Circuit complexity

K. Subramani[1]

[1] Lane Department of Computer Science and Electrical Engineering
West Virginia University

April 1, 2009

Outline

Outline

Outline

Outline

Outline

Circuit Refresher

Definition

A boolean circuit is a directed acyclic graph $G = \langle V, E \rangle$, where the nodes in $V = \{1, 2, \ldots, n\}$ are called gates and the edges are of the form (i, j), $i < j$.

Observation

A circuit with n variable inputs can compute any boolean function with n variables.

Observation

Alternatively, a circuit accepts some subset of strings in $x = x_1 x_2 \ldots x_n \in \{0,1\}^n$, where the i^{th} input is **true** *if and only if $x_i = 1$.*

Circuit Refresher

Definition

A boolean circuit is a directed acyclic graph $G = \langle V, E \rangle$, where the nodes in $V = \{1, 2, \ldots, n\}$ are called gates and the edges are of the form (i, j), $i < j$.

Observation

A circuit with n variable inputs can compute any boolean function with n variables.

Observation

Alternatively, a circuit accepts some subset of strings in $x = x_1 x_2 \ldots x_n \in \{0,1\}^n$, where the i^{th} input is **true** *if and only if $x_i = 1$.*

Circuit Refresher

Definition

A boolean circuit is a directed acyclic graph $G = \langle V, E \rangle$, where the nodes in $V = \{1, 2, \ldots, n\}$ are called gates and the edges are of the form (i, j), $i < j$.

Observation

A circuit with n variable inputs can compute any boolean function with n variables.

Observation

Alternatively, a circuit accepts some subset of strings in $x = x_1 x_2 \ldots x_n \in \{0, 1\}^n$, where the i^{th} input is **true** *if and only if $x_i = 1$.*

Circuits as Language acceptors

Definition

The size of a circuit is the number of gates in it.

Definition

A family of circuits is an infinite sequence $\mathcal{C} = (C_0, C_1, \ldots)$ of boolean circuits, where C_n has n input variables. We say that language $L \subseteq \{0,1\}^*$ has polynomial circuits, if there exists a family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ such that

(i) $|C_n| \leq p(n)$, where p is some fixed polynomial, and

(ii) $\forall x \in \{0,1\}^*$ $x \in L \leftrightarrow$ the output of $C_{|x|}$ is **true** under the assignment that forces the i^{th} input variable to be **true** when $x_i = 1$ and 0 otherwise.

Circuits as Language acceptors

Definition

The size of a circuit is the number of gates in it.

Definition

A family of circuits is an infinite sequence $\mathcal{C} = (C_0, C_1, \ldots)$ of boolean circuits, where C_n has n input variables. We say that language $L \subseteq \{0,1\}^*$ has polynomial circuits, if there exists a family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ such that

(i) $|C_n| \leq p(n)$, where p is some fixed polynomial, and

(ii) $\forall x \in \{0,1\}^*$ $x \in L \leftrightarrow$ the output of $C_{|x|}$ is **true** under the assignment that forces the i^{th} input variable to be **true** when $x_i = 1$ and 0 otherwise.

Circuits as Language acceptors

Definition

The size of a circuit is the number of gates in it.

Definition

A family of circuits is an infinite sequence $\mathcal{C} = (C_0, C_1, \ldots)$ of boolean circuits, where C_n has n input variables. We say that language $L \subseteq \{0,1\}^*$ has polynomial circuits, if there exists a family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ such that

(i) $|C_n| \leq p(n)$, where p is some fixed polynomial, and

(ii) $\forall x \in \{0,1\}^*$ $x \in L \leftrightarrow$ the output of $C_{|x|}$ is **true** under the assignment that forces the i^{th} input variable to be **true** when $x_i = 1$ and 0 otherwise.

Circuits as Language acceptors

Definition

The size of a circuit is the number of gates in it.

Definition

A family of circuits is an infinite sequence $\mathcal{C} = (C_0, C_1, \ldots)$ of boolean circuits, where C_n has n input variables. We say that language $L \subseteq \{0,1\}^*$ has polynomial circuits, if there exists a family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ such that

(i) $|C_n| \leq p(n)$, where p is some fixed polynomial, and

(ii) $\forall x \in \{0,1\}^*$ $x \in L \leftrightarrow$ the output of $C_{|x|}$ is **true** under the assignment that forces the i^{th} input variable to be **true** when $x_i = 1$ and 0 otherwise.

Outline

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Reduction

We reduce Reachability to the Circuit Value problem. Let $G = \langle V, E \rangle$ be a graph and let $(1, n)$ denote the source and target of the reachability problem respectively.

