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

- DTIME(T)/a

P/poly and P/log

- 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

- DTIME(T)/a
  - Languages decided by a TM in time T(n) using non-uniform advice of length a(n)
- P/poly = $\cup_{c,d,k>0}$ DTIME(kn^c)/kn^d

P/poly and P/log

- DTIME(T)/a
  - Languages decided by a TM in time T(n) using non-uniform advice of length a(n)
- P/poly = $\cup_{c,d,k>0}$ DTIME(kn^c)/kn^d
- P/log = $\cup_{c,k>0}$ DTIME(kn^c)/k log n

NP vs. P/log, P/poly

NP vs. P/log, P/poly

- 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 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?

$NP \subseteq P/\log \Rightarrow NP=P$

$NP \subseteq P/\log \Rightarrow NP=P$

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

Search using Decision

Search using Decision

- Suppose given "oracles" for deciding all NP languages, can we easily find certificates?

Search using Decision

- Suppose given "oracles" for deciding all NP languages, can we easily find certificates?
  - Yes! So, if decision easy (i.e., oracles realizable), then search is easy too!

Search using Decision

- Suppose given "oracles" for deciding all NP languages, can we easily find certificates?
  - Yes! So, if decision easy (i.e., oracles realizable), then search is easy too!
- Say need to find w s.t. $(x,w) \in L'$

Search using Decision

- Suppose given "oracles" for deciding all NP languages, can we easily find certificates?
  - Yes! So, if decision easy (i.e., oracles realizable), then search is easy too!
- Say need to find w s.t. $(x,w) \in L'$
  - 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).

Search using Decision

- Suppose given "oracles" for deciding all NP languages, can we easily find certificates?
  - Yes! So, if decision easy (i.e., oracles realizable), then search is easy too!
- Say need to find w s.t. $(x,w) \in L'$
  - 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). One of the two must be positive: say $(x,0) \in L_1$; then first bit of w be 0.

Search using Decision

- Suppose given "oracles" for deciding all NP languages, can we easily find certificates?
  - Yes! So, if decision easy (i.e., oracles realizable), then search is easy too!
- Say need to find w s.t. $(x,w) \in L'$
  - 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). One of the two must be positive: say $(x,0) \in L_1$; then first bit of w be 0.
  - For next bit query oracle with (x,00) and (x,01)

Search using Decision

- Suppose given "oracles" for deciding all NP languages, can we easily find certificates?
  - Yes! So, if decision easy (i.e., oracles r
    then search is easy too!

Use L_2 so that (x,z,pad) in L_2 iff (x,z) in L_1. Can query L_2 with same size instances

- Say need to find w s.t. $(x,w) \in L'$
  - 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). One of the two must be positive: say $(x,0) \in L_1$; then first bit of w be 0.
  - For next bit query oracle with (x,00) and (x,01)

$NP \subseteq P/\log \Rightarrow NP = P$

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

$NP \subseteq P/\log \Rightarrow NP=P$

- 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 <u>find</u> witness (via self-reduction)

$NP \subseteq P/\log \Rightarrow NP=P$

- 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 <u>find</u> witness (via self-reduction)
- Guess advice (poly many), and for each guessed advice, run the TM and see if it finds witness

$NP \subseteq P/\log \Rightarrow NP=P$

- 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 <u>find</u> witness (via self-reduction)
- 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

$NP \subseteq P/poly \Rightarrow PH = \Sigma_2^P$

$NP \subseteq P/poly \Rightarrow PH=\Sigma_2^P$

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

$NP \subseteq P/poly \Rightarrow PH=\Sigma_2^P$

- 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

  $L' = \{(x,w_1) \mid \exists w_2\ F(x,w_1,w_2)\} \in NP$

$NP \subseteq P/poly \Rightarrow PH=\Sigma_2^P$

- Will show $\Pi_2^P = \Sigma_2^P$
- Consider $L = \{x|\ \forall w_1\ (x,w_1) \in L'\ \} \in \Pi_2^P$ where

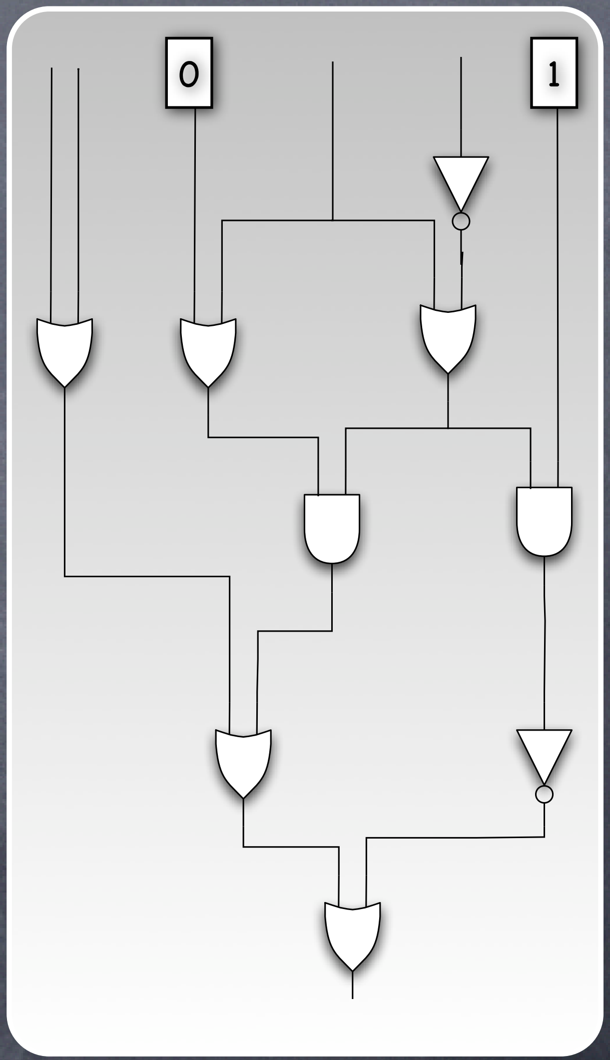
  $L' = \{(x,w_1)|\ \exists w_2\ F(x,w_1,w_2)\} \in NP$

  - 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. $F(x,w_1,w_2)$

$NP \subseteq P/poly \Rightarrow PH = \Sigma_2^P$

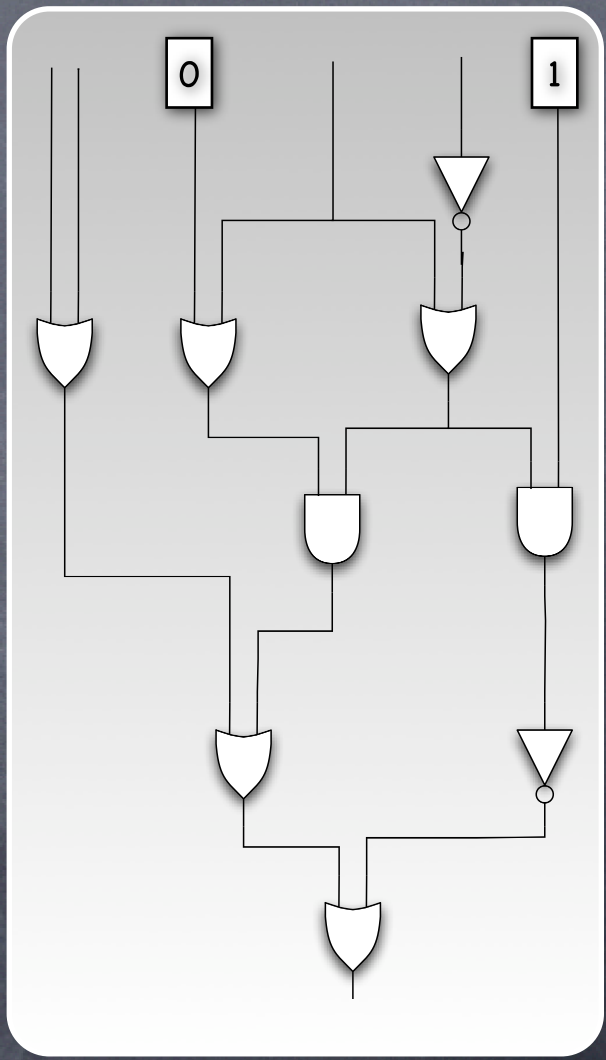
- Will show $\Pi_2^P = \Sigma_2^P$
- Consider $L = \{x \mid \forall w_1\ (x,w_1) \in L' \} \in \Pi_2^P$ where
  $L' = \{(x,w_1) \mid \exists w_2\ F(x,w_1,w_2)\} \in NP$
  - If $NP \subseteq P/poly$ then consider M with advice $\{A_n\}$ which finds witness for L': i.e. if $(x,w_1) \in L'$, then $M(x,w_1; A_n)$ outputs a witness w_2 s.t. $F(x,w_1,w_2)$
  - $L = \{x \mid \exists z\ \forall w_1\ F(x, w_1, M(x,w_1; z)) \}$

