UMB CS 622

NP-Completeness

Monday, November 15, 2021
Announcements

• HW8 due Wed 11:59pm

• Good HW discussions on Piazza
Last Time: Verifiers, Formally

\[ \text{PATH} = \{ \langle G, s, t \rangle \mid G \text{ is a directed graph that has a directed path from } s \text{ to } t \} \]

A verifier for a language \( A \) is an algorithm \( V \), where

\[ A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c \} \]

We measure the time of a verifier only in terms of the length of \( w \), so a polynomial time verifier runs in polynomial time in the length of \( w \). A language \( A \) is polynomially verifiable if it has a polynomial time verifier.

- Cert \( c \) has length at most \( n^k \), where \( n = \text{length of } w \)
Last Time: The class **NP**

**DEFINITION**

NP is the class of languages that have polynomial time verifiers.

**THEOREM**

A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine.
Last Time: **NP** Problems

- **CLIQUE** = \{⟨G, k⟩ | G is an undirected graph with a k-clique\}
  - A clique is a subgraph where every two nodes are connected
  - A k-clique contains k nodes

- **SUBSET-SUM** = \{⟨S, t⟩ | S = \{x₁, …, xₖ\}, and for some \{y₁, …, y_l\} ⊆ \{x₁, …, xₖ\}, we have ∑yᵢ = t\}
  - Some subset of a set of numbers S must sum to a total t
  - e.g., \{4, 11, 16, 21, 27\}, 25 \in SUBSET-SUM
Theorem: \( \text{SUBSET-SUM} \) is in NP

\[
\text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \Sigma y_i = t \}
\]

**Proof Idea** The subset is the certificate.

To prove a lang is in NP, create either:
- Deterministic poly time verifier
- Nondeterministic poly time decider

**Proof** The following is a verifier \( V \) for SUBSET-SUM.

\( V = \) "On input \( \langle S, t \rangle, c \):

1. Test whether \( c \) is a collection of numbers that sum to \( t \).
2. Test whether \( S \) contains all the numbers in \( c \).
3. If both pass, accept; otherwise, reject."

Does this run in poly time?
Proof 2: \textit{SUBSET-SUM} is in NP

\[ \text{SUBSET-SUM} = \{ \langle S, t \rangle \mid S = \{x_1, \ldots, x_k\}, \text{ and for some } \{y_1, \ldots, y_l\} \subseteq \{x_1, \ldots, x_k\}, \text{ we have } \sum y_i = t \} \]

To prove a lang is in \textbf{NP}, create either:
- Deterministic poly time verifier
- Nondeterministic poly time decider

\textbf{ALTERNATIVE PROOF} We can also prove this theorem by giving a nondeterministic polynomial time Turing machine for \textit{SUBSET-SUM} as follows.

\[ N = \text{“On input } \langle S, t \rangle:\]

1. Nondeterministically select a subset \(c\) of the numbers in \(S\).
2. Test whether \(c\) is a collection of numbers that sum to \(t\).
3. If the test passes, \textit{accept}; otherwise, \textit{reject}.

Does this run in poly time? Nondeterministically runs the verifier many times in parallel
Last Time: NP vs P

**P** The class of languages that have a deterministic poly time decider

I.e., the class of languages that can be solved “quickly”
- We want search problems to be in here ... but they often are not

**NP** The class of languages that have a deterministic poly time verifier

Also, the class of languages that have a nondeterministic poly time decider

I.e., the class of language that can be verified “quickly”
- Search problems, even those not in P, are often in here
One of the Greatest unsolved

**HW Question: Does P = NP?**

Proving $P \neq NP$ is hard: how do you prove that an algorithm won’t ever have a poly time solution? (in general, it’s hard to prove that something doesn’t exist)
Not Much Progress on whether $P = NP$?

The Status of the P Versus NP Problem

By Lance Fortnow
Communications of the ACM, September 2009, Vol. 52 No. 9, Pages 78-86
10.1145/1592164.1592166

• One important concept:
  • NP-Completeness
**NP-Completeness**

**DEFINITION**

A language $B$ is $NP$-complete if it satisfies two conditions:

1. $B$ is in $NP$, and
2. every $A$ in $NP$ is polynomial time reducible to $B$.

**THEOREM**

If $B$ is NP-complete and $B \in P$, then $P = NP$.

• How does this help the $P = NP$ problem?
Flashback: Mapping Reducibility

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$  

The function $f$ is called the **reduction** from $A$ to $B$.

