $A_n$ can be the program for inputs of size $n$
Non-uniform advice

- Program: TM M and advice strings \{A_n\}
  - M given \(A_{|x|}\) along with \(x\)
  - \(A_n\) can be the program for inputs of size \(n\)
  - \(|A_n| = 2^n\) is sufficient
Non-uniform advice

- Program: TM $M$ and advice strings $\{A_n\}$
  - $M$ given $A_{|x|}$ along with $x$
  - $A_n$ can be the program for inputs of size $n$
  - $|A_n| = 2^n$ is sufficient
  - But $\{A_n\}$ can be uncomputable (even if just one bit long)
Non-uniform advice

- Program: TM $M$ and advice strings $\{A_n\}$
  - $M$ given $A_{|x|}$ along with $x$
  - $A_n$ can be the program for inputs of size $n$
  - $|A_n| = 2^n$ is sufficient
- But $\{A_n\}$ can be uncomputable (even if just one bit long)
  - e.g. advice to decide undecidable unary languages
P/poly and P/log
P/poly and P/log

\[ \text{DTIME}(T)/a \]
P/poly and P/log

\[ \text{DTIME}(T)/a \]

Languages decided by a TM in time \( T(n) \) using non-uniform advice of length \( a(n) \)
P/poly and \( P/\log \)

\[ \text{DTIME}(T)/a \]

Languages decided by a TM in time \( T(n) \) using non-uniform advice of length \( a(n) \)

\[ P/poly = \bigcup_{c,d,k>0} \text{DTIME}(kn^c)/kn^d \]
P/poly and P/log

- $\text{DTIME}(T)/a$
  - Languages decided by a TM in time $T(n)$ using non-uniform advice of length $a(n)$

- $P/poly = \bigcup_{c,d,k>0} \text{DTIME}(kn^c)/kn^d$

- $P/log = \bigcup_{c,k>0} \text{DTIME}(kn^c)/k \log n$
NP vs. P/log, P/poly
NP vs. \( \text{P/log, P/poly} \)

\( \text{P/log (or even DTIME(1)/1) has undecidable languages} \)
NP vs. P/log, P/poly

P/log (or even DTIME(1)/1) has undecidable languages

- e.g. unary undecidable languages
NP vs. $P/\log$, $P/poly$

- $P/\log$ (or even $\text{DTIME}(1)/1$) has undecidable languages
  - e.g. unary undecidable languages
- So $P/\log$ cannot be contained in any of the uniform complexity classes
NP vs. P/log, P/poly

- P/log (or even DTIME(1)/1) has undecidable languages
  - e.g. unary undecidable languages
  - So P/log cannot be contained in any of the uniform complexity classes
- P/log contains P
NP vs. P/log, P/poly

- P/log (or even DTIME(1)/1) has undecidable languages
  - e.g. unary undecidable languages
  - So P/log cannot be contained in any of the uniform complexity classes
- P/log contains P
- Does P/log or P/poly contain NP?
NP ⊆ P/log ⇒ NP=P
\[ \text{NP} \subseteq \text{P/log} \Rightarrow \text{NP} = \text{P} \]

Recall finding witness for an NP language is Turing reducible to deciding the language
Suppose given “oracles” for deciding all NP languages, can we easily find certificates?

Yes! So, if decision easy (decision-oracles realizable), then search is easy too!

Say, given $x$, need to find $w$ s.t. $(x,w) \in L'$ (if such $w$ exists)

Consider $L_1$ in NP: $(x,y) \in L_1$ iff $\exists z$ s.t. $(x,yz) \in L'$. (i.e., can $y$ be a prefix of a certificate for $x$).

Query $L_1$-oracle with $(x,0)$ and $(x,1)$. If $\exists w$, one of the two must be positive: say $(x,0) \in L_1$; then first bit of $w$ be 0.

For next bit query $L_1$-oracle with $(x,00)$ and $(x,01)$.
Suppose given “oracles” for deciding all NP languages, can we easily find certificates?

Yes! So, if decision easy (decision-oracles realizable), then search is easy too!

Say, given x, need to find w s.t. (x,w) ∈ L’ (if such w exists)

consider L_1 in NP: (x,y) ∈ L_1 iff ∃ z s.t. (x,yz) ∈ L’. (i.e., can y be a prefix of a certificate for x).

Query L_1-oracle with (x,0) and (x,1). If ∃ w, one of the two must be positive: say (x,0) ∈ L_1; then first bit of w be 0.