Boolean Circuits



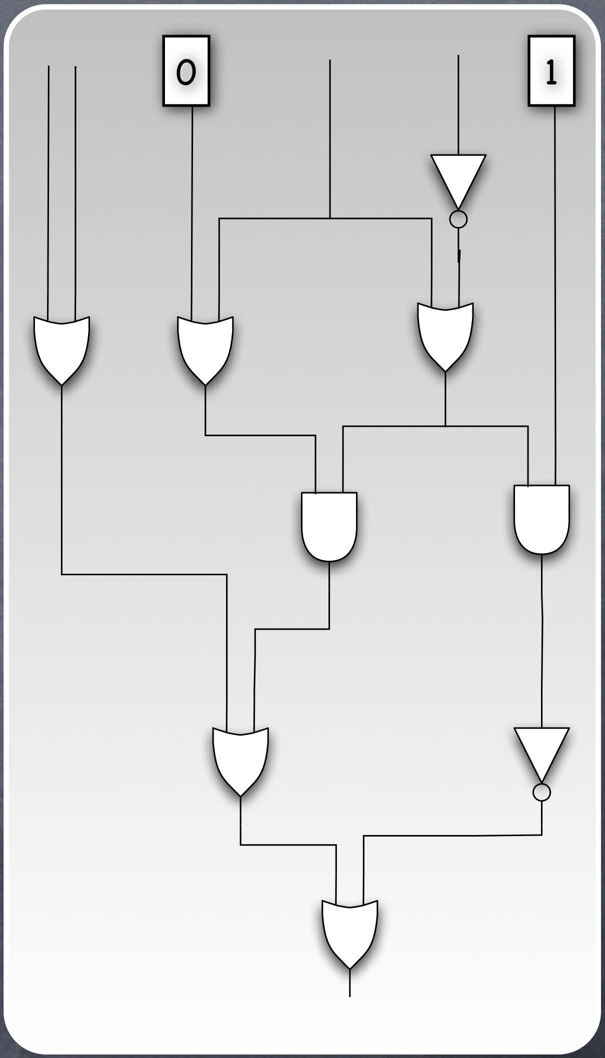
Boolean Circuits

- Directed acyclic graph



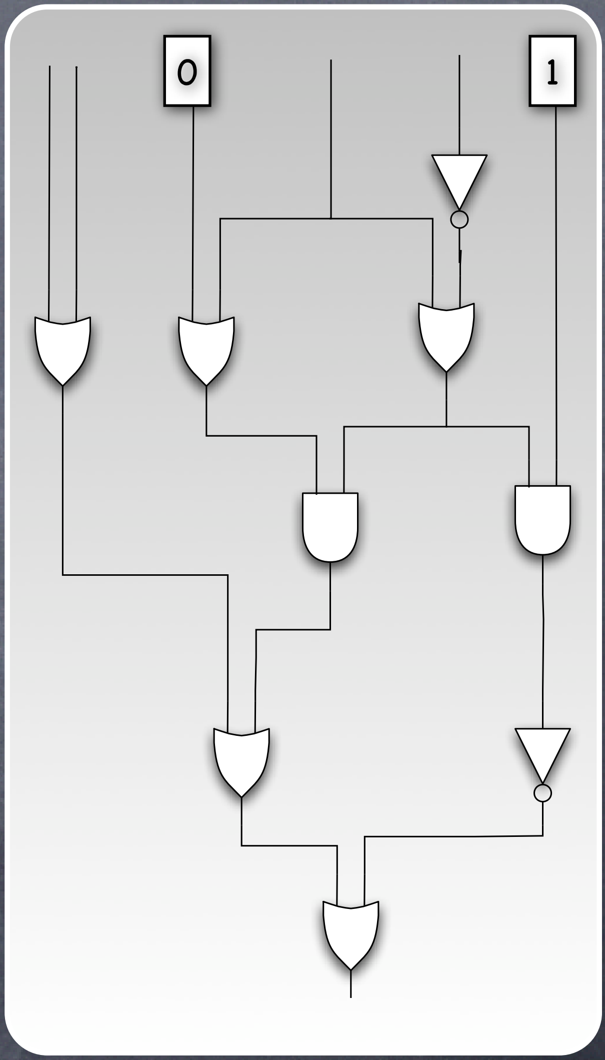
Boolean Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)



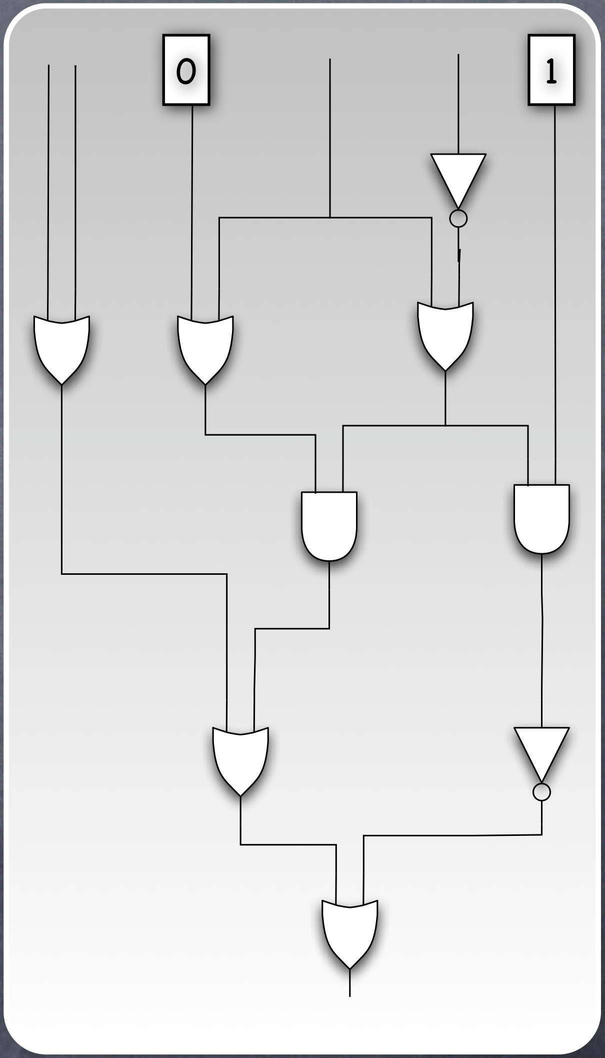
Boolean Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires



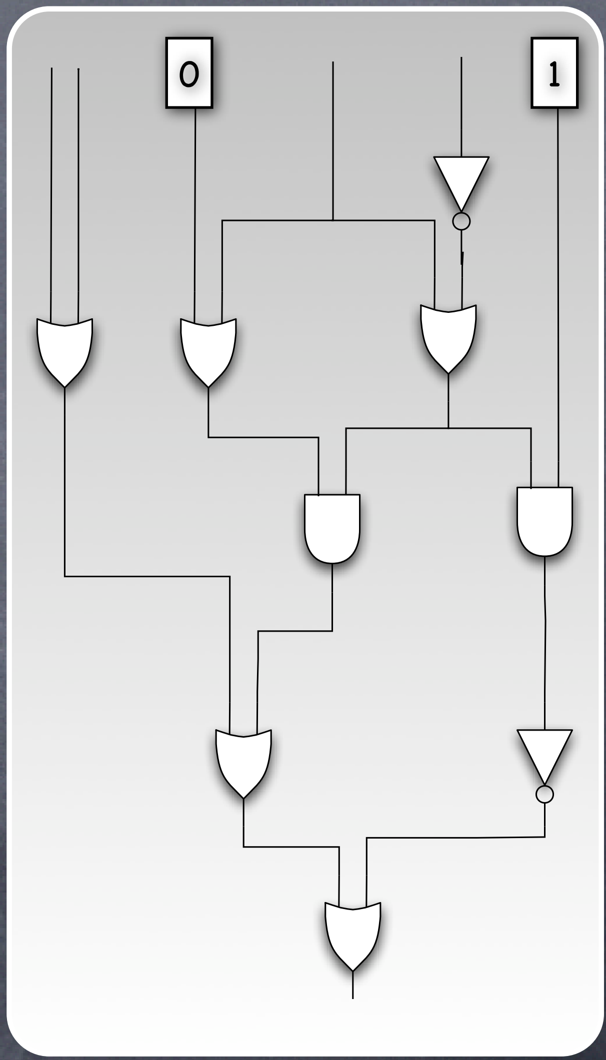
Boolean Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - AND/OR fan-ins can be bounded (say two) or unbounded