**IMPORTANT:** “if and only if” ...

To show mapping reducibility:
1. create **computable fn**
2. and then show **forward direction**
3. and **reverse direction** (or contrapositive of forward direction)

$A_{TM} = \{ \langle M, w \rangle | M$ is a TM and $M$ accepts $w \}$  

$HALT_{TM} = \{ \langle M, w \rangle | M$ is a TM and $M$ halts on input $w \}$

... means $\overline{A} \leq_m \overline{B}$

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.
Polynomial Time Mapping Reducibility

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$  

The function $f$ is called the **reduction** from $A$ to $B$.

Language $A$ is **polynomial time mapping reducible**, or simply **polynomial time reducible**, to language $B$, written $A \leq_P B$, if a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$ exists, where for every $w$,

$$w \in A \iff f(w) \in B.$$  

The function $f$ is called the **polynomial time reduction** of $A$ to $B$.

A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine $M$, on every input $w$, halts with just $f(w)$ on its tape.
Flashback: If $A \leq_m B$ and $B$ is decidable, then $A$ is decidable.

**Proof**

We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

$N = \text{“On input } w:\$

1. Compute $f(w)$.
2. Run $M$ on input $f(w)$ and output whatever $M$ outputs.”

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every $w$,  

$$w \in A \iff f(w) \in B.$$  

The function $f$ is called the **reduction** from $A$ to $B$. 

This proof only works because of the if-and-only-if requirement.
Thm: If $A \leq_m^P B$ and $B \in \mathbb{P}$ is decidable, then $A \in \mathbb{P}$ is decidable.

**Proof** We let $M$ be the decider for $B$ and $f$ be the reduction from $A$ to $B$. We describe a decider $N$ for $A$ as follows.

$N = \text{“On input } w:\ $

1. Compute $f(w)$.
2. Run $M$ on input $f(w)$ and output whatever $M$ outputs.”

Language $A$ is **mapping reducible** to language $B$, written $A \leq_m B$, if there is a computable function $f : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

$$w \in A \iff f(w) \in B.$$ 

The function $f$ is called the **reduction** from $A$ to $B$. 

126
Thm: If \( A \leq_m B \) and \( B \in \mathbb{P} \) is decidable, then \( A \in \mathbb{P} \).

Proof: We let \( M \) be the decider for \( B \) and \( f \) be the reduction from \( A \) to \( B \). We describe a decider \( N \) for \( A \) as follows.

\[
N = "\text{On input } w:\n1. \text{ Compute } f(w).\n2. \text{ Run } M \text{ on input } f(w) \text{ and output whatever } M \text{ outputs.}"
\]

Language \( A \) is mapping reducible to language \( B \), written \( A \leq_m B \), if there is a computable function \( f : \Sigma^* \rightarrow \Sigma^* \), where for every \( w \),

\[
w \in A \iff f(w) \in B.
\]

The function \( f \) is called the reduction from \( A \) to \( B \).
Theorem: 3SAT is polynomial time reducible to CLIQUE.
**Last Class:** CLIQUE is in NP

\[ \text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique} \} \]

**Proof Idea**

The clique is the certificate.

**Proof**

The following is a verifier \( V \) for CLIQUE.

\[ V = \text{"On input } \langle \langle G, k \rangle, c \rangle : \]

1. Test whether \( c \) is a subgraph with \( k \) nodes in \( G \).
2. Test whether \( G \) contains all edges connecting nodes in \( c \).
3. If both pass, accept; otherwise, reject."
Theorem: $3SAT$ is polynomial time reducible to $CLIQUE$. 

??
# Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
</tbody>
</table>
## Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean _____</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
</tbody>
</table>
Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ((\land, \lor, \text{and} \neg))</td>
</tr>
</tbody>
</table>
## Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean _____</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value</strong></td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td><strong>Variable</strong></td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td><strong>Operation</strong></td>
<td>Combines Boolean <strong>variables</strong></td>
<td>AND, OR, NOT ((\land, \lor, \text{ and } \neg))</td>
</tr>
<tr>
<td><strong>Formula (\phi)</strong></td>
<td>Combines <strong>vars</strong> and <strong>operations</strong></td>
<td>((\overline{x} \land y) \lor (x \land \overline{z}))</td>
</tr>
</tbody>
</table>
Boolean Satisfiability

- A Boolean formula is **satisfiable** if...

