Lecture 14: Boolean Circuits I

Arijit Bishnu

17.04.2010

Outline

Outline

Boolean Circuits

- It is a generalization of Boolean formulas and a simplified model of the silicon chips.

Boolean Circuits

- It is a generalization of Boolean formulas and a simplified model of the silicon chips.
- A boolean circuit is a DAG formed of OR, AND and NOT gates. How can it model the silicon chips which are not cyclic and use cycles to implement memory?

Boolean Circuits

- It is a generalization of Boolean formulas and a simplified model of the silicon chips.
- A boolean circuit is a DAG formed of OR, AND and NOT gates. How can it model the silicon chips which are not cyclic and use cycles to implement memory?
- Any computation that runs on a silicon chip using C gates and finishes in time T can be performed by a Boolean circuit of size $O(C \cdot T)$.

Boolean Circuits

- It is a generalization of Boolean formulas and a simplified model of the silicon chips.
- A boolean circuit is a DAG formed of OR, AND and NOT gates. How can it model the silicon chips which are not cyclic and use cycles to implement memory?
- Any computation that runs on a silicon chip using C gates and finishes in time T can be performed by a Boolean circuit of size $O(C \cdot T)$.
- Boolean circuits is a natural model for non-uniform computation as opposed to Turing machines.

Boolean Circuits

- It is a generalization of Boolean formulas and a simplified model of the silicon chips.
- A boolean circuit is a DAG formed of OR, AND and NOT gates. How can it model the silicon chips which are not cyclic and use cycles to implement memory?
- Any computation that runs on a silicon chip using C gates and finishes in time T can be performed by a Boolean circuit of size $O(C \cdot T)$.
- Boolean circuits is a natural model for non-uniform computation as opposed to Turing machines.
- Mathematically simpler than TMs.

Boolean Circuits

- It is a generalization of Boolean formulas and a simplified model of the silicon chips.
- A boolean circuit is a DAG formed of OR, AND and NOT gates. How can it model the silicon chips which are not cyclic and use cycles to implement memory?
- Any computation that runs on a silicon chip using C gates and finishes in time T can be performed by a Boolean circuit of size $O(C \cdot T)$.
- Boolean circuits is a natural model for non-uniform computation as opposed to Turing machines.
- Mathematically simpler than TMs.
- Proving lower bounds might be easier.

Outline

Boolean Circuits

Definition (Boolean Circuits)

For every $n \in \mathbb{N}$, an n-input, single-output Boolean circuit is a DAG with n sources and 1 sink. All non-source vertices are called gates and are labeled with one of $\wedge$ (AND), $\vee$ (OR) and $\neg$ (NOT). The AND and OR gates have fan-in equal to 2 and the NOT gate has fan-in equal to 1. The size of the circuit C, denoted by $|C|$, is the number of vertices in it.

If C is a boolean circuit, and $x \in \{0,1\}^n$ is some input, then the output of C on x is denoted by $C(x)$ and is defined in the usual way.

Circuit Families and Language Recognition

Definition: Circuit Families and Language Recognition

Let $T : \mathbb{N} \to \mathbb{N}$ be a function. A $T(n)$-size circuit family is a sequence $\{C_n\}_{n \in \mathbb{N}}$ of Boolean circuits, where C_n has n-inputs and a single output, and its size $|C_n| \leq T(n)$ for every n.

We say that a language L is in SIZE($T(n)$) if $\exists$ a $T(n)$-size circuit family $\{C_n\}_{n \in \mathbb{N}}$ such that for every $x \in \{0,1\}^n$,
$x \in L \iff C_n(x) = 1$.

Examples

Example

What about the language 2COLORABLE $= \{< G > \mid$ Graph G is 2-colorable$\}$?

Examples

Example

What about the language 2COLORABLE $= \{< G > \mid$ Graph G is 2-colorable$\}$?

Example

What about the language INDSET $= \{< G, k > \mid$ Graph G has an independent set of size $\geq k\}$?

Examples

Example

What about the language 2COLORABLE $= \{< G > \mid$ Graph G is 2-colorable$\}$?

Example

What about the language INDSET $= \{< G, k > \mid$ Graph G has an independent set of size $\geq k\}$?

Example

What about the language $= \{1^n \mid n \in \mathbb{Z}\}$?

Examples

Example

What about the language 2COLORABLE $= \{< G > \mid$ Graph G is 2-colorable$\}$?

Example

What about the language INDSET $= \{< G, k > \mid$ Graph G has an independent set of size $\geq k\}$?

Example

What about the language $= \{1^n \mid n \in \mathbb{Z}\}$?