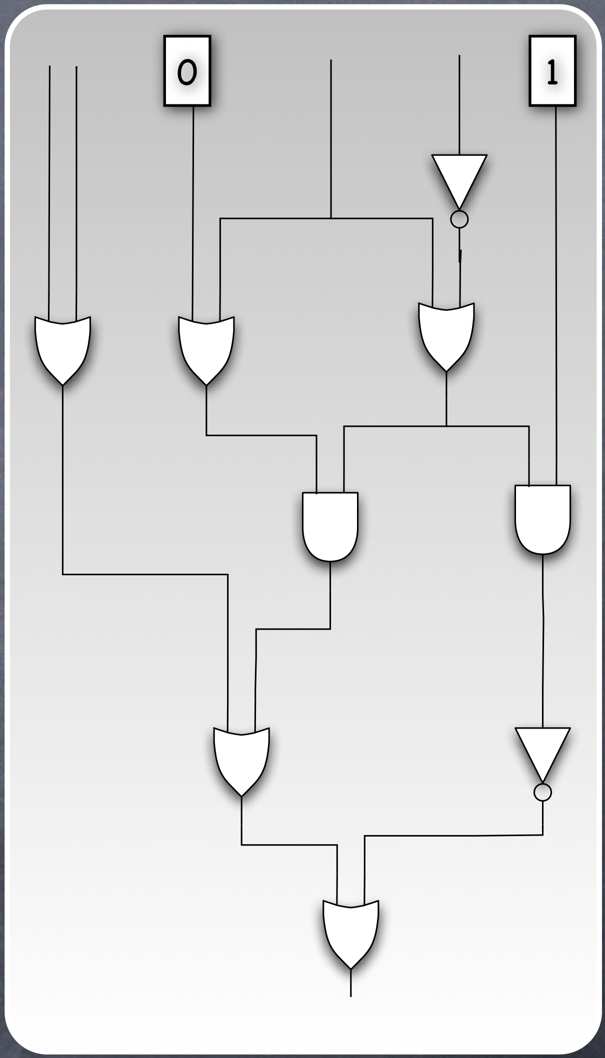
Boolean Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - AND/OR fan-ins can be bounded (say two) or unbounded
  - Acyclic: output well-defined



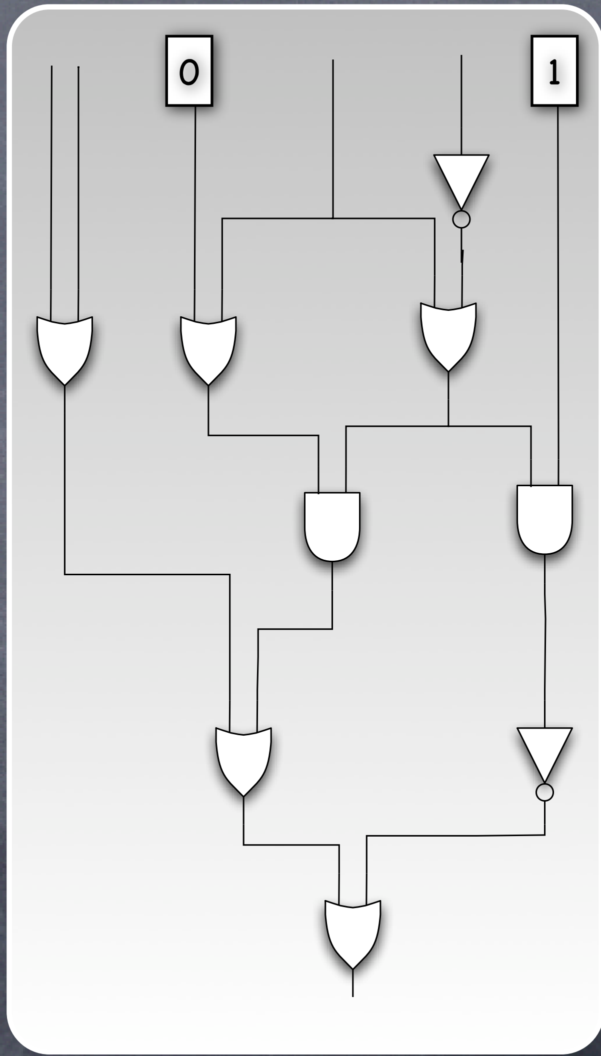
Boolean Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - AND/OR fan-ins can be bounded (say two) or unbounded
  - Acyclic: output well-defined
    - Note: no memory gates

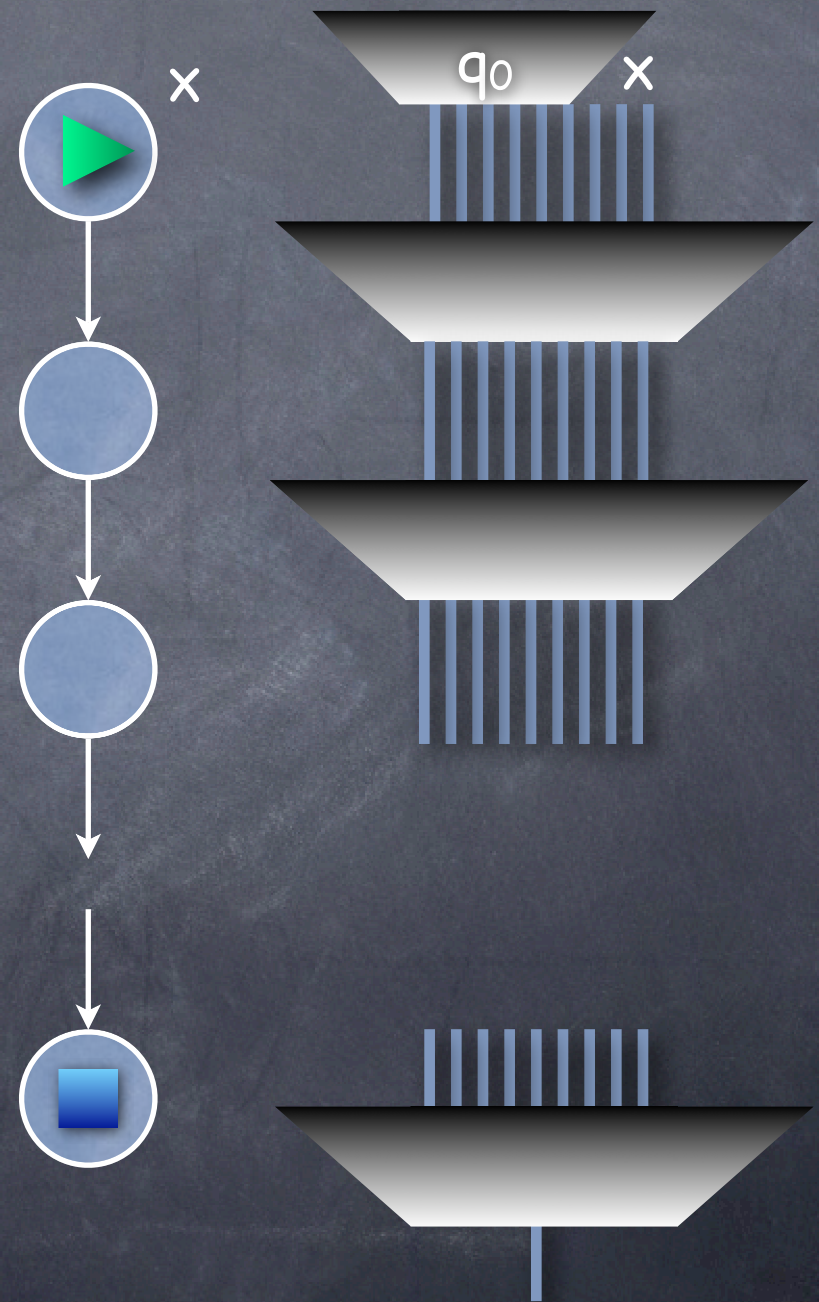


Boolean Circuits

- Directed acyclic graph
  - Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
  - Edges: Boolean valued wires
  - AND/OR fan-ins can be bounded (say two) or unbounded
  - Acyclic: output well-defined
    - Note: no memory gates
  - Size of circuit: number of wires