- ...there is some assignment of TRUE or FALSE (1 or 0) to its variables that makes the entire formula TRUE

- Is \((\overline{x} \land y) \lor (x \land \overline{z})\) satisfiable?
  - Yes
  - \(x = \text{FALSE,}\)
    - \(y = \text{TRUE,}\)
    - \(z = \text{FALSE}\)
The Boolean Satisfiability Problem

**Theorem:** $SAT$ is in $NP$:

- Let $n$ = the number of variables in the formula

<table>
<thead>
<tr>
<th>Verifier:</th>
</tr>
</thead>
<tbody>
<tr>
<td>On input $&lt;\phi, c&gt;$, where $c$ is a possible assignment of variables in $\phi$ to values:</td>
</tr>
<tr>
<td>• Accept if $c$ satisfies $\phi$</td>
</tr>
</tbody>
</table>

**Running Time:** $O(n)$

<table>
<thead>
<tr>
<th>Non-deterministic Decider:</th>
</tr>
</thead>
<tbody>
<tr>
<td>On input $&lt;\phi&gt;$, where $\phi$ is a boolean formula:</td>
</tr>
<tr>
<td>• Non-deterministically try all possible assignments in parallel</td>
</tr>
<tr>
<td>• Accept if any satisfy $\phi$</td>
</tr>
</tbody>
</table>

**Running Time:** Checking each assignment takes time $O(n)$

$SAT = \{ <\phi> | \phi \text{ is a satisfiable Boolean formula} \}$
Theorem: 3SAT is polynomial time reducible to CLIQUE.
More Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT (∧, ∨, and ¬)</td>
</tr>
<tr>
<td>Formula $\phi$</td>
<td>Combines vars and operations</td>
<td>$(\overline{x} \land y) \lor (x \land \overline{z})$</td>
</tr>
</tbody>
</table>
More Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ((\land, \lor), and (\neg))</td>
</tr>
<tr>
<td>Formula (\phi)</td>
<td>Combines vars and operations</td>
<td>((\overline{x} \land y) \lor (x \land \overline{z}))</td>
</tr>
<tr>
<td>Literal</td>
<td>A var or a negated var</td>
<td>(x, \overline{x})</td>
</tr>
</tbody>
</table>

144
## More Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ((\land, \lor, \text{ and } \neg))</td>
</tr>
<tr>
<td>Formula (\phi)</td>
<td>Combines vars and operations</td>
<td>((\bar{x} \land y) \lor (x \land \bar{z}))</td>
</tr>
<tr>
<td>Literal</td>
<td>A var or a negated var</td>
<td>(x \text{ or } \bar{x})</td>
</tr>
<tr>
<td>Clause</td>
<td><strong>Literals</strong> ORed together</td>
<td>((x_1 \lor \bar{x}_2 \lor \bar{x}_3 \lor x_4))</td>
</tr>
</tbody>
</table>
# More Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ((\land, \lor, \text{ and } \neg))</td>
</tr>
<tr>
<td>Formula (\phi)</td>
<td>Combines vars and operations</td>
<td>((\overline{x} \land y) \lor (x \land \overline{z}))</td>
</tr>
<tr>
<td>Literal</td>
<td>A var or a negated var</td>
<td>(x) or (\overline{x})</td>
</tr>
<tr>
<td>Clause</td>
<td>Literals ORed together</td>
<td>((x_1 \lor \overline{x_2} \lor \overline{x_3} \lor x_4))</td>
</tr>
<tr>
<td>Conjunctive Normal Form (CNF)</td>
<td>Clauses ANDed together</td>
<td>((x_1 \lor \overline{x_2} \lor \overline{x_3} \lor x_4) \land (x_3 \lor \overline{x_5} \lor x_6))</td>
</tr>
</tbody>
</table>

\(\land = \text{AND} = \text{“Conjunction”}\)
\(\lor = \text{OR} = \text{“Disjunction”}\)
\(\neg = \text{NOT} = \text{“Negation”}\)
### More Boolean Formulas