For next bit query L_1-oracle with (x,00) and (x,01)
NP \subseteq P/log \Rightarrow \text{NP=P}

Recall finding witness for an NP language is Turing reducible to deciding the language
NP \subseteq P/\log \implies NP=P

- Recall finding witness for an NP language is Turing reducible to deciding the language.

- If NP \subseteq P/\log, then for each L in NP, there is a poly-time TM with log advice which can find witness (via self-reduction).
NP ⊆ P/log ⇒ NP=P

- Recall finding witness for an NP language is Turing reducible to deciding the language.

- If NP ⊆ P/log, then for each L in NP, there is a poly-time TM with log advice which can find witness (via self-reduction).

- Guess advice (poly many), and for each guessed advice, run the TM and see if it finds witness.
If $NP \subseteq P/\log$, then for each $L$ in $NP$, there is a poly-time TM with log advice which can find witness (via self-reduction).

Recall finding witness for an NP language is Turing reducible to deciding the language.

Guess advice (poly many), and for each guessed advice, run the TM and see if it finds witness.

If no advice worked (one of them was correct), then input not in language.
$\mathsf{NP} \subseteq \mathsf{P/poly} \implies \mathsf{PH} = \Sigma_2^p$
NP \subseteq P/poly \Rightarrow \mathsf{PH}=\Sigma_2^P

\(\because\) Will show \(\Pi_2^P = \Sigma_2^P\)
\[ \text{NP} \subseteq \text{P/poly} \Rightarrow \text{PH} = \Sigma_2^P \]

- Will show \( \Pi_2^P = \Sigma_2^P \)
- Consider \( L = \{ x \mid \forall w_1 (x, w_1) \in L' \} \in \Pi_2^P \) where
  \( L' = \{(x,w_1) \mid \exists w_2 \ F(x,w_1,w_2)\} \in \text{NP} \)
\[
\text{NP} \subseteq \text{P/poly} \implies \text{PH} = \Sigma_{2}^{P}
\]

- Will show \( \Pi_{2}^{P} = \Sigma_{2}^{P} \)
- Consider \( L = \{ x | \forall w_{1} (x,w_{1}) \in L' \} \in \Pi_{2}^{P} \) where
  \( L' = \{ (x,w_{1}) | \exists w_{2} F(x,w_{1},w_{2}) \} \in \text{NP} \)

- If \( \text{NP} \subseteq \text{P/poly} \) then consider \( M \) with advice \( \{ A_{n} \} \) which finds witness for \( L' \): i.e. if \( (x,w_{1}) \in L' \), then \( M(x,w_{1}; A_{n}) \) outputs a witness \( w_{2} \) s.t. \( F(x,w_{1},w_{2}) \)
\[ \text{NP} \subseteq \text{P/poly} \Rightarrow \text{PH} = \Sigma_2^P \]

Will show $\Pi_2^P = \Sigma_2^P$

Consider $L = \{x | \forall w_1 \ (x, w_1) \in L' \} \in \Pi_2^P$ where
$L' = \{(x, w_1) | \exists w_2 \ F(x, w_1, w_2)\} \in \text{NP}$

If NP $\subseteq$ P/poly then consider M with advice \{A_n\} which finds witness for L': i.e. if $(x, w_1) \in L'$, then $M(x, w_1; A_n)$ outputs a witness $w_2$ s.t. $F(x, w_1, w_2)$

$L = \{x | \exists z \forall w_1 \ F(x, w_1, M(x, w_1; z)) \}$
Boolean Circuits
Boolean Circuits

Non-uniformity: circuit family $\{C_n\}$
Boolean Circuits

Non-uniformity: circuit family \( \{C_n\} \)

Given non-uniform computation \((M,\{A_n\})\), can define equivalent \( \{C_n\} \)
Boolean Circuits

- Non-uniformity: circuit family \( \{C_n\} \)

- Given non-uniform computation \((M,\{A_n\})\), can define equivalent \( \{C_n\} \)
Boolean Circuits

- Non-uniformity: circuit family \{C_n\}

- Given non-uniform computation \((M,\{A_n\})\), can define equivalent \{C_n\}

- Advice \(A_n\) is hard-wired into circuit \(C_n\)
Boolean Circuits

- Non-uniformity: circuit family \( \{C_n\} \)

- Given non-uniform computation \((M,\{A_n\})\), can define equivalent \( \{C_n\} \)

- Advice \( A_n \) is hard-wired into circuit \( C_n \)