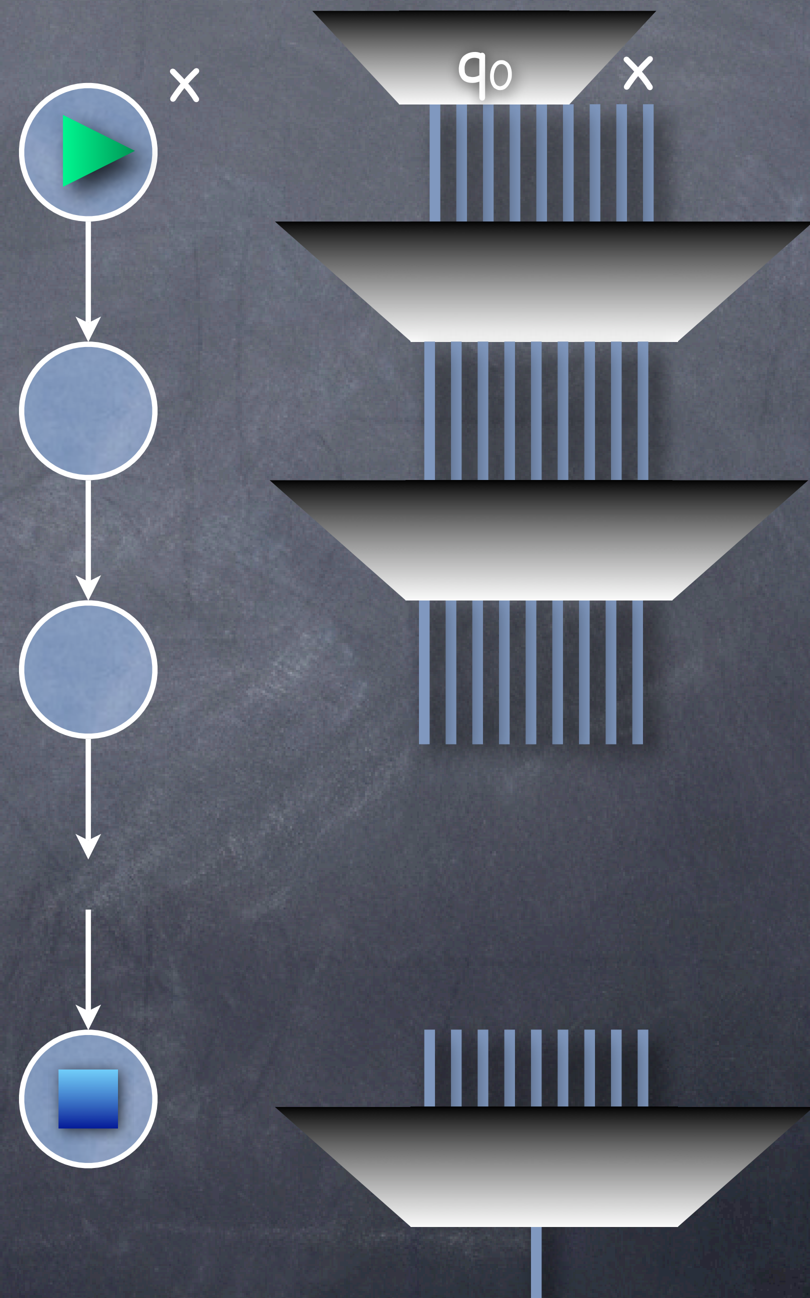


Boolean Circuits



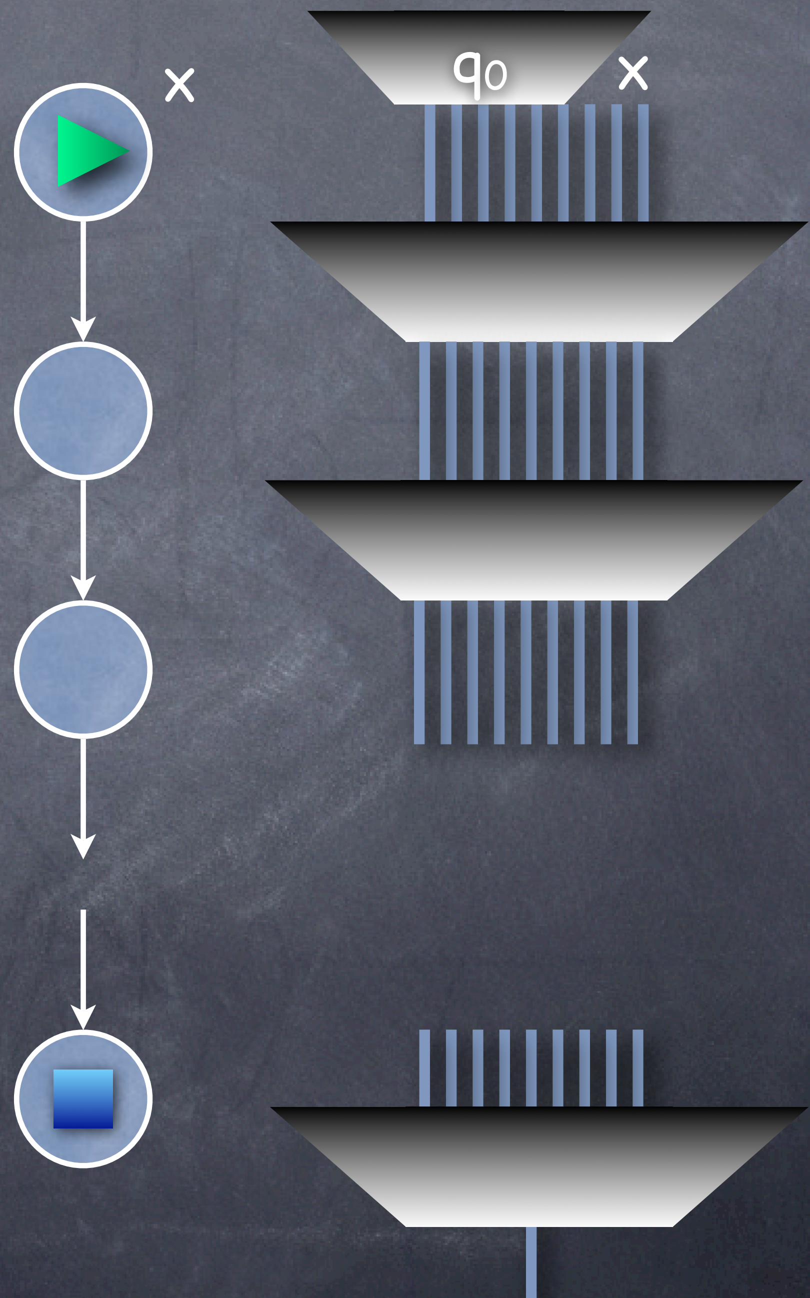
Boolean Circuits

- Recall: a TM's execution on inputs of fixed length can be captured by a Boolean circuit



