Time to learn about NP-completeness!

Craig Weidert

Harvey Mudd College

March 19, 2007
Languages

• A language is a set of strings

• Examples
  • The language of strings of all “a”s with odd length
  • The language of strings with the same number of “a”s and “b”s
Languages

• A language is a set of strings
• Examples
  • The language of strings of all “a”s with odd length
  • The language of strings with the same number of “a”s and “b”s
• If we can figure out whether to accept or reject any particular string as part of a language, we say that that language is **decidable**.
Turing Machines

- One convenient computation model: The Turing Machine
  - Tape containing symbols from an alphabet
  - States
  - Transition rules
- Example: a Turing Machine that decides the language of strings of “a”s with odd length.

```
# a a a a a a a # ...
```
Non-Deterministic Turing Machines

- Instead of having just one transition rule per state per symbol read on the tape, it may have many.
- Allows branching (like running many machines in parallel).
- If any branch reaches the accepting state, then the machine accepts the string.
Time Complexity

- **Big-Oh notation**
  - Counting your toes takes $O(\text{number of toes})$ time.
  - Adding two $n$ bit numbers takes $O(n)$ time.
  - Multiplying two $n \times n$ matrices takes $O(n^3)$ time.

- Language is polynomial time decidable if any string is either accepted or rejected in time proportional to some polynomial in the size of the string.
Polynomial Time Reducibility

- Language A is reducible to language B if there is some mapping from strings to strings such the first string is in language A iff the mapping of that string is in language B.
- Silly Example: the language of strings of odd length can be reduced to the language of strings of even length.
- Complicated Example: From the last talk, 3-SAT can be reduced to Graph 3-Colorability.
Polynomial Time Reducibility

• Language A is reducible to language B if there is some mapping from strings to strings such the first string is in language A iff the mapping of that string is in language B.
• Silly Example: the language of strings of odd length can be reduced to the language of strings of even length
• Complicated Example: From the last talk, 3-SAT can be reduced to Graph 3-Colorability
• If you can perform this reduction in polynomial time, A is polynomial time reducible to B.
$P$ vs $NP$

- $P$ is the class of languages which can be decided in polynomial time by a deterministic Turing Machine
- $NP$ is the class of languages which can be decided in polynomial time by a non-deterministic Turing Machine
- Equivalently, $NP$ is the class of languages which can be verified in polynomial time
- $P \subseteq NP$
P vs NP

- $P$ is the class of languages which can be decided in polynomial time by a deterministic Turing Machine.
- $NP$ is the class of languages which can be decided in polynomial time by a non-deterministic Turing Machine.
- Equivalently, $NP$ is the class of languages which can be verified in polynomial time.
- $P \subseteq NP$
- $P = NP$?
  - Worth $1,000,000$
NP-Completeness

- Two requirements for a language to be \( \mathcal{NP} \)-complete
  - The language must be in \( \mathcal{NP} \)
  - Any other language in \( \mathcal{NP} \) is polynomial time reducible to that language
NP-Completeness

- Two requirements for a language to be $\mathcal{NP}$-complete
  - The language must be in $\mathcal{NP}$
  - Any other language in $\mathcal{NP}$ is polynomial time reducible to that language

- Implications of $\mathcal{NP}$-completeness
  - Polynomial time algorithm for any $\mathcal{NP}$-complete language yields a polynomial time algorithm for all languages in $\mathcal{NP}$ and means that $\mathcal{P} = \mathcal{NP}$. 
SAT Review

- Variables may either be true or false.
- Variables may be NOTed (with $\neg$), ORed (with $\lor$), or ANDed (with $\land$).
- Example formula: $(a \lor \neg b \lor c \lor \neg d) \land (\neg a \lor c \lor d)$
- SAT is the language which is a collection of formulas such that there is some assignment of variables that makes the formula true.
Cook’s Theorem - SAT is $\mathcal{NP}$-complete

- SAT $\in \mathcal{NP}$: easy to verify given a correct assignment of variables
- Need to show that any language B in NP can be reduced in polynomial time to SAT
  - Language B can be decided by a non-deterministic Turing Machine in $n^k$ time for some constant $k$.
  - We can build a huge formula to simulate a Turing Machine running on a string to decide whether it’s in language B.
Tableau

<table>
<thead>
<tr>
<th>#</th>
<th>$q_{start}$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$\ldots$</th>
<th>$w_l$</th>
<th>$\square$</th>
<th>$\ldots$</th>
<th>$\square$</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Essentially a simulates of the tape and state of the Turing Machine at different steps
- Doesn’t need to be more than $n^k$ wide
Formula Overview

- Variables for each possible symbol in cell of tableau: \( x_{i,j,c} \)
  - \( i \) and \( j \) correspond to the row and column of the tableau
  - \( c \in [(\text{alphabet of TM}) \cup (\text{states of TM}) \cup \#] \) correspond to different things which may be in cells
- Variable is true if there that particular cell contains that particular symbol
- Will make one massive SAT formula \( \phi \) corresponding to tableau for an instance
- \( \phi = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{accept}} \land \phi_{\text{transition}} \)
Cell Subformula

\[
\phi_{\text{cell}} = \bigwedge_{1 \leq i, j \leq n^k} \left[ \left( \bigvee_{s \in C} x_{i,j,s} \right) \wedge \left( \bigwedge_{s, t \in C \,(s \neq t)} (\overline{x_{i,j,s}} \vee \overline{x_{i,j,t}}) \right) \right]
\]