- Size of circuit polynomially related to running time of TM
Boolean Circuits

- Non-uniformity: circuit family \( \{C_n\} \)

- Given non-uniform computation \((M, \{A_n\})\), can define equivalent \( \{C_n\} \)

- Advice \( A_n \) is hard-wired into circuit \( C_n \)

- Size of circuit polynomially related to running time of TM

Size = no. of wires
Boolean Circuits

- Non-uniformity: circuit family \( \{C_n\} \)

- Given non-uniform computation \((M,\{A_n\})\), can define equivalent \( \{C_n\} \)

  Advice \( A_n \) is hard-wired into circuit \( C_n \)

  Size of circuit polynomially related to running time of TM

- Conversely, given \( \{C_n\} \), can use description of \( C_n \) as advice \( A_n \) for a “universal” TM
Boolean Circuits

- Non-uniformity: circuit family \( \{C_n\} \)

- Given non-uniform computation \((M, \{A_n\})\), can define equivalent \( \{C_n\} \)

  - Advice \( A_n \) is hard-wired into circuit \( C_n \)

  - Size of circuit polynomially related to running time of TM

- Conversely, given \( \{C_n\} \), can use description of \( C_n \) as advice \( A_n \) for a “universal” TM

  - \( |A_n| \) comparable to size of circuit \( C_n \)
SIZE(T)
SIZE(T)

- SIZE(T): languages solved by circuit families of size T(n)
SIZE(T)

- SIZE(T): languages solved by circuit families of size $T(n)$
- $P/poly = SIZE(poly)$
SIZE(T)

- SIZE(T): languages solved by circuit families of size $T(n)$
- $P/poly = SIZE(poly)$
  - $SIZE(poly) \subseteq P/poly$: Size $T$ circuit can be described in $O(T \log T)$ bits (advice). Universal TM can evaluate this circuit in poly time
SIZE(T)

- SIZE(T): languages solved by circuit families of size $T(n)$
- $P/poly = SIZE(poly)$
  - $SIZE(poly) \subseteq P/poly$: Size $T$ circuit can be described in $O(T \log T)$ bits (advice). Universal TM can evaluate this circuit in poly time
  - $P/poly \subseteq SIZE(poly)$: Transformation from Cook’s theorem, with advice string hardwired into circuit
SIZE bounds
SIZE bounds

- All languages (decidable or not) are in SIZE(T) for T=O(n2^n)
SIZE bounds

- All languages (decidable or not) are in SIZE(T) for T=O(n2^n)
- Circuit encodes truth-table
SIZE bounds

- All languages (decidable or not) are in SIZE(T) for T=O(n2^n)
  - Circuit encodes truth-table
- Most languages need circuits of size \( \Omega(2^n/n) \)
SIZE bounds

- All languages (decidable or not) are in SIZE(T) for $T = O(n2^n)$
  - Circuit encodes truth-table
- Most languages need circuits of size $\Omega(2^n/n)$
- Number of circuits of size $T$ is at most $T^{2^T}$
SIZE bounds

- All languages (decidable or not) are in SIZE(T) for $T=O(n2^n)$
  - Circuit encodes truth-table
- Most languages need circuits of size $\Omega(2^n/n)$
  - Number of circuits of size $T$ is at most $T^{2T}$
- If $T = 2^n/4n$, say, $T^{2T} < 2^{(2^n)/2}$
SIZE bounds

- All languages (decidable or not) are in \( \text{SIZE}(T) \) for \( T = O(n2^n) \)
  - Circuit encodes truth-table
- Most languages need circuits of size \( \Omega(2^n/n) \)
  - Number of circuits of size \( T \) is at most \( T^{2^T} \)
    - If \( T = 2^{n/4}n \), say, \( T^{2^T} < 2^{(2^{n})/2} \)
  - Number of languages = \( 2^{2^n} \)
SIZE hierarchy
SIZE hierarchy

\[ \text{SIZE}(T') \subsetneq \text{SIZE}(T) \text{ if } T = \Omega(t2^t) \text{ and } T' = O(2^t/t) \]
SIZE hierarchy

\[ \text{SIZE}(T') \subseteq \text{SIZE}(T) \text{ if } T = \Omega(t2^t) \text{ and } T' = O(2^t/t) \]

- Consider functions on \( t \) bits (ignoring \( n-t \) bits)
SIZE hierarchy

\[ \text{SIZE}(T') \subsetneq \text{SIZE}(T) \text{ if } T = \Omega(t2^t) \text{ and } T' = O(2^t/t) \]

