Time to learn about NP-completeness!

Craig Weidert

Harvey Mudd College

March 19, 2007

Languages

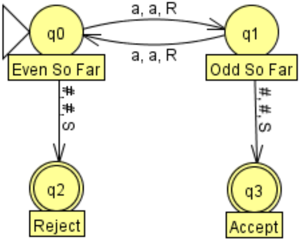
- A language is a set of strings
- Examples
  - The language of strings of all "a"s with odd length
  - The language of strings with the same number of "a"s and "b"s

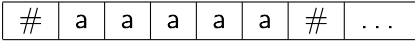
Languages

- 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.





Non-Deterministic Turing Machines

- Instead of a 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 $\mathcal{O}(\text{number of toes})$ time.
  - Adding two n bit numbers takes $\mathcal{O}(n)$ time.
  - Multiplying two $n \times n$ matrices takes $\mathcal{O}(n^3)$ 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.

$\mathcal{P}$ vs $\mathcal{NP}$

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

$\mathcal{P}$ vs $\mathcal{NP}$

- $\mathcal{P}$ is the class of languages which can be decided in polynomial time by a deterministic Turing Machine
- $\mathcal{NP}$ is the class of languages which can be decided in polynomial time by a non-deterministic Turing Machine
- Equivalently, $\mathcal{NP}$ is the class of languages which can be verified in polynomial time
- $\mathcal{P} \subseteq \mathcal{NP}$
- $\mathcal{P} = \mathcal{NP}$?
  - 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 $\vee$), or ANDed (with $\wedge$).
- Example formula: $(a \vee \neg b \vee c \vee \neg d) \wedge (\neg a \vee c \vee d)$
- 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

#	q_{start}	w_1	w_2	$\dots$	w_l	$\sqcup$	$\dots$	$\sqcup$	#	starting config.
#									#	second config.
#									#	n^kth config.

- Essentially a similates 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 ϕ corresponding to tableau for an instance
- $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{transition}}$

Cell Subformula

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

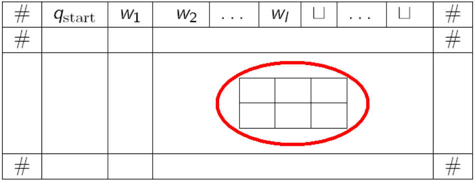
In English: "Every cell in the tableau must have exactly one symbol."

Start and Accept Subformulas

- Start subformula
  - $x_{1,1,\#} \wedge x_{1,2,q_{\text{start}}} \wedge x_{1,3,w_0} \wedge \cdots \wedge x_{1,l,w_l} \wedge x_{1,l+1,\sqcup} \wedge \cdots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}$
  - 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.



Transition Subformula (cont.)

Legal Windows

#	q_{even}	a
#	a	q_{odd}

#	a	q_{even}
#	a	a

Illegal Windows

#	q_{even}	a
#	a	q_{even}

#	q_{even}	a
a	a	q_{odd}

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

In English: "Make sure that all of the windows in our tableau jive with our transition states."

Reduction takes Polynomial Time

- $\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{transition}}$
- Size of formula
  - ϕ_{start} is size $\mathcal{O}(n^k)$
  - ϕ_{cell}, ϕ_{cell}, and ϕ_{accept} are all of size $\mathcal{O}(n^k * n^k)$
- 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 $\phi = \phi_{\text{start}} \wedge \phi_{\text{cell}} \wedge \phi_{\text{transition}} \wedge \phi_{\text{accept}}$ 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 $\phi = \phi_{\text{start}} \wedge \phi_{\text{cell}} \wedge \phi_{\text{transition}} \wedge \phi_{\text{accept}}$ 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 $(a_1 \vee a_2 \vee \cdots \vee a_l)$ in SAT, make the clauses $(a_1 \vee a_2 \vee z_1) \wedge (\neg z_1 \vee a_3 \vee z_2) \wedge \cdots \wedge (\neg z_{l-3} \vee a_{l-1} \vee a_l)$ 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 Compuation" 2nd ed. 2006.