Example

What about the language $\{< m, n, m+n > \mid m, n \in \mathbb{Z}\}$?

Definition of a New Class

- We know by now that any Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ can be computed by a Boolean circuit of size $n2^n$.

Definition of a New Class

- We know by now that any Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ can be computed by a Boolean circuit of size $n2^n$.
- So, "large-sized circuits" are not of much interest.

Definition of a New Class

- We know by now that any Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ can be computed by a Boolean circuit of size $n2^n$.
- So, "large-sized circuits" are not of much interest.
- Interesting complexity classes arise when we consider "small" circuits.

Definition of a New Class

- We know by now that any Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be computed by a Boolean circuit of size $n2^n$.
- So, "large-sized circuits" are not of much interest.
- Interesting complexity classes arise when we consider "small" circuits.

Definition: The Class $P_{/poly}$

$P_{/poly}$ is the class of languages that are decidable by polynomial-sized circuit families. That is, $P_{/poly} = \bigcup_c \mathrm{SIZE}(n^c)$.

$P_{/poly}$ and P

- Does SAT $\in P_{/poly}$?

$P_{/poly}$ and P

- Does SAT $\in P_{/poly}$?
- We believe SAT $\notin P_{/poly}$. Then, how are P and $P_{/poly}$ related?

$P_{/poly}$ and P

- Does SAT $\in P_{/poly}$?
- We believe SAT $\notin P_{/poly}$. Then, how are P and $P_{/poly}$ related?

$P_{/poly}$ and P

- Does SAT $\in P_{/poly}$?
- We believe SAT $\notin P_{/poly}$. Then, how are P and $P_{/poly}$ related?

P and $P_{/poly}$

$P \subseteq P_{/poly}$

$P_{/poly}$ and P

- Does SAT $\in P_{/poly}$?
- We believe SAT $\notin P_{/poly}$. Then, how are P and $P_{/poly}$ related?

P and $P_{/poly}$

$P \subseteq P_{/poly}$

Proof

$P_{/poly}$ and P

- Does SAT $\in P_{/poly}$?
- We believe SAT $\notin P_{/poly}$. Then, how are P and $P_{/poly}$ related?

P and $P_{/poly}$

$P \subseteq P_{/poly}$

Proof

- The proof basically shows that for every oblivious TM M (whose head movement is independent of the input) running in $T(n)$-time, there exists an $O(T(n))$-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that $C_n(x) = M(x)$ for every $x \in \{0,1\}^n$.

$P_{/poly}$ and P

- Does SAT $\in P_{/poly}$?
- We believe SAT $\notin P_{/poly}$. Then, how are P and $P_{/poly}$ related?

P and $P_{/poly}$

$P \subseteq P_{/poly}$

Proof

- The proof basically shows that for every oblivious TM M (whose head movement is independent of the input) running in $T(n)$-time, there exists an $O(T(n))$-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that $C_n(x) = M(x)$ for every $x \in \{0,1\}^n$.
- We already know that an oblivious TM can simulate every other TM using an amount of speed-up.

$P_{/poly}$ and P

- Does SAT $\in P_{/poly}$?
- We believe SAT $\notin P_{/poly}$. Then, how are P and $P_{/poly}$ related?

P and $P_{/poly}$

$P \subseteq P_{/poly}$

Proof

- The proof basically shows that for every oblivious TM M (whose head movement is independent of the input) running in $T(n)$-time, there exists an $O(T(n))$-sized circuit family $\{C_n\}_{n\in\mathbb{N}}$ such that $C_n(x) = M(x)$ for every $x \in \{0,1\}^n$.
- We already know that an oblivious TM can simulate every other TM using an amount of speed-up.
- This proof also gives the poly-sized circuit in poly-time.

The Proof

- Take snapshots $z_1, \ldots, z_{T(n)}$ of the oblivious k-tape TM. $z_1, \ldots, z_{T(n)}$ depends only on machine's states and symbols read by all heads.

The Proof

- Take snapshots $z_1, \ldots, z_{T(n)}$ of the oblivious k-tape TM. $z_1, \ldots, z_{T(n)}$ depends only on machine's states and symbols read by all heads.
- We encode each snapshot z_i by a $O(1)$-sized binary string as z_i is independent of input.

The Proof

- Take snapshots $z_1, \ldots, z_{T(n)}$ of the oblivious k-tape TM. $z_1, \ldots, z_{T(n)}$ depends only on machine's states and symbols read by all heads.
- We encode each snapshot z_i by a $O(1)$-sized binary string as z_i is independent of input.
- We can compute z_i by knowing z_{i-1} and the snapshots $z_{i_1}, z_{i_2}, \ldots, z_{i_k}$ where z_{i_j} denotes the last step that M's j-th head was in the same position as it is in the i-th step.