Steps

(i) *Construct gates g_{ijk} with $1 \leq i, j \leq n$ and $0 \leq k \leq n$; intuitively, g_{ijk} is* **true** *if and only if there is a path in G from i to j using all intermediate nodes in the set $S_k = \{1, 2, \ldots, k\}$.*

(ii) *Construct gates h_{ijk} with $1 \leq i, j, k \leq n$; intuitively, h_{ijk} is* **true** *if and only if there is a path in G from i to j with all intermediate nodes in S_k* **and** *k is an intermediate node.*

(iii) *g_{ij0} (input gate) is* **true** *if $i = j$ or there is an edge from i to j; it is* **false** *otherwise.*

(iv) *For $k = 1, 2, \ldots, n$, h_{ijk} is an* **AND** *gate with predecessors $g_{i,k,k-1}$ and $g_{k,j,k-1}$.*

(v) *For $k = 1, 2, \ldots n$, g_{ijk} is an* **OR** *gate with predecessors $g_{i,j,k-1}$ and h_{ijk}.*

Note

The circuit is merely a hardware representation of the Floyd-Warshall algorithm discussed in Homework I!

Circuits and Complexity

Proposition

All languages in **P** *have polynomial sized circuits.*

Proof.

We actually reduced an arbitrary Turing Machine that halts in polynomial time to a variable-free circuit, when we showed that CIRCUIT-VALUE is **P-complete**. □

Theorem

There are undecidable languages which have polynomial circuits.

Proof.

Let $L \subseteq \{0,1\}^*$ be any undecidable language in the alphabet $\{0,1\}$ and let $U \subseteq 1^*$ be the language $\{1^n$: the binary expansion of n is in $L.\}$ Clearly U is undecidable (Why?) Now consider the following family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$. If $1^n \in U$, then C_n consists of $(n-1)$ **AND** gates that compute the conjunction of all the inputs. If $1^n \notin U$, then C_n consists of its input gates and an output gate that is **false**. Thus, for all inputs, $1^n \in U \leftrightarrow C_n$ outputs **true**. □

Circuits and Complexity

Proposition

All languages in **P** *have polynomial sized circuits.*

Proof.

We actually reduced an arbitrary Turing Machine that halts in polynomial time to a variable-free circuit, when we showed that CIRCUIT-VALUE is **P-complete**. □

Theorem

There are undecidable languages which have polynomial circuits.

Proof.

Let $L \subseteq \{0,1\}^*$ be any undecidable language in the alphabet $\{0,1\}$ and let $U \subseteq 1^*$ be the language $\{1^n$: the binary expansion of n is in $L.\}$ Clearly U is undecidable (Why?) Now consider the following family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$. If $1^n \in U$, then C_n consists of $(n-1)$ **AND** gates that compute the conjunction of all the inputs. If $1^n \notin U$, then C_n consists of its input gates and an output gate that is **false**. Thus, for all inputs, $1^n \in U \leftrightarrow C_n$ outputs **true**. □

Circuits and Complexity

Proposition

All languages in **P** *have polynomial sized circuits.*

Proof.

We actually reduced an arbitrary Turing Machine that halts in polynomial time to a variable-free circuit, when we showed that CIRCUIT-VALUE is **P-complete**. □

Theorem

There are undecidable languages which have polynomial circuits.

Proof.

Let $L \subseteq \{0,1\}^*$ be any undecidable language in the alphabet $\{0,1\}$ and let $U \subseteq 1^*$ be the language $\{1^n$: the binary expansion of n is in $L.\}$ Clearly U is undecidable (Why?) Now consider the following family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$. If $1^n \in U$, then C_n consists of $(n-1)$ **AND** gates that compute the conjunction of all the inputs. If $1^n \notin U$, then C_n consists of its input gates and an output gate that is **false**. Thus, for all inputs, $1^n \in U \leftrightarrow C_n$ outputs **true**. □

Circuits and Complexity

Proposition

All languages in **P** *have polynomial sized circuits.*

Proof.

We actually reduced an arbitrary Turing Machine that halts in polynomial time to a variable-free circuit, when we showed that CIRCUIT-VALUE is **P-complete**. □

Theorem

There are undecidable languages which have polynomial circuits.

Proof.

Let $L \subseteq \{0,1\}^*$ be any undecidable language in the alphabet $\{0,1\}$ and let $U \subseteq 1^*$ be the language $\{1^n$: the binary expansion of n is in $L.\}$ Clearly U is undecidable (Why?) Now consider the following family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$. If $1^n \in U$, then C_n consists of $(n-1)$ **AND** gates that compute the conjunction of all the inputs. If $1^n \notin U$, then C_n consists of its input gates and an output gate that is **false**. Thus, for all inputs, $1^n \in U \leftrightarrow C_n$ outputs **true**. □

Circuits and Complexity

Proposition

All languages in **P** *have polynomial sized circuits.*

Proof.

We actually reduced an arbitrary Turing Machine that halts in polynomial time to a variable-free circuit, when we showed that CIRCUIT-VALUE is **P-complete**. □

Theorem

There are undecidable languages which have polynomial circuits.