In English: “Every cell in the tableau must have exactly one symbol.”
Start and Accept Subformulas

- **Start subformula**
  
  \[
  x_{1,1}, \# \land x_{1,2}, q_{\text{start}} \land x_{1,3}, w_0 \land \cdots \land x_{1,l}, w_l \land x_{1,l+1}, \sqcup \land \cdots \land x_{1,n^k-1}, \sqcup \land x_{1,n^k}, \# \]

  - In English: “The first row in the tableau must correspond to the input.”

- **Accept subformula**

  \[
  \bigvee_{1 \leq i, j \leq n^k} x_{i,j}, q_{\text{accept}}
  \]

  - In English: “Somewhere in our tableau, we have record of being in an accepting state.”
Transition Subformula

- Ensures that our transitions from one row to another are legal.
- Since TM can only move one cell at a time, enough to look at all $2 \times 3$ windows.

<table>
<thead>
<tr>
<th>#</th>
<th>$q_{\text{start}}$</th>
<th>$w_1$</th>
<th>$w_2$</th>
<th>$\ldots$</th>
<th>$w_I$</th>
<th>$\square$</th>
<th>$\ldots$</th>
<th>$\square$</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#</td>
</tr>
<tr>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>#</td>
</tr>
</tbody>
</table>

[Table representation of the transition subformula with a highlighted section showing legal transitions]
Transition Subformula (cont.)

Legal Windows

<table>
<thead>
<tr>
<th>#</th>
<th>q_{even}</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>a</td>
<td>q_{odd}</td>
</tr>
<tr>
<td>#</td>
<td>a</td>
<td>q_{even}</td>
</tr>
<tr>
<td>#</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>

Illegal Windows

<table>
<thead>
<tr>
<th>#</th>
<th>q_{even}</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>a</td>
<td>q_{even}</td>
</tr>
<tr>
<td>#</td>
<td>q_{even}</td>
<td>a</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>q_{odd}</td>
</tr>
</tbody>
</table>

\( \phi_{transition} = \bigwedge_{1 < i \leq n^k, 1 < j < n^k} (\text{the window centered at } i, j \text{ is ok}) \)

In English: “Make sure that all of the windows in our tableau jive with our transition states.”
Reduction takes Polynomial Time

- $\phi = \phi_{\text{cell}} \land \phi_{\text{start}} \land \phi_{\text{accept}} \land \phi_{\text{transition}}$
- Size of formula
  - $\phi_{\text{start}}$ is size $O(n^k)$
  - $\phi_{\text{cell}}, \phi_{\text{cell}},$ and $\phi_{\text{accept}}$ are all of size $O(n^k \times n^k)$
- Formation of formula
  - The formation of the formula is very repetitive
Proof Review

- SAT is in $\mathcal{NP}$
- Any language in $\mathcal{NP}$ is polynomial time reducible to SAT
  - Languages in $\mathcal{NP}$ are decidable by a non-deterministic Turing Machine in polynomial time
  - Can reduce this machine’s running into a formula
  - Formula $\phi = \phi_{\text{start}} \land \phi_{\text{cell}} \land \phi_{\text{transition}} \land \phi_{\text{accept}}$ is satisfiable iff Turing Machine can run to acceptance
  - Reduction in polynomial time
Proof Review

- SAT is in \( \mathcal{NP} \)
- Any language in \( \mathcal{NP} \) is polynomial time reducible to SAT
  - Languages in \( \mathcal{NP} \) are decidable by a non-deterministic Turing Machine in polynomial time
  - Can reduce this machine’s running into a formula
  - Formula \( \phi = \phi_{\text{start}} \land \phi_{\text{cell}} \land \phi_{\text{transition}} \land \phi_{\text{accept}} \) is satisfiable iff Turing Machine can run to acceptance
  - Reduction in polynomial time
- So, SAT is \( \mathcal{NP} \)-complete!
3-SAT $\mathcal{NP}$-Complete

• Now that we know SAT is $\mathcal{NP}$-complete, to show that another is $\mathcal{NP}$-complete, we only have to show that
  • it’s in $\mathcal{NP}$ and
  • that we can reduce SAT to it in polynomial time

• 3-SAT is $\mathcal{NP}$-Complete
  • 3-SAT is verifiable in polynomial time
  • For every clause $(a_1 \lor a_2 \lor \cdots \lor a_l)$ in SAT, make the clauses
    $(a_1 \lor a_2 \lor z_1) \land (\neg z_1 \lor a_3 \lor z_2) \land \cdots \land (\neg z_{l-3} \lor a_{l-1} \lor a_l)$ in 3-SAT
Presentation based on the proof from Prof. Pippenger’s Complexity Theory class and Cook’s Theorem in Michael Sipser’s “Introduction to the Theory of Computation” 2nd ed. 2006.