- Consider functions on \( t \) bits (ignoring \( n-t \) bits)
- All of them in \( \text{SIZE}(T) \), most not in \( \text{SIZE}(T') \)
Uniform Circuits
Uniform Circuits

Circuits are interesting for their structure too (not just size)!
Uniform Circuits

- Circuits are interesting for their structure too (not just size)!
- Uniform circuit family: constructed by a TM
Uniform Circuits

- Circuits are interesting for their structure too (not just size)!
- Uniform circuit family: constructed by a TM
  - Undecidable languages are undecidable for these circuits families
Uniform Circuits

- Circuits are interesting for their structure too (not just size)!
- Uniform circuit family: constructed by a TM
  - Undecidable languages are undecidable for these circuits families
  - Can relate their complexity classes to classes defined using TMs
Uniform Circuits

- Circuits are interesting for their structure too (not just size)!
- Uniform circuit family: constructed by a TM
  - Undecidable languages are undecidable for these circuits families
  - Can relate their complexity classes to classes defined using TMs
- Logspace-uniform:
Uniform Circuits

Circuits are interesting for their structure too (not just size)!

Uniform circuit family: constructed by a TM

Undecidable languages are undecidable for these circuits families

Can relate their complexity classes to classes defined using TMs

Logspace-uniform:

An $O(\log n)$ space TM can compute the circuit
NC and AC
NC and AC

NC and AC: languages decided by poly size and poly-log depth logspace-uniform circuits
NC and AC

- NC and AC: languages decided by poly size and poly-log depth logspace-uniform circuits
- NC with bounded fan-in and AC with unbounded fan-in
NC and AC

- NC and AC: languages decided by poly size and poly-log depth logspace-uniform circuits
- NC with bounded fan-in and AC with unbounded fan-in
- $\text{NC}^i$: decided by bounded fan-in logspace-uniform circuits of poly size and depth $O(\log^i n)$
NC and AC

NC and AC: languages decided by poly size and poly-log depth logspace-uniform circuits

NC with bounded fan-in and AC with unbounded fan-in

NC<sub>i</sub>: decided by bounded fan-in logspace-uniform circuits of poly size and depth $O(\log^i n)$

NC = $\cup_{i>0} NC^i$
NC and AC

- NC and AC: languages decided by poly size and poly-log depth logspace-uniform circuits

- NC with bounded fan-in and AC with unbounded fan-in

- $\text{NC}^i$: decided by bounded fan-in logspace-uniform circuits of poly size and depth $O(\log^i n)$

- $\text{NC} = \bigcup_{i>0} \text{NC}^i$

- Similarly $\text{AC}^i$ and $\text{AC} = \bigcup_{i>0} \text{AC}^i$
NC$^i$ and AC$^i$
$\mathsf{NC}^i$ and $\mathsf{AC}^i$

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

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

- Clearly $NC^i \subseteq AC^i$
$\text{NC}^i$ and $\text{AC}^i$

$\text{NC}^i \subseteq \text{AC}^i \subseteq \text{NC}^{i+1}$

- Clearly $\text{NC}^i \subseteq \text{AC}^i$

- $\text{AC}^i \subseteq \text{NC}^{i+1}$ because polynomial fan-in can be reduced to constant fan-in by using a log depth tree
$NC^i$ and $AC^i$

- $NC^i \subseteq AC^i \subseteq NC^{i+1}$
- Clearly $NC^i \subseteq AC^i$
- $AC^i \subseteq NC^{i+1}$ because polynomial fan-in can be reduced to constant fan-in by using a log depth tree
- So $NC = AC$
NC and P
NC and $\mathbb{P}$

$\mathbb{NC} \subseteq \mathbb{P}$
NC and P

- $\text{NC} \subseteq \text{P}$

- Build the circuit in logspace (so poly time) and evaluate it in time polynomial in the size of the circuit
NC and P

\[ \text{NC} \subseteq \text{P} \]

- Build the circuit in logspace (so poly time) and evaluate it in time polynomial in the size of the circuit

- Open problem: Is NC = P?
Motivation for NC
Motivation for NC

Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time
Motivation for NC

 rápido paralelo computación se (loosely) modeled as
 having poly many processors and taking poly-log time

 Corresponds to NC
Motivation for NC

- Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time
- Corresponds to NC
- Depth translates to time
Motivation for NC

Fast parallel computation is (loosely) modeled as having poly many processors and taking poly-log time

- Corresponds to NC
- Depth translates to time
- Total "work" is size of the circuit