Proof.

Let $L \subseteq \{0,1\}^*$ be any undecidable language in the alphabet $\{0,1\}$ and let $U \subseteq 1^*$ be the language $\{1^n$: the binary expansion of n is in $L.\}$ Clearly U is undecidable (Why?) Now consider the following family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$. If $1^n \in U$, then C_n consists of $(n-1)$ **AND** gates that compute the conjunction of all the inputs. If $1^n \notin U$, then C_n consists of its input gates and an output gate that is **false**. Thus, for all inputs, $1^n \in U \leftrightarrow C_n$ outputs **true**. □

Circuits and Complexity

Proposition

All languages in **P** *have polynomial sized circuits.*

Proof.

We actually reduced an arbitrary Turing Machine that halts in polynomial time to a variable-free circuit, when we showed that CIRCUIT-VALUE is **P-complete**. □

Theorem

There are undecidable languages which have polynomial circuits.

Proof.

Let $L \subseteq \{0,1\}^*$ be any undecidable language in the alphabet $\{0,1\}$ and let $U \subseteq 1^*$ be the language $\{1^n$: the binary expansion of n is in $L.\}$ Clearly U is undecidable (Why?) Now consider the following family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$. If $1^n \in U$, then C_n consists of $(n-1)$ **AND** gates that compute the conjunction of all the inputs. If $1^n \notin U$, then C_n consists of its input gates and an output gate that is **false**. Thus, for all inputs, $1^n \in U \leftrightarrow C_n$ outputs **true**. □

Circuits and Complexity

Proposition

All languages in **P** *have polynomial sized circuits.*

Proof.

We actually reduced an arbitrary Turing Machine that halts in polynomial time to a variable-free circuit, when we showed that CIRCUIT-VALUE is **P-complete**. □

Theorem

There are undecidable languages which have polynomial circuits.

Proof.

Let $L \subseteq \{0,1\}^*$ be any undecidable language in the alphabet $\{0,1\}$ and let $U \subseteq 1^*$ be the language $\{1^n : \text{the binary expansion of } n \text{ is in } L.\}$ Clearly U is undecidable (Why?) Now consider the following family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$. If $1^n \in U$, then C_n consists of $(n-1)$ **AND** gates that compute the conjunction of all the inputs. If $1^n \notin U$, then C_n consists of its input gates and an output gate that is **false**. Thus, for all inputs, $1^n \in U \leftrightarrow C_n$ outputs **true**. □

Circuits and Complexity

Proposition

All languages in **P** *have polynomial sized circuits.*

Proof.

We actually reduced an arbitrary Turing Machine that halts in polynomial time to a variable-free circuit, when we showed that CIRCUIT-VALUE is **P-complete**. □

Theorem

There are undecidable languages which have polynomial circuits.

Proof.

Let $L \subseteq \{0,1\}^*$ be any undecidable language in the alphabet $\{0,1\}$ and let $U \subseteq 1^*$ be the language $\{1^n$: the binary expansion of n is in $L.\}$ Clearly U is undecidable (Why?) Now consider the following family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$. If $1^n \in U$, then C_n consists of $(n-1)$ **AND** gates that compute the conjunction of all the inputs. If $1^n \notin U$, then C_n consists of its input gates and an output gate that is **false**. Thus, for all inputs, $1^n \in U \leftrightarrow C_n$ outputs **true**. □

Circuits and Complexity

Proposition

All languages in **P** *have polynomial sized circuits.*

Proof.

We actually reduced an arbitrary Turing Machine that halts in polynomial time to a variable-free circuit, when we showed that CIRCUIT-VALUE is **P-complete**. □

Theorem

There are undecidable languages which have polynomial circuits.

Proof.

Let $L \subseteq \{0,1\}^*$ be any undecidable language in the alphabet $\{0,1\}$ and let $U \subseteq 1^*$ be the language $\{1^n$: the binary expansion of n is in $L.\}$ Clearly U is undecidable (Why?) Now consider the following family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$. If $1^n \in U$, then C_n consists of $(n-1)$ **AND** gates that compute the conjunction of all the inputs. If $1^n \notin U$, then C_n consists of its input gates and an output gate that is **false**. Thus, for all inputs, $1^n \in U \leftrightarrow C_n$ outputs **true**. □

Circuits and Complexity

Proposition

All languages in **P** *have polynomial sized circuits.*

Proof.

We actually reduced an arbitrary Turing Machine that halts in polynomial time to a variable-free circuit, when we showed that CIRCUIT-VALUE is **P-complete**. □

Theorem

There are undecidable languages which have polynomial circuits.

Proof.