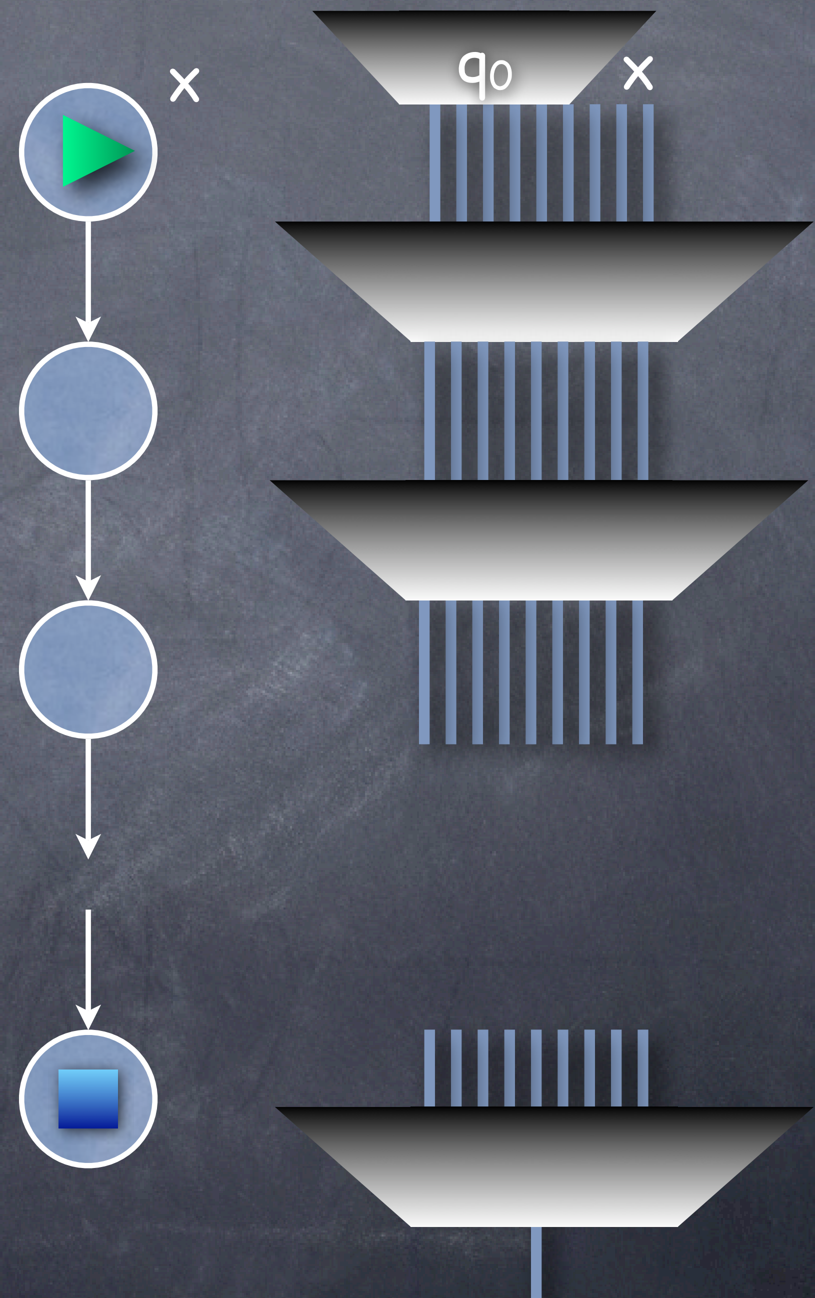
Boolean Circuits

- Recall: a TM's execution on inputs of fixed length can be captured by a Boolean circuit
  - From proof of Cook's theorem



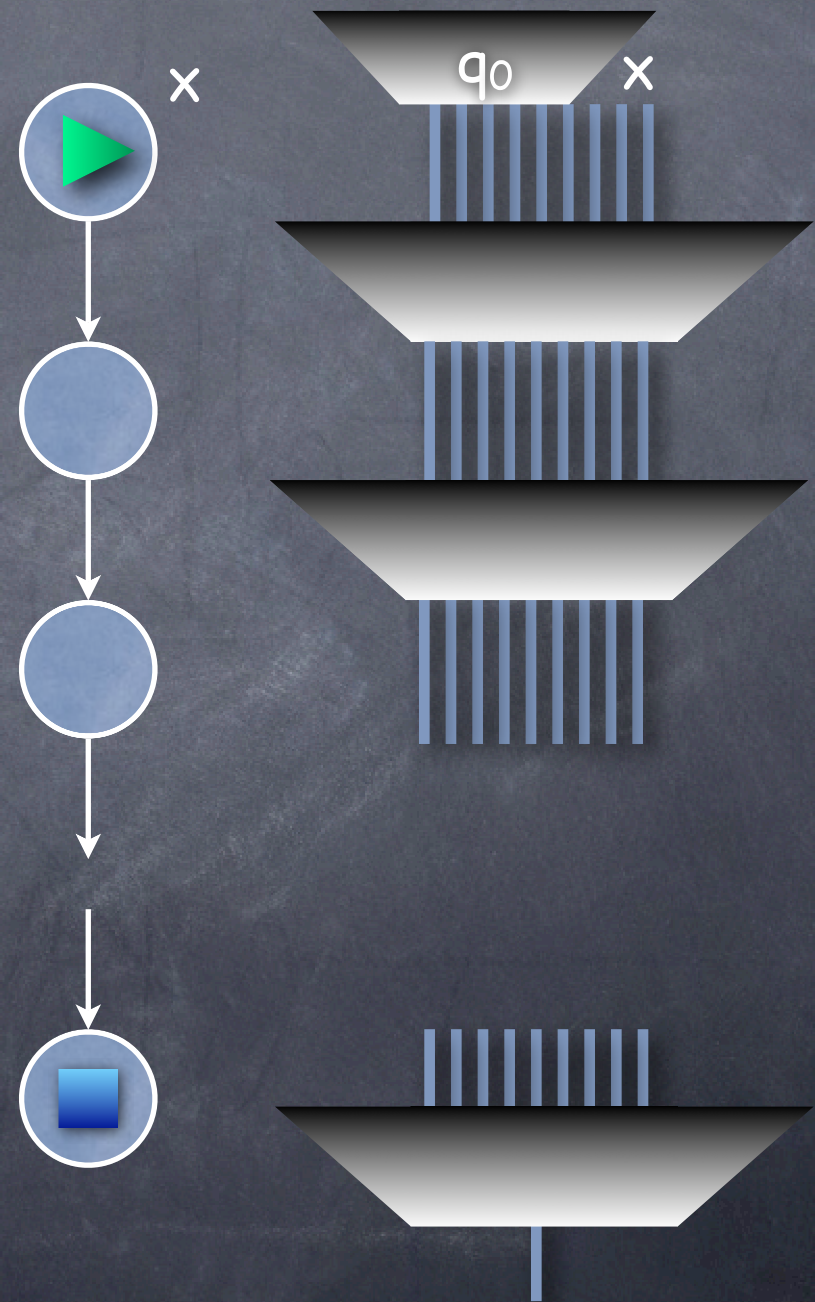
Boolean Circuits

- Recall: a TM's execution on inputs of fixed length can be captured by a Boolean circuit
  - From proof of Cook's theorem
  - Size of circuit polynomially related to running time of TM