The Proof

- Take snapshots $z_1, \ldots, z_{T(n)}$ of the oblivious k-tape TM. $z_1, \ldots, z_{T(n)}$ depends only on machine's states and symbols read by all heads.
- We encode each snapshot z_i by a $O(1)$-sized binary string as z_i is independent of input.
- We can compute z_i by knowing z_{i-1} and the snapshots $z_{i_1}, z_{i_2}, \ldots, z_{i_k}$ where z_{i_j} denotes the last step that M's j-th head was in the same position as it is in the i-th step.
- We have $O(1)$ strings each of size $O(1)$.

The Proof

- Take snapshots $z_1, \ldots, z_{T(n)}$ of the oblivious k-tape TM. $z_1, \ldots, z_{T(n)}$ depends only on machine's states and symbols read by all heads.
- We encode each snapshot z_i by a $O(1)$-sized binary string as z_i is independent of input.
- We can compute z_i by knowing z_{i-1} and the snapshots $z_{i_1}, z_{i_2}, \ldots, z_{i_k}$ where z_{i_j} denotes the last step that M's j-th head was in the same position as it is in the i-th step.
- We have $O(1)$ strings each of size $O(1)$.
- So, we can compute z_i from previous snapshots using a constant-sized circuit.

The Proof

- Take snapshots $z_1, \ldots, z_{T(n)}$ of the oblivious k-tape TM. $z_1, \ldots, z_{T(n)}$ depends only on machine's states and symbols read by all heads.
- We encode each snapshot z_i by a $O(1)$-sized binary string as z_i is independent of input.
- We can compute z_i by knowing z_{i-1} and the snapshots $z_{i_1}, z_{i_2}, \ldots, z_{i_k}$ where z_{i_j} denotes the last step that M's j-th head was in the same position as it is in the i-th step.
- We have $O(1)$ strings each of size $O(1)$.
- So, we can compute z_i from previous snapshots using a constant-sized circuit.
- Compose all the circuits to get a $O(T(n))$-sized circuit.

A Remark about the Proof

Remark

The circuit of the above Theorem is not only of polynomial size but can also be computed in polynomial time. Moreover, space requirement is only logarithmic as head positions need to be stored.

$\mathsf{P}_{/\mathsf{poly}}$ and P

Claim

Let $L \subseteq \{0,1\}^*$ be a unary language, i.e. $L \subseteq \{1^n \mid n \in \mathbb{N}\}$. Then, $L \in \mathsf{P}_{/\mathsf{poly}}$.

$\mathsf{P}_{/\mathrm{poly}}$ and P

Claim

Let $L \subseteq \{0,1\}^*$ be a unary language, i.e. $L \subseteq \{1^n \mid n \in \mathbb{N}\}$. Then, $L \in \mathsf{P}_{/\mathrm{poly}}$.

Proof

If $1^n \in L$, then the circuit for inputs of size n is a tree of AND gates; otherwise, it is the circuit that always outputs 0.

$P_{/poly}$ and P

Claim

Let $L \subseteq \{0,1\}^*$ be a unary language, i.e. $L \subseteq \{1^n \mid n \in \mathbb{N}\}$. Then, $L \in P_{/poly}$.

Proof

If $1^n \in L$, then the circuit for inputs of size n is a tree of AND gates; otherwise, it is the circuit that always outputs 0.

Unary version of the Halting Problem

UHALT $= \{1^n \mid n'$s binary expansion encodes a pair $< M, x >$ s.t. M halts on input $x\}$.

$P_{/poly}$ and P

Claim

Let $L \subseteq \{0,1\}^*$ be a unary language, i.e. $L \subseteq \{1^n \mid n \in \mathbb{N}\}$. Then, $L \in P_{/poly}$.

Proof

If $1^n \in L$, then the circuit for inputs of size n is a tree of AND gates; otherwise, it is the circuit that always outputs 0.

Unary version of the Halting Problem

UHALT $= \{1^n \mid n'$s binary expansion encodes a pair $< M, x >$ s.t. M halts on input $x\}$.

- The above language is undecidable and hence is not in P.

$P_{/poly}$ and P

Claim

Let $L \subseteq \{0,1\}^*$ be a unary language, i.e. $L \subseteq \{1^n \mid n \in \mathbb{N}\}$. Then, $L \in P_{/poly}$.