Let $L \subseteq \{0,1\}^*$ be any undecidable language in the alphabet $\{0,1\}$ and let $U \subseteq 1^*$ be the language $\{1^n$: the binary expansion of n is in $L.\}$ Clearly U is undecidable (Why?) Now consider the following family of polynomial circuits $\mathcal{C} = (C_0, C_1, \ldots)$. If $1^n \in U$, then C_n consists of $(n-1)$ **AND** gates that compute the conjunction of all the inputs. If $1^n \notin U$, then C_n consists of its input gates and an output gate that is **false**. Thus, for all inputs, $1^n \in U \leftrightarrow C_n$ outputs **true**. □

Uniform Circuits

Definition

A family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ is said to be uniform if there is a log n-space bounded Turing machine that on input 1^n, outputs C_n.

Definition

A language L has uniformly polynomial circuits if there is a uniform family of polynomial circuits $(C_0, C_1, \ldots)$ which decides L.

Note

Reachability has uniformly polynomial circuits, while the circuit family for the undecidable language is **not** *uniform.*

Theorem

A language L has uniformly polynomial circuits if and only if it is in **P**.

Uniform Circuits

Definition

A family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ is said to be uniform if there is a log n-space bounded Turing machine that on input 1^n, outputs C_n.

Definition

A language L has uniformly polynomial circuits if there is a uniform family of polynomial circuits $(C_0, C_1, \ldots)$ which decides L.

Note

Reachability has uniformly polynomial circuits, while the circuit family for the undecidable language is **not** *uniform.*

Theorem

A language L has uniformly polynomial circuits if and only if it is in **P**.

Uniform Circuits

Definition

A family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ is said to be uniform if there is a log n-space bounded Turing machine that on input 1^n, outputs C_n.

Definition

A language L has uniformly polynomial circuits if there is a uniform family of polynomial circuits $(C_0, C_1, \ldots)$ which decides L.

Note

Reachability has uniformly polynomial circuits, while the circuit family for the undecidable language is **not** *uniform.*

Theorem

A language L has uniformly polynomial circuits if and only if it is in **P**.

Uniform Circuits

Definition

A family of circuits $\mathcal{C} = (C_0, C_1, \ldots)$ is said to be uniform if there is a log n-space bounded Turing machine that on input 1^n, outputs C_n.

Definition

A language L has uniformly polynomial circuits if there is a uniform family of polynomial circuits $(C_0, C_1, \ldots)$ which decides L.

Note

Reachability has uniformly polynomial circuits, while the circuit family for the undecidable language is **not** *uniform.*

Theorem

A language L has uniformly polynomial circuits if and only if it is in **P**.

Uniform circuits (contd.)

Proof.

The fact that if $L \in \mathbf{P}$ then it has uniformly polynomial circuits was established by the reduction of any polynomial time Turing machine to a circuit.

For the converse, let L have a uniformly polynomial circuit. Given x, build $C_{|x|}$ in $\log |x|$ space and hence polynomial time. Evaluate $C_{|x|}$ by setting the inputs so that they spell x! □

Conjecture

NP-complete *problems have no uniformly polynomial circuits.*

Conjecture

NP-complete *problems have no polynomial circuits, uniform or not!*

Uniform circuits (contd.)

Proof.

The fact that if $L \in \mathbf{P}$ then it has uniformly polynomial circuits was established by the reduction of any polynomial time Turing machine to a circuit.
For the converse, let L have a uniformly polynomial circuit. Given x, build $C_{|x|}$ in $\log |x|$ space and hence polynomial time. Evaluate $C_{|x|}$ by setting the inputs so that they spell x! □

Conjecture

NP-complete *problems have no uniformly polynomial circuits.*

Conjecture

NP-complete *problems have no polynomial circuits, uniform or not!*

Uniform circuits (contd.)

Proof.

The fact that if $L \in \mathbf{P}$ then it has uniformly polynomial circuits was established by the reduction of any polynomial time Turing machine to a circuit.
For the converse, let L have a uniformly polynomial circuit. Given x, build $C_{|x|}$ in $\log |x|$ space and hence polynomial time. Evaluate $C_{|x|}$ by setting the inputs so that they spell x! □

Conjecture

NP-complete *problems have no uniformly polynomial circuits.*

Conjecture

NP-complete *problems have no polynomial circuits, uniform or not!*

Uniform circuits (contd.)

Proof.

The fact that if $L \in \mathbf{P}$ then it has uniformly polynomial circuits was established by the reduction of any polynomial time Turing machine to a circuit.
For the converse, let L have a uniformly polynomial circuit. Given x, build $C_{|x|}$ in $\log |x|$ space and hence polynomial time. Evaluate $C_{|x|}$ by setting the inputs so that they spell x! □

Conjecture

NP-complete *problems have no uniformly polynomial circuits.*

Conjecture

NP-complete *problems have no polynomial circuits, uniform or not!*

The Probabilistic Method

Two themes

(i) Every random variable assumes at least one value no less than its expected value and at least one value no more than its expected value.

(ii) If an object chosen randomly from a universe satisfies a property with positive probability, then there must exist at least one object in the universe which satisfies the property.