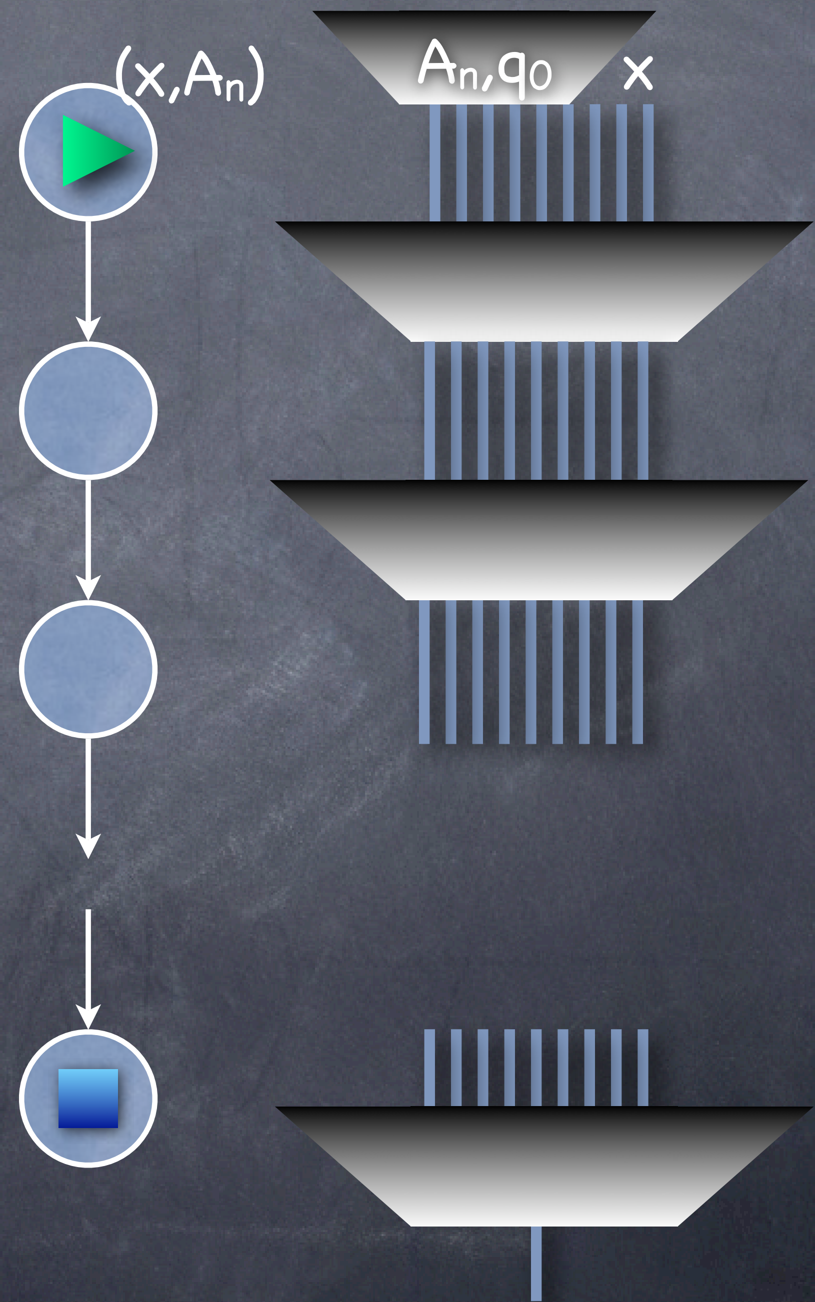


Boolean Circuits

- Recall: a TM's execution on inputs of fixed length can be captured by a Boolean circuit
  - From proof of Cook's theorem
  - Size of circuit polynomially related to running time of TM
    - If poly time TM, then poly sized circuit

