CSE 431 Theory of Computation **Spring 2014**

Lecture 12: May 8

Lecturer: James R. Lee *Scribe: Brandon Krueger*

Cook-Levin Theorem

12.1 Review of NP-completeness

Definition 12.1 *A language L is NP-complete if*

1. $A \in NP$
2. *For every language* $B \in NP, B \leq_p L$

Definition 12.2 *Reduction:* $B \leq_p L$

There is a poly-time computable map $f : \Sigma^* \to \Sigma^*$ *such that* $w \in B \leftrightarrow f(w)\ L\ \forall w$

Note that this notation does not imply that f is symmetric.

12.2 Agenda Items

- SAT is NP-complete
- 3-SAT is NP-complete
- CLIQUE is NP-complete
- HAMPATH is NP-complete

12.3 SAT is NP-complete

Consider the following SAT formula:

$\phi = (x_1 \wedge (\overline{x_2} \vee x_3)) \vee (\overline{x_4} \wedge x_5) \wedge (x_1 \vee x_5)$

Question: Is there an assignment to the variables that makes the formula true? In other words, does there exist a satisfying assignment? The class of problems that address this question are known at SAT.

Claim 12.3 *SAT is NP-complete*

SAT must satisfy both of the requirements for NP-completeness.

12.3.1 SAT is in NP

First it will be considered whether or not $SAT \in NP$.

Certificate: An assignment to the x_i variables. Finding this certificate could be exceptionally difficult, potentially requiring 2^n checks through brute force.

Verifier: Check in polynomial time (i.e. $c*n^k$) that the certificate satisfies ϕ when evaluated with the variable assignments. Thus, the requirement for $SAT \in NP$ is satisfied since there exists a polynomial time verifier.

12.3.2 SAT is NP-Hard: $B \leq_p SAT$

Proposition 12.4 *Consider a language $B \in NP \rightarrow B$ has a decider M running in non-deterministic polynomial-time.*

Basic idea of the proof requires a tableau, which is an n^k by n^k table with an input $w \in \Sigma^*$ of length n.

#	w_1	w_2	w_3	...	w_n	--	--	#
#								#
#								#
#			cell[i,j]					#
#								#
#								#

Given some input string $w \in \Sigma^*$ of length n, the table will be initialized by writing w in the top row of the table. If the machine M accepts the input string w, then there should exist a sequence of configurations that ends in an accept state. The notion of acceptance is that there exists at least one non-deterministic execution that leads to an accept state. The machine M always runs in n^k time, so this table will always have enough room for the sequence of configurations that it accepts. Now we want to write down a SAT formula that captures the sequence of constraints. In other words, the SAT formula will be satisfiable if and only if there exists a way to fill in the table where it reaches an accept state.

The idea is to take the input w and map it to a SAT formula ϕ (i.e. $f(w)$) which should happen in polynomial time such that w is in B if and only if ϕ is satisfiable.

Proposition 12.5 *Given $w \in \Sigma^* \xrightarrow[polytime]{} \phi = f(w)$ where $w \in B \leftrightarrow \phi \