Theorem

Every CNF formula on m clauses has an assignment that satisfies at least $\frac{m}{2}$ clauses.

The Probabilistic Method

Two themes

(i) Every random variable assumes at least one value no less than its expected value and at least one value no more than its expected value.

(ii) If an object chosen randomly from a universe satisfies a property with positive probability, then there must exist at least one object in the universe which satisfies the property.

Theorem

Every CNF formula on m clauses has an assignment that satisfies at least $\frac{m}{2}$ clauses.

The Probabilistic Method

Two themes

(i) Every random variable assumes at least one value no less than its expected value and at least one value no more than its expected value.

(ii) If an object chosen randomly from a universe satisfies a property with positive probability, then there must exist at least one object in the universe which satisfies the property.

Theorem

Every CNF formula on m clauses has an assignment that satisfies at least $\frac{m}{2}$ *clauses.*

The Chernoff Bound

Theorem

Let $x_1, x_2, \ldots, x_n$ denote n independent $0/1$ Bernoulli variables with $\mathbf{Pr}[x_i = 1] = p$, for each $i = 1, 2, \ldots n$. Let $X = \sum_{i=1}^{n} x_i$. Then for $0 \leq \theta \leq 1$,

$$\mathbf{Pr}[X \geq (1+\theta)np] \leq e^{-\frac{\theta^2}{3}np}$$

Circuits and **BPP**

Theorem

All languages in **BPP** *have polynomial circuits.*

Proof.

Let $L \in$ **BPP** be decided by a non-deterministic Turing machine N that decides by a clear majority and halts in time $p(n)$, for all inputs x, $|x| = n$.
We need to construct C_n for each n, but an explicit construction is unlikely! (Why?)
Consider the sequence of bit strings $\mathcal{A}_n = \{a_1, a_2, \ldots, a_m\}$ with each $a_i \in \{0,1\}^{p(n)}$ and $m = 12 \cdot (n+1)$. Each bit string simulates a computation of N. C_n on input x, $|x| = n$ simulates N with each sequence of choices in $\mathcal{A}_n$ and takes the majority.
Clearly, given A_n, C_n can be constructed with polynomially many gates.
But is there such an $\mathcal{A}_n$? □

Circuits and **BPP**

Theorem

All languages in **BPP** *have polynomial circuits.*

Proof.

Let $L \in$ **BPP** be decided by a non-deterministic Turing machine N that decides by a clear majority and halts in time $p(n)$, for all inputs x, $|x| = n$.
We need to construct C_n for each n, but an explicit construction is unlikely! (Why?)
Consider the sequence of bit strings $\mathcal{A}_n = \{a_1, a_2, \ldots, a_m\}$ with each $a_i \in \{0,1\}^{p(n)}$ and $m = 12 \cdot (n+1)$. Each bit string simulates a computation of N. C_n on input x, $|x| = n$ simulates N with each sequence of choices in $\mathcal{A}_n$ and takes the majority.
Clearly, given A_n, C_n can be constructed with polynomially many gates.
But is there such an $\mathcal{A}_n$? □

Circuits and **BPP**

Theorem

All languages in **BPP** *have polynomial circuits.*

Proof.

Let $L \in$ **BPP** be decided by a non-deterministic Turing machine N that decides by a clear majority and halts in time $p(n)$, for all inputs x, $|x| = n$.
We need to construct C_n for each n, but an explicit construction is unlikely! (Why?)
Consider the sequence of bit strings $\mathcal{A}_n = \{a_1, a_2, \ldots, a_m\}$ with each $a_i \in \{0,1\}^{p(n)}$ and $m = 12 \cdot (n+1)$. Each bit string simulates a computation of N. C_n on input x, $|x| = n$ simulates N with each sequence of choices in $\mathcal{A}_n$ and takes the majority.
Clearly, given A_n, C_n can be constructed with polynomially many gates.
But is there such an $\mathcal{A}_n$? □

Circuits and **BPP**

Theorem

All languages in **BPP** *have polynomial circuits.*

Proof.

Let $L \in$ **BPP** be decided by a non-deterministic Turing machine N that decides by a clear majority and halts in time $p(n)$, for all inputs x, $|x| = n$.
We need to construct C_n for each n, but an explicit construction is unlikely! (Why?)
Consider the sequence of bit strings $\mathcal{A}_n = \{a_1, a_2, \ldots, a_m\}$ with each $a_i \in \{0,1\}^{p(n)}$ and $m = 12 \cdot (n+1)$. Each bit string simulates a computation of N. C_n on input x, $|x| = n$ simulates N with each sequence of choices in $\mathcal{A}_n$ and takes the majority.
Clearly, given A_n, C_n can be constructed with polynomially many gates.
But is there such an $\mathcal{A}_n$? □

Circuits and **BPP**

Theorem

All languages in **BPP** *have polynomial circuits.*

Proof.