<table>
<thead>
<tr>
<th>A Boolean ______</th>
<th>Is ...</th>
<th>Example:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>TRUE or FALSE (or 1 or 0)</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>Variable</td>
<td>Represents a Boolean value</td>
<td>x, y, z</td>
</tr>
<tr>
<td>Operation</td>
<td>Combines Boolean variables</td>
<td>AND, OR, NOT ((\land, \lor, \text{ and } \neg))</td>
</tr>
<tr>
<td>Formula (\phi)</td>
<td>Combines vars and operations</td>
<td>((\overline{x} \land y) \lor (x \land \overline{z}))</td>
</tr>
<tr>
<td>Literal</td>
<td>A var or a negated var</td>
<td>(x) or (\overline{x})</td>
</tr>
<tr>
<td>Clause</td>
<td>Literals ORed together</td>
<td>((x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4))</td>
</tr>
<tr>
<td>Conjunctive Normal Form (CNF)</td>
<td>Clauses ANDed together</td>
<td>((x_1 \lor \overline{x}_2 \lor \overline{x}_3 \lor x_4) \land (x_3 \lor \overline{x}_5 \lor x_6))</td>
</tr>
<tr>
<td>3CNF Formula</td>
<td>Three literals in each clause</td>
<td>((x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (x_3 \lor \overline{x}_5 \lor x_6) \land (x_3 \lor \overline{x}_6 \lor x_4))</td>
</tr>
</tbody>
</table>

\(\land = \text{AND} = \text{“Conjunction”}\)
\(\lor = \text{OR} = \text{“Disjunction”}\)
\(\neg = \text{NOT} = \text{“Negation”}\)
The **3SAT** Problem

\[ 3SAT = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula} \} \]
Theorem: \( SAT \) is Poly Time Reducible to \( 3SAT \)

\[
SAT = \{ \phi \mid \phi \text{ is a satisfiable Boolean formula} \} \\
3SAT = \{ \phi \mid \phi \text{ is a satisfiable 3cnf-formula} \}
\]

To show poly time mapping reducibility:
1. create computable fn \( f \),
2. show that it runs in poly time,
3. then show forward direction of mapping red.,
   \[ \Rightarrow \text{ if } \phi \in SAT, \text{ then } f(\phi) \in 3SAT \]
4. and reverse direction
   \[ \Leftarrow \text{ if } f(\phi) \in 3SAT, \text{ then } \phi \in SAT \]
   (or contrapositive of forward direction)
   \[ \Leftarrow (\text{alternative}) \text{ if } \phi \notin SAT, \text{ then } f(\phi) \notin 3SAT \]
Theorem: \textbf{SAT} is Poly Time Reducible to \textbf{3SAT}

\[ SAT = \{ \phi \mid \phi \text{ is a satisfiable Boolean formula} \} \]
\[ 3SAT = \{ \phi \mid \phi \text{ is a satisfiable 3cnf-formula} \} \]

Need: poly time computable fn converting a Boolean formula \( \phi \) to 3CNF:

1. Convert \( \phi \) to CNF (an AND of OR clauses)
   a) Use DeMorgan’s Law to push negations onto literals
      \[ \neg(P \lor Q) \iff (\neg P) \land (\neg Q) \]
      \[ \neg(P \land Q) \iff (\neg P) \lor (\neg Q) \quad O(n) \]
   b) Distribute ORs to get ANDs outside of parens
      \[ (P \lor (Q \land R)) \iff ((P \lor Q) \land (P \lor R)) \quad O(n) \]

2. Convert to 3CNF by adding new variables
   \[ (a_1 \lor a_2 \lor a_3 \lor a_4) \iff (a_1 \lor a_2 \lor z) \land (\overline{z} \lor a_3 \lor a_4) \quad O(n) \]
Theorem: $3SAT$ is polynomial time reducible to $CLIQUE$.

3SAT = \{\langle \phi \rangle | \phi \text{ is a satisfiable 3cnf-formula} \}

CLIQUE = \{\langle G, k \rangle | G \text{ is an undirected graph with a } k\text{-clique} \}

To show poly time mapping reducibility:
1. create computable fn,
2. show that it runs in poly time,
3. then show forward direction of mapping red.,
4. and reverse direction
   (or contrapositive of forward direction)
**Theorem**: \( \text{3SAT} \) is polynomial time reducible to \( \text{CLIQUE} \).

\[ 3\text{SAT} = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-formula} \} \]

\[ \text{CLIQUE} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique} \} \]

**Need**: poly time computable fn converting a 3cnf-formula ...

- ... to a graph containing a clique:
  - Each clause maps to a group of 3 nodes
  - Connect all nodes except:
    - Contradictory nodes
    - Nodes in the same group

⇒ If \( \phi \in \text{3SAT} \)

- Then each clause has a TRUE literal
  - Those are nodes in the clique!
  - E.g., \( x_1 = 0, x_2 = 1 \)