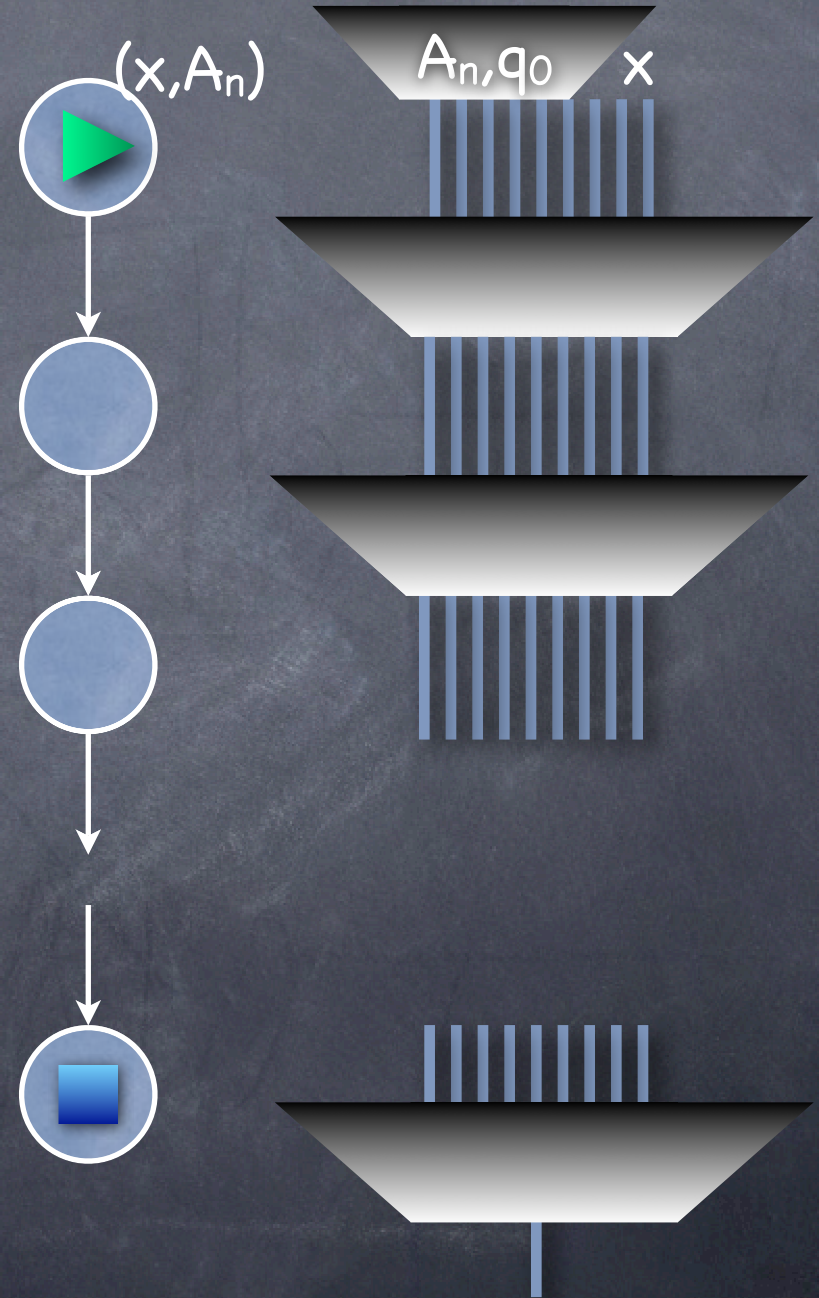


Boolean Circuits



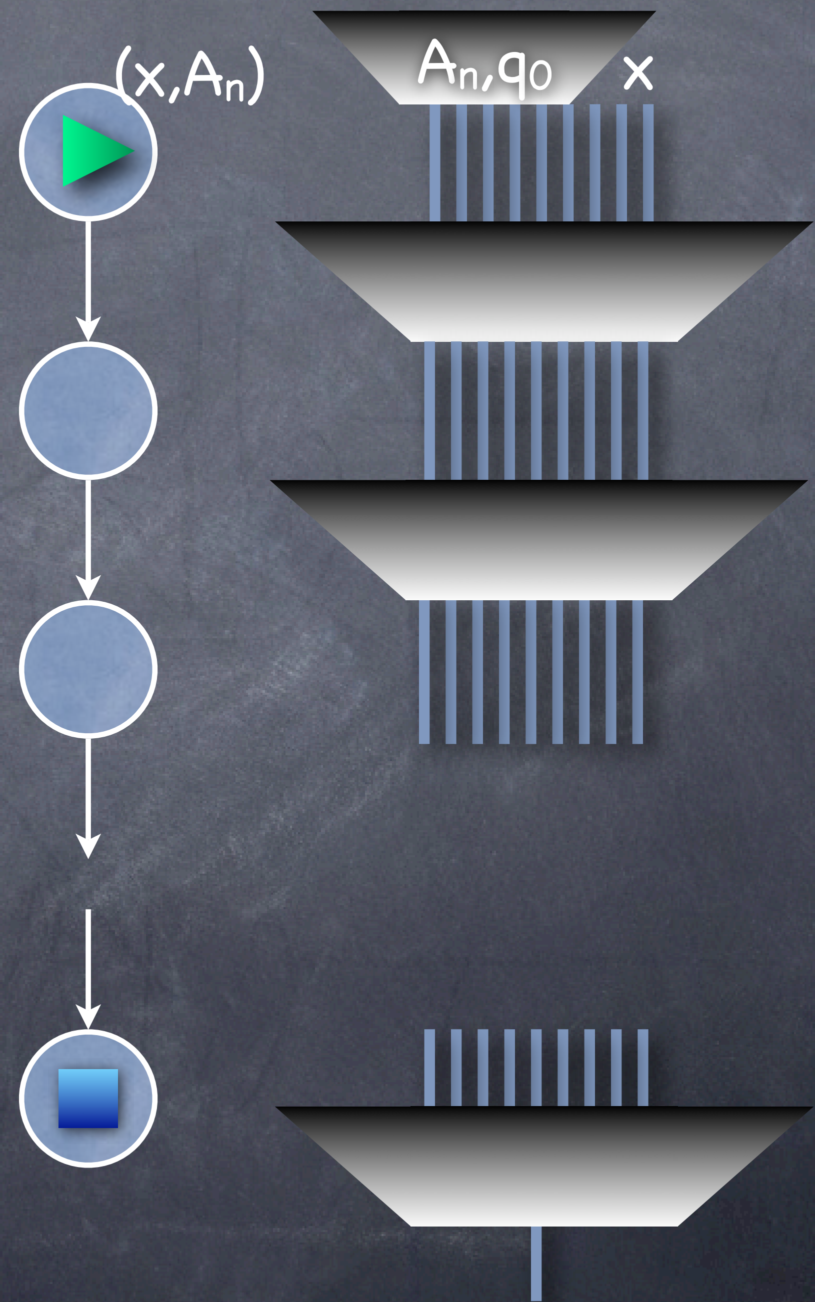
Boolean Circuits

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



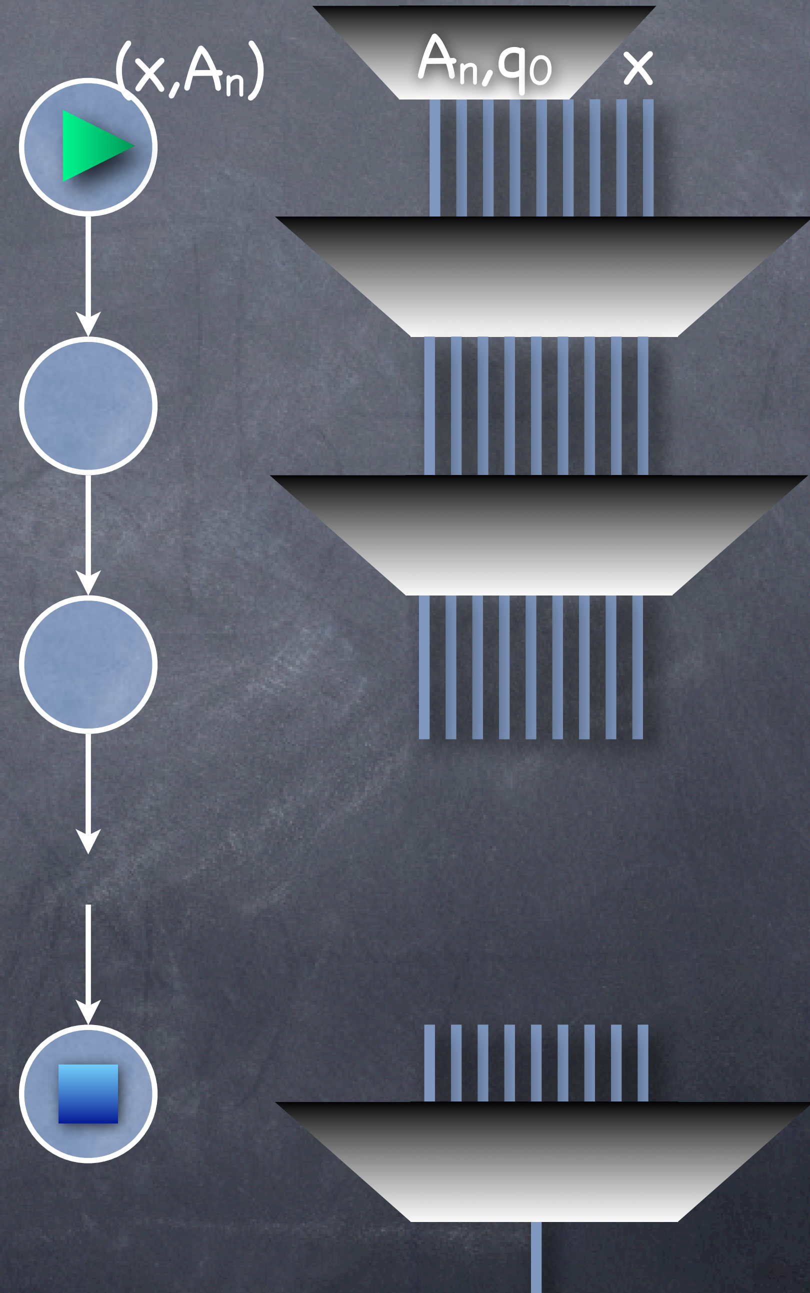
Boolean Circuits

- Non-uniformity: circuit family $\{C_n\}$
  - Given non-uniform computation $(M,\{A_n\})$ can define equivalent $\{C_n\}$



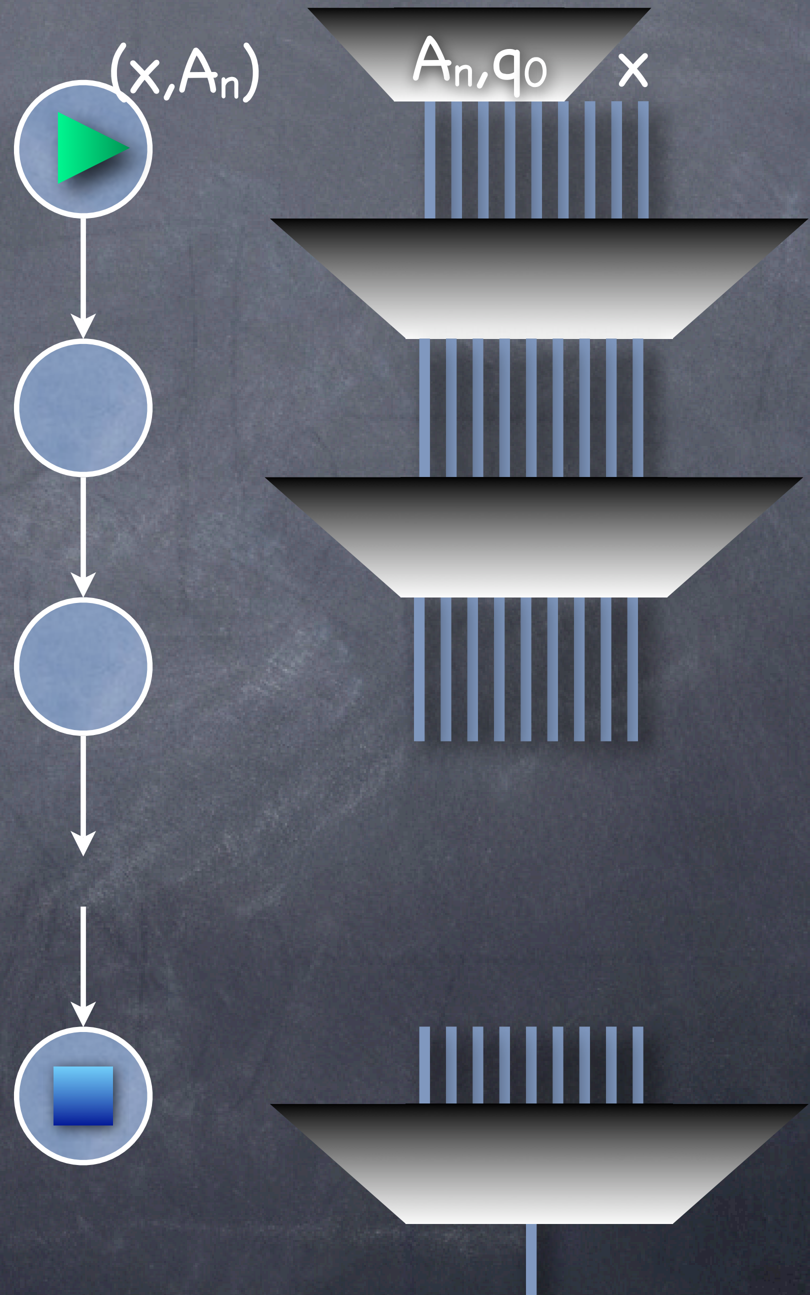
Boolean Circuits

- Non-uniformity: circuit family $\{C_n\}$
  - Given non-uniform computation $(M,\{A_n\})$ can define equivalent $\{C_n\}$
    - Advice A_n is hard-wired into circuit C_n