Let $L \in$ **BPP** be decided by a non-deterministic Turing machine N that decides by a clear majority and halts in time $p(n)$, for all inputs x, $|x| = n$.
We need to construct C_n for each n, but an explicit construction is unlikely! (Why?)
Consider the sequence of bit strings $\mathcal{A}_n = \{a_1, a_2, \ldots, a_m\}$ with each $a_i \in \{0,1\}^{p(n)}$ and $m = 12 \cdot (n+1)$. Each bit string simulates a computation of N. C_n on input x, $|x| = n$ simulates N with each sequence of choices in $\mathcal{A}_n$ and takes the majority.
Clearly, given A_n, C_n can be constructed with polynomially many gates.
But is there such an $\mathcal{A}_n$? □

Circuits and **BPP**

Theorem

All languages in **BPP** *have polynomial circuits.*

Proof.

Let $L \in$ **BPP** be decided by a non-deterministic Turing machine N that decides by a clear majority and halts in time $p(n)$, for all inputs x, $|x| = n$.
We need to construct C_n for each n, but an explicit construction is unlikely! (Why?)
Consider the sequence of bit strings $\mathcal{A}_n = \{a_1, a_2, \ldots, a_m\}$ with each $a_i \in \{0,1\}^{p(n)}$ and $m = 12 \cdot (n+1)$. Each bit string simulates a computation of N. C_n on input x, $|x| = n$ simulates N with each sequence of choices in $\mathcal{A}_n$ and takes the majority.
Clearly, given A_n, C_n can be constructed with polynomially many gates.
But is there such an $\mathcal{A}_n$? □

Circuits and **BPP**

Theorem

All languages in **BPP** *have polynomial circuits.*

Proof.

Let $L \in$ **BPP** be decided by a non-deterministic Turing machine N that decides by a clear majority and halts in time $p(n)$, for all inputs x, $|x| = n$.
We need to construct C_n for each n, but an explicit construction is unlikely! (Why?)
Consider the sequence of bit strings $\mathcal{A}_n = \{a_1, a_2, \ldots, a_m\}$ with each $a_i \in \{0,1\}^{p(n)}$ and $m = 12 \cdot (n+1)$. Each bit string simulates a computation of N. C_n on input x, $|x| = n$ simulates N with each sequence of choices in $\mathcal{A}_n$ and takes the majority.
Clearly, given A_n, C_n can be constructed with polynomially many gates.
But is there such an $\mathcal{A}_n$? □

Circuits and **BPP**

Theorem

All languages in **BPP** *have polynomial circuits.*

Proof.

Let $L \in$ **BPP** be decided by a non-deterministic Turing machine N that decides by a clear majority and halts in time $p(n)$, for all inputs x, $|x| = n$.
We need to construct C_n for each n, but an explicit construction is unlikely! (Why?)
Consider the sequence of bit strings $\mathcal{A}_n = \{a_1, a_2, \ldots, a_m\}$ with each $a_i \in \{0,1\}^{p(n)}$ and $m = 12 \cdot (n+1)$. Each bit string simulates a computation of N. C_n on input x, $|x| = n$ simulates N with each sequence of choices in $\mathcal{A}_n$ and takes the majority.
Clearly, given A_n, C_n can be constructed with polynomially many gates.
But is there such an $\mathcal{A}_n$? □

Circuits and **BPP** (contd.)

Lemma

For all $n > 0$, there is a set $\mathcal{A}_n$ of $m = 12 \cdot (n+1)$ bit strings such that for all inputs x, with $|x| = n$, fewer than half the choices in $\mathcal{A}_n$ are bad, i.e., lead to either a false positive or a false negative.

Proof.

Consider a sequence $\mathcal{A}_n$ of m bit strings of length p_n selected at random by m independent samplings of $\{0,1\}^{p(n)}$. What is the probability that more than half the choices in $\mathcal{A}_n$ are correct?
For each $x \in \{0,1\}^n$, at most one-quarter of the computations are bad. (Why?) Since the sequences in $\mathcal{A}_n$ were picked randomly and independently, the expected number of bad sequences is at most $\frac{1}{4}m$. By the Chernoff bound, the probability that the number of bad strings is $\frac{1}{2}m$ or more is at most $e^{\frac{-m}{12}} < \frac{1}{2^{n+1}}$.
The above inequality holds regardless of the choice of $x \in \{0,1\}^n$. Thus, the probability that there exists an x with no accepting sequence in $\mathcal{A}_n$ is at most $2^n \frac{1}{2^{n+1}} = \frac{1}{2}$.
With probability at least one-half, the random selection of sequences has the desired property. □

Circuits and **BPP** (contd.)

Lemma

For all $n > 0$, there is a set $\mathcal{A}_n$ of $m = 12 \cdot (n+1)$ bit strings such that for all inputs x, with $|x| = n$, fewer than half the choices in $\mathcal{A}_n$ are bad, i.e., lead to either a false positive or a false negative.