Proof

If $1^n \in L$, then the circuit for inputs of size n is a tree of AND gates; otherwise, it is the circuit that always outputs 0.

Unary version of the Halting Problem

UHALT $= \{1^n \mid n'$s binary expansion encodes a pair $< M, x >$ s.t. M halts on input $x\}$.

- The above language is undecidable and hence is not in P.
- But, UHALT being a unary language belongs to $P_{/poly}$.

$P_{/poly}$ and P

Claim

Let $L \subseteq \{0,1\}^*$ be a unary language, i.e. $L \subseteq \{1^n \mid n \in \mathbb{N}\}$. Then, $L \in P_{/poly}$.

Proof

If $1^n \in L$, then the circuit for inputs of size n is a tree of AND gates; otherwise, it is the circuit that always outputs 0.

Unary version of the Halting Problem

UHALT $= \{1^n \mid n'$s binary expansion encodes a pair $< M, x >$ s.t. M halts on input $x\}$.

- The above language is undecidable and hence is not in P.
- But, UHALT being a unary language belongs to $P_{/poly}$.
- So, $P \subset P_{/poly}$, i.e. the inclusion is proper.

$P_{/poly}$ is Awkward!

- $P_{/poly}$ contains even undecidable languages. What is it in the definition of the class that allows even undecidable languages?

$P_{/poly}$ is Awkward!

- $P_{/poly}$ contains even undecidable languages. What is it in the definition of the class that allows even undecidable languages?
- A language $L \in P_{/poly}$ if $\exists$ a circuit family; we do not even need to construct it!!

Outline

Uniformly Generated Circuits

- We restrict our attention to circuits that can be built by an efficient TM.

Uniformly Generated Circuits

- We restrict our attention to circuits that can be built by an efficient TM.

Definition: P-uniform Circuit Families

A circuit family $\{C_n\}$ is P-uniform if there is a poly-time TM that on input 1^n outputs the description of the circuit C_n.

Uniformly Generated Circuits

- We restrict our attention to circuits that can be built by an efficient TM.

Definition: P-uniform Circuit Families

A circuit family $\{C_n\}$ is P-uniform if there is a poly-time TM that on input 1^n outputs the description of the circuit C_n.

- What happens when we restrict circuits to be P-uniform.

Uniformly Generated Circuits

- We restrict our attention to circuits that can be built by an efficient TM.

Definition: P-uniform Circuit Families

A circuit family $\{C_n\}$ is P-uniform if there is a poly-time TM that on input 1^n outputs the description of the circuit C_n.

- What happens when we restrict circuits to be P-uniform.

Theorem

A language L is computable by a P-uniform circuit family iff $L \in \mathrm{P}$.

The Proof

The Proof of the Theorem (if part)

The Proof

The Proof of the Theorem (if part)

- Assume L is computable by a circuit family $\{C_n\}$ that is generated by a poly-time TM M.

The Proof

The Proof of the Theorem (if part)

- Assume L is computable by a circuit family $\{C_n\}$ that is generated by a poly-time TM M.
- We now design a poly-time TM M' that works as follows:

The Proof

The Proof of the Theorem (if part)

- Assume L is computable by a circuit family $\{C_n\}$ that is generated by a poly-time TM M.
- We now design a poly-time TM M' that works as follows:
  - On input x, M' runs $M(1^{|x|})$ to obtain the circuit $C_{|x|}$.

The Proof

The Proof of the Theorem (if part)

- Assume L is computable by a circuit family $\{C_n\}$ that is generated by a poly-time TM M.
- We now design a poly-time TM M' that works as follows:
  - On input x, M' runs $M(1^{|x|})$ to obtain the circuit $C_{|x|}$.
  - M' evaluates $C_{|x|}$ on x.

The Proof

The Proof of the Theorem (if part)

- Assume L is computable by a circuit family $\{C_n\}$ that is generated by a poly-time TM M.
- We now design a poly-time TM M' that works as follows:
  - On input x, M' runs $M(1^{|x|})$ to obtain the circuit $C_{|x|}$.
  - M' evaluates $C_{|x|}$ on x.

The Proof of the Theorem (only if part)

The Proof

The Proof of the Theorem (if part)

- Assume L is computable by a circuit family $\{C_n\}$ that is generated by a poly-time TM M.
- We now design a poly-time TM M' that works as follows:
  - On input x, M' runs $M(1^{|x|})$ to obtain the circuit $C_{|x|}$.
  - M' evaluates $C_{|x|}$ on x.

The Proof of the Theorem (only if part)

- The proof of $P \subseteq P_{/poly}$ can be used as the proof is constructive and generates a circuit family that is P-uniform.