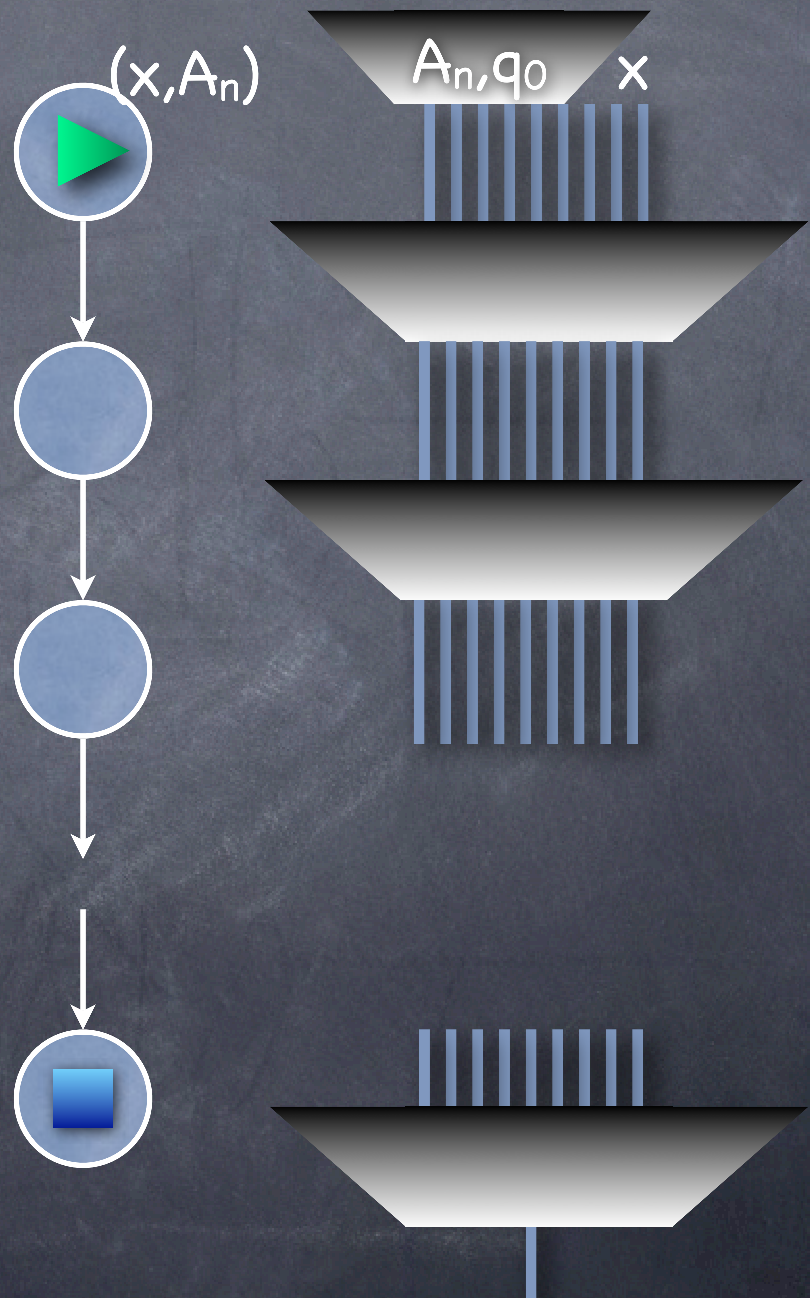
Boolean Circuits

- 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
    - Doesn't affect circuit size



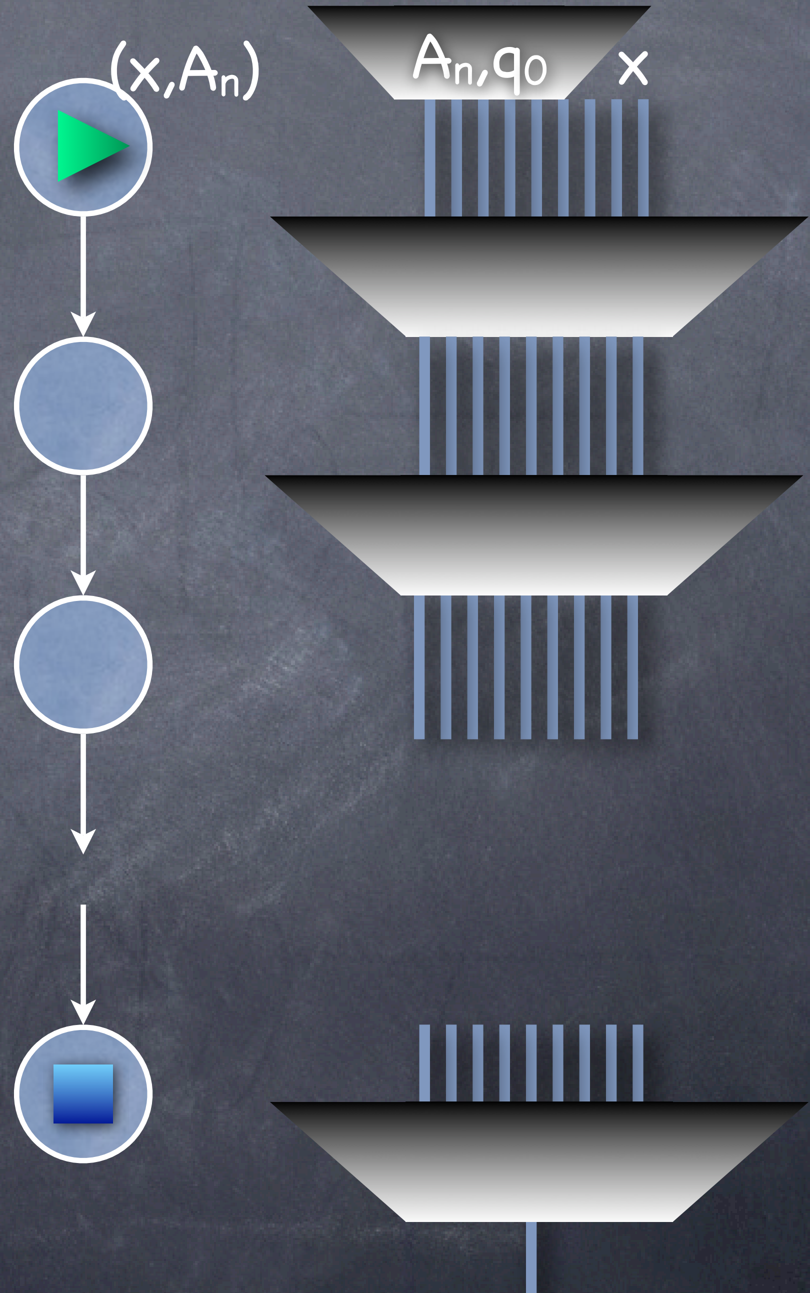
Boolean Circuits

- 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
    - Doesn't affect circuit size
  - 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
    - Doesn't affect circuit size
  - 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) ⊆ 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 ⊆ 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^{2T}

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 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, $T^{2T} < 2^{(2\hat{}n)/2}$
  - Number of languages = $2^{2\hat{}n}$

SIZE hierarchy

SIZE hierarchy

- $\mathrm{SIZE}(T') \subsetneq \mathrm{SIZE}(T)$ if $T=\Omega(t2^t)$ and $T'=O(2^t/t)$

SIZE hierarchy

- SIZE(T′) $\subsetneq$ SIZE(T) if T=Ω(t2^t) and T′=O(2^t/t)
  - Consider functions on t bits (ignoring n-t bits)

SIZE hierarchy

- SIZE(T′) $\subsetneq$ SIZE(T) if $T=\Omega(t2^t)$ and $T'=O(2^t/t)$
  - Consider functions on t bits (ignoring n-t bits)
    - All of them in SIZE(T), most not in 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
  - 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 <u>and</u> poly-log depth logspace-uniform circuits
  - NC with bounded fan-in and AC with unbounded fan-in
  - NC^i: 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
  - NC^i: decided by bounded fan-in logspace-uniform circuits of poly size and depth $O(\log^i n)$
  - $NC = \cup_{i>0} NC^i$
  - Similarly AC^i and $AC = \cup_{i>0} AC^i$

NC^i and AC^i

NC^i and AC^i

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

NC^i and AC^i

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

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

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 P

- $NC \subseteq P$

NC and P

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

NC and P

- $NC \subseteq P$
  - 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

- Fast parallel computation is (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