Proof.

Consider a sequence $\mathcal{A}_n$ of m bit strings of length p_n selected at random by m independent samplings of $\{0,1\}^{p(n)}$. What is the probability that more than half the choices in $\mathcal{A}_n$ are correct?

For each $x \in \{0,1\}^n$, at most one-quarter of the computations are bad. (Why?) Since the sequences in $\mathcal{A}_n$ were picked randomly and independently, the expected number of bad sequences is at most $\frac{1}{4}m$. By the Chernoff bound, the probability that the number of bad strings is $\frac{1}{2}m$ or more is at most $e^{\frac{-m}{12}} < \frac{1}{2^{n+1}}$.

The above inequality holds regardless of the choice of $x \in \{0,1\}^n$. Thus, the probability that there exists an x with no accepting sequence in $\mathcal{A}_n$ is at most $2^n \frac{1}{2^{n+1}} = \frac{1}{2}$.

With probability at least one-half, the random selection of sequences has the desired property. □

Circuits and **BPP** (contd.)

Lemma

For all $n > 0$, there is a set $\mathcal{A}_n$ of $m = 12 \cdot (n+1)$ bit strings such that for all inputs x, with $|x| = n$, fewer than half the choices in $\mathcal{A}_n$ are bad, i.e., lead to either a false positive or a false negative.

Proof.

Consider a sequence $\mathcal{A}_n$ of m bit strings of length p_n selected at random by m independent samplings of $\{0,1\}^{p(n)}$. What is the probability that more than half the choices in $\mathcal{A}_n$ are correct?
For each $x \in \{0,1\}^n$, at most one-quarter of the computations are bad. (Why?) Since the sequences in $\mathcal{A}_n$ were picked randomly and independently, the expected number of bad sequences is at most $\frac{1}{4}m$. By the Chernoff bound, the probability that the number of bad strings is $\frac{1}{2}m$ or more is at most $e^{\frac{-m}{12}} < \frac{1}{2^{n+1}}$.
The above inequality holds regardless of the choice of $x \in \{0,1\}^n$. Thus, the probability that there exists an x with no accepting sequence in $\mathcal{A}_n$ is at most $2^n \frac{1}{2^{n+1}} = \frac{1}{2}$.
With probability at least one-half, the random selection of sequences has the desired property. □

Circuits and **BPP** (contd.)

Lemma

For all $n > 0$, there is a set $\mathcal{A}_n$ of $m = 12 \cdot (n+1)$ bit strings such that for all inputs x, with $|x| = n$, fewer than half the choices in $\mathcal{A}_n$ are bad, i.e., lead to either a false positive or a false negative.

Proof.

Consider a sequence $\mathcal{A}_n$ of m bit strings of length p_n selected at random by m independent samplings of $\{0,1\}^{p(n)}$. What is the probability that more than half the choices in $\mathcal{A}_n$ are correct?
For each $x \in \{0,1\}^n$, at most one-quarter of the computations are bad. (Why?) Since the sequences in $\mathcal{A}_n$ were picked randomly and independently, the expected number of bad sequences is at most $\frac{1}{4}m$. By the Chernoff bound, the probability that the number of bad strings is $\frac{1}{2}m$ or more is at most $e^{\frac{-m}{12}} < \frac{1}{2^{n+1}}$.
The above inequality holds regardless of the choice of $x \in \{0,1\}^n$. Thus, the probability that there exists an x with no accepting sequence in $\mathcal{A}_n$ is at most $2^n \frac{1}{2^{n+1}} = \frac{1}{2}$.
With probability at least one-half, the random selection of sequences has the desired property. □

Circuits and **BPP** (contd.)

Lemma

For all $n > 0$, there is a set $\mathcal{A}_n$ of $m = 12 \cdot (n+1)$ bit strings such that for all inputs x, with $|x| = n$, fewer than half the choices in $\mathcal{A}_n$ are bad, i.e., lead to either a false positive or a false negative.

Proof.

Consider a sequence $\mathcal{A}_n$ of m bit strings of length p_n selected at random by m independent samplings of $\{0,1\}^{p(n)}$. What is the probability that more than half the choices in $\mathcal{A}_n$ are correct?
For each $x \in \{0,1\}^n$, at most one-quarter of the computations are bad. (Why?) Since the sequences in $\mathcal{A}_n$ were picked randomly and independently, the expected number of bad sequences is at most $\frac{1}{4}m$. By the Chernoff bound, the probability that the number of bad strings is $\frac{1}{2}m$ or more is at most $e^{\frac{-m}{12}} < \frac{1}{2^{n+1}}$.
The above inequality holds regardless of the choice of $x \in \{0,1\}^n$. Thus, the probability that there exists an x with no accepting sequence in $\mathcal{A}_n$ is at most $2^n \frac{1}{2^{n+1}} = \frac{1}{2}$.
With probability at least one-half, the random selection of sequences has the desired property. □

Circuits and **BPP** (contd.)