⇔ If \( \phi \notin \text{3SAT} \)

- For any assignment, some clause must have a contradiction with another clause
- Then in the graph, some clause's group of nodes won't be connected to another group, preventing the clique

**Example:**

\[ \phi = (x_1 \lor x_1 \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor \overline{x_2}) \land (\overline{x_1} \lor x_2 \lor \overline{x_2}) \]

**Runs in poly time:**

- # literals = # nodes
- # edges poly in # nodes

\[ O(n) \]

\[ O(n^2) \]
Theorem: \(3SAT\) is polynomial time reducible to \(CLIQUE\).

\[3SAT = \{\langle \phi \rangle | \phi \text{ is a satisfiable 3cnf-formula}\}\]

\[CLIQUE = \{\langle G, k \rangle | G \text{ is an undirected graph with a } k\text{-clique}\}\]

- But this a single language reducing to another single language
NP-Completeness

**Definition**

A language $B$ is **NP-complete** if it satisfies two conditions:

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$.

**Theorem**

If $B$ is NP-complete and $B \leq_P C$ for $C$ in NP, then $C$ is NP-complete.

---

Must prove for all langs, not just a single language

It's very hard to prove NP-Completeness, but only for first problem!

(Just like figuring out the first undecidable problem was hard!)

After we find one, then we use that problem to prove other problems NP-Complete!
The Cook-Levin Theorem

The first NP-Complete problem

**THEOREM**

SAT is NP-complete.

But it makes sense that every problem can be reduced to it ...
The Cook–Levin Theorem

SAT is NP-complete.

The Complexity of Theorem-Proving Procedures
Stephen A. Cook
University of Toronto

Summary

It is shown that any recognition problem solved by a polynomial time-bounded nondeterministic Turing machine can be "reduced" to the problem of determining whether a given propositional formula is a tautology. Here "reduced" means, roughly speaking, that the first problem can be solved deterministically in polynomial time provided an oracle is available for solving the second. From this notion of reducibility, polynomial degrees of difficulty are defined, and it is shown that the problem of determining tautologyhood has the same polynomial degree as the certain recursive set of strings on this alphabet, and we are interested in the problem of finding a good lower bound on its possible recognition times. We provide no such lower bound here, but theorem 1 will give evidence that (tautologies) is a difficult set to recognize, since many apparently difficult problems can be reduced to determining tautologyhood. By reduced we mean, roughly speaking, that if tautologyhood could be decided instantly (by an "oracle") then these problems could be decided in polynomial time. In order to make this notion precise, we introduce query machines, which are like Turing machines with oracles.

1971

1973

КРАТКИЕ СООБЩЕНИЯ

УНИВЕРСАЛЬНЫЕ ЗАДАЧИ ПЕРЕБОРА

Л. А. Леми

В статье рассматриваются несколько известных массовых задач переборного типа и доказывается, что эти задачи можно решить лишь за такое время, за которое можно решить вообще любые задачи указанного типа.

После уточнения понятия алгоритма была доказана алгоритмическая неразрешимость ряда классических массовых проблем (например, проблем точности элементов групп, гомеоморфности многообразий, разрешимости диофантовых уравнений и других). Тем самым был снят вопрос о нахождении практического способа их решения. Однако существование алгоритмов для решения этих задач не дает для них аналогичного вопроса из-за фантастически большого объема работы, предсказываемого этими алгоритмами. Та же ситуация с так называемыми переборными задачами: минимизация булевых функций, поиска доказательства ограниченной длины, вычисления изоморфности графов и другими. Все эти задачи решаются тривиальными алгоритмами, состоящими в переборе всех возможностей. Однако эти алгоритмы требуют экспоненциального времени работы и у математиков сложилось убеждение, что

DEFINITION

A language $B$ is NP-complete if it satisfies two conditions:

1. $B$ is in NP, and
2. every $A$ in NP is polynomial time reducible to $B$. 

Hard part

15
Reducing every **NP** language to **SAT**

Some **NP** lang = \{w \mid w \text{ is ???}\}

**SAT** = \{⟨φ⟩ \mid φ \text{ is a satisfiable Boolean formula}\}

How can we reduce some w to a Boolean formula if we don’t know w???
Proving theorems about an entire class of langs?

We can still use general facts about the languages!

**THEOREM**

\[ \text{E.g., The class of regular languages is closed under the union operation.} \]

**PROOF**

uses the fact that every regular lang has an NFA accepting it

Let \( N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) \) recognize \( A_1 \), and \( N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) \) recognize \( A_2 \).