Logspace-uniform families

Definition: Logspace-uniform Circuit Families

A circuit family $\{C_n\}$ is *logspace-uniform* if there is an implicitly logspace computable function mapping 1^n to the description of the circuit C_n.

Logspace-uniform families

Definition: Logspace-uniform Circuit Families

A circuit family $\{C_n\}$ is *logspace-uniform* if there is an implicitly logspace computable function mapping 1^n to the description of the circuit C_n.

Theorem

A language L has logspace-uniform circuits of polynomial size iff $L \in \mathrm{P}$.

Outline

Turing Machines that take Advice

Definition

We say that a language $L \in P_{/poly}$ if $\exists$ a sequence of advice $\{a_1, a_2, \ldots\} \subseteq \{0,1\}^*$, a polynomial $p(n)$ and a language $L' \in P$, s.t.

Turing Machines that take Advice

Definition

We say that a language $L \in P_{/poly}$ if $\exists$ a sequence of advice $\{a_1, a_2, \ldots\} \subseteq \{0,1\}^*$, a polynomial $p(n)$ and a language $L' \in P$, s.t.

- $\forall n \in \mathbb{N}$, $|a_n| \leq p(n)$

Turing Machines that take Advice

Definition

We say that a language $L \in P_{/poly}$ if $\exists$ a sequence of advice $\{a_1, a_2, \ldots\} \subseteq \{0,1\}^*$, a polynomial $p(n)$ and a language $L' \in P$, s.t.

- $\forall n \in \mathbb{N}$, $|a_n| \leq p(n)$
- $\forall x \in \{0,1\}^*$ $x \in L \iff < x, a_{|x|} > \in L'$

Turing Machines that take Advice

Definition

We say that a language $L \in P_{/poly}$ if $\exists$ a sequence of advice $\{a_1, a_2, \ldots\} \subseteq \{0,1\}^*$, a polynomial $p(n)$ and a language $L' \in P$, s.t.

- $\forall n \in \mathbb{N}$, $|a_n| \leq p(n)$
- $\forall x \in \{0,1\}^*$ $x \in L \iff < x, a_{|x|} > \in L'$

Example

Every unary language can be decided by a poly-time TM with 1 bit of advice. The advice string for inputs of length n is the single bit indicating whether or not 1^n is in the language.

Turing Machines that take Advice

Theorem

$L \in P_{/poly}$ iff L has polynomial size circuits.

Turing Machines that take Advice

Theorem

$L \in P_{/poly}$ iff L has polynomial size circuits.

Proof (if part)

Turing Machines that take Advice

Theorem

$L \in P_{/poly}$ iff L has polynomial size circuits.

Proof (if part)

- Suppose $L \in P_{/poly}$. We follow the definition.

Turing Machines that take Advice

Theorem

$L \in P_{/poly}$ iff L has polynomial size circuits.

Proof (if part)

- Suppose $L \in P_{/poly}$. We follow the definition.
- There exists a sequence of advice $\{a_1, a_2, \ldots\}$ and a language $L' \in P$. We know from earlier theorem that L' has polynomial size circuits, say $\{C_1, C_2, \ldots\}$.

Turing Machines that take Advice

Theorem

$L \in P_{/poly}$ iff L has polynomial size circuits.

Proof (if part)

- Suppose $L \in P_{/poly}$. We follow the definition.
- There exists a sequence of advice $\{a_1, a_2, \ldots\}$ and a language $L' \in P$. We know from earlier theorem that L' has polynomial size circuits, say $\{C_1, C_2, \ldots\}$.
- Obtain the circuits $\{C'_1, C'_2, \ldots\}$ as follows:

Turing Machines that take Advice

Theorem

$L \in P_{/poly}$ iff L has polynomial size circuits.

Proof (if part)

- Suppose $L \in P_{/poly}$. We follow the definition.
- There exists a sequence of advice $\{a_1, a_2, \ldots\}$ and a language $L' \in P$. We know from earlier theorem that L' has polynomial size circuits, say $\{C_1, C_2, \ldots\}$.
- Obtain the circuits $\{C'_1, C'_2, \ldots\}$ as follows:
  - C'_i is obtained from $C_{i+|a_i|}$ by presetting the advice a_i.

Turing Machines that take Advice

Proof (only if part)

Turing Machines that take Advice

Proof (only if part)

- We encode the polynomial size circuits for L as advice.

Turing Machines that take Advice

Proof (only if part)

- We encode the polynomial size circuits for L as advice.
- A poly-size circuit can be evaluated on any input efficiently. So, this encoding constitutes a valid advice sequence.