Lemma

For all $n > 0$, there is a set $\mathcal{A}_n$ of $m = 12 \cdot (n+1)$ bit strings such that for all inputs x, with $|x| = n$, fewer than half the choices in $\mathcal{A}_n$ are bad, i.e., lead to either a false positive or a false negative.

Proof.

Consider a sequence $\mathcal{A}_n$ of m bit strings of length p_n selected at random by m independent samplings of $\{0,1\}^{p(n)}$. What is the probability that more than half the choices in $\mathcal{A}_n$ are correct?
For each $x \in \{0,1\}^n$, at most one-quarter of the computations are bad. (Why?) Since the sequences in $\mathcal{A}_n$ were picked randomly and independently, the expected number of bad sequences is at most $\frac{1}{4}m$. By the Chernoff bound, the probability that the number of bad strings is $\frac{1}{2}m$ or more is at most $e^{\frac{-m}{12}} < \frac{1}{2^{n+1}}$.
The above inequality holds regardless of the choice of $x \in \{0,1\}^n$. Thus, the probability that there exists an x with no accepting sequence in $\mathcal{A}_n$ is at most $2^n \frac{1}{2^{n+1}} = \frac{1}{2}$.
With probability at least one-half, the random selection of sequences has the desired property. □

Circuits and **BPP** (contd.)

Lemma

For all $n > 0$, there is a set $\mathcal{A}_n$ of $m = 12 \cdot (n+1)$ bit strings such that for all inputs x, with $|x| = n$, fewer than half the choices in $\mathcal{A}_n$ are bad, i.e., lead to either a false positive or a false negative.

Proof.

Consider a sequence $\mathcal{A}_n$ of m bit strings of length p_n selected at random by m independent samplings of $\{0,1\}^{p(n)}$. What is the probability that more than half the choices in $\mathcal{A}_n$ are correct?
For each $x \in \{0,1\}^n$, at most one-quarter of the computations are bad. (Why?) Since the sequences in $\mathcal{A}_n$ were picked randomly and independently, the expected number of bad sequences is at most $\frac{1}{4}m$. By the Chernoff bound, the probability that the number of bad strings is $\frac{1}{2}m$ or more is at most $e^{\frac{-m}{12}} < \frac{1}{2^{n+1}}$.
The above inequality holds regardless of the choice of $x \in \{0,1\}^n$. Thus, the probability that there exists an x with no accepting sequence in $\mathcal{A}_n$ is at most $2^n \frac{1}{2^{n+1}} = \frac{1}{2}$.
With probability at least one-half, the random selection of sequences has the desired property. □

Circuits and **BPP** (contd.)

Lemma

For all $n > 0$, there is a set $\mathcal{A}_n$ of $m = 12 \cdot (n+1)$ bit strings such that for all inputs x, with $|x| = n$, fewer than half the choices in $\mathcal{A}_n$ are bad, i.e., lead to either a false positive or a false negative.

Proof.

Consider a sequence $\mathcal{A}_n$ of m bit strings of length p_n selected at random by m independent samplings of $\{0,1\}^{p(n)}$. What is the probability that more than half the choices in $\mathcal{A}_n$ are correct?
For each $x \in \{0,1\}^n$, at most one-quarter of the computations are bad. (Why?) Since the sequences in $\mathcal{A}_n$ were picked randomly and independently, the expected number of bad sequences is at most $\frac{1}{4}m$. By the Chernoff bound, the probability that the number of bad strings is $\frac{1}{2}m$ or more is at most $e^{\frac{-m}{12}} < \frac{1}{2^{n+1}}$.
The above inequality holds regardless of the choice of $x \in \{0,1\}^n$. Thus, the probability that there exists an x with no accepting sequence in $\mathcal{A}_n$ is at most $2^n \frac{1}{2^{n+1}} = \frac{1}{2}$.
With probability at least one-half, the random selection of sequences has the desired property. □

Completing the Main theorem

Proof of Main theorem (contd.)

Given a suitable $\mathcal{A}_n$, construct C_n with $O(n^2(p(n))^2)$ gates so that it simulates N with each of these sequences and then takes the majority of these outcomes. Based on the choice of $\mathcal{A}_n$, C_n outputs **true** if and only if the input is in $L \cap \{0,1\}^n$, i.e., L has a polynomial family of circuits.

Completing the Main theorem

Proof of Main theorem (contd.)

Given a suitable $\mathcal{A}_n$, construct C_n with $O(n^2(p(n))^2)$ gates so that it simulates N with each of these sequences and then takes the majority of these outcomes. Based on the choice of $\mathcal{A}_n$, C_n outputs **true** if and only if the input is in $L \cap \{0,1\}^n$, i.e., L has a polynomial family of circuits.