Construct \( N = (Q, \Sigma, \delta, q_0, F') \) to recognize \( A_1 \cup A_2 \).

**THEOREM**

- \( \text{E.g., } A_{\text{CFG}} \) is a decidable language. \( A_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \} \)

Proof uses the theorem that every CFG has a Chomsky Normal Form
What do we know about $NP$ languages?

They are:

1. Verified by a deterministic poly time verifier

2. Decided by a nondeterministic poly time decider (NTM)
Flashback: Non-deterministic TMs

- Formally defined with states, transitions, alphabet ...

A Turing machine is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where \(Q, \Sigma, \Gamma\) are all finite sets and

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet not containing the blank symbol \(\$$\$\),
3. \(\Gamma\) is the tape alphabet, where \(\$$ \in \Gamma\) and \(\Sigma \subseteq \Gamma\),
4. \(\delta: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})\) transition function,
5. \(q_0 \in Q\) is the start state,
6. \(q_{\text{accept}} \in Q\) is the accept state, and
7. \(q_{\text{reject}} \in Q\) is the reject state, where \(q_{\text{reject}} \neq q_{\text{accept}}\).

- Computation can branch
- Each node in the tree represents a TM configuration
Flashback: \( \text{TM Config} = \text{State} + \text{Head} + \text{Tape} \)

\[ 101101111 \ldots \]

Textual representation of “configuration”

1st char after state is current head position
Flashback: Non-deterministic TMs

- Formally defined with states, transitions, alphabet ...

A *Turing machine* is a 7-tuple, \((Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})\), where

1. \(Q\) is the set of states,
2. \(\Sigma\) is the input alphabet not containing the *blank symbol* \(\sqcup\),
3. \(\Gamma\) is the tape alphabet, where \(\sqcup \in \Gamma\) and \(\Sigma \subseteq \Gamma\),
4. \(\delta: Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})\) transition function,
5. \(q_0 \in Q\) is the start state,
6. \(q_{\text{accept}} \in Q\) is the accept state, and
7. \(q_{\text{reject}} \in Q\) is the reject state, where \(q_{\text{reject}} \neq q_{\text{accept}}\).

**Idea:** We don’t know the specific language or strings in the language, but ...

... we know those strings must have an accepting sequence of configurations!

- Computation can branch
- Each node in the tree represents a TM configuration
- Transitions specify valid configuration sequences
Accepting config sequence = “Tableau”

- input $w = w_1 \ldots w_n$
- Assume configs start/end with #
- Must have an accepting config
- At most $n^k$ configs
  - (why?)
- Each config has length $n^k$
  - (why?)
Theorem: SAT is NP-complete

• Proof idea:
  • Give an algorithm that reduces accepting tableaus to satisfiable formulas

• Thus every string in the NP lang will be mapped to a sat. formula
  • and vice versa

\[
SAT = \{ \langle \phi \rangle | \phi \text{ is a satisfiable Boolean formula} \}
\]

Resulting formulas will have four components:
\[
\phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{move}} \land \phi_{\text{accept}}
\]
Tableau Terminology

- A tableau **cell** has coordinate \( i,j \)

- A cell has symbol: \( s \in C = Q \cup \Gamma \cup \{#\} \)

A **Turing machine** is a 7-tuple, \( (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}) \), where \( Q, \Sigma, \Gamma \) are all finite sets and

1. \( Q \) is the set of states,
2. \( \Sigma \) is the input alphabet not containing the **blank symbol** \( \sqcup \),
3. \( \Gamma \) is the tape alphabet where \( \sqcup \in \Gamma \) and \( \Sigma \subseteq \Gamma \),
4. \( \delta : Q \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\}) \) is the transition function,
5. \( q_0 \in Q \) is the start state,
6. \( q_{\text{accept}} \in Q \) is the accept state, and
7. \( q_{\text{reject}} \in Q \) is the reject state, where \( q_{\text{reject}} \neq q_{\text{accept}} \).
Formula Variables

- A tableau cell has coordinate $i,j$

- A cell has symbol: $s \in C = Q \cup \Gamma \cup \{\#\}$

- For every $i,j,s$ create variable $x_{i,j,s}$
  - i.e., one var for every possible symbol/cell combination

- Total variables =
  - # cells * # symbols =
  - $n^k \times n^k \times |C| = O(n^{2k})$
Check-in Quiz 11/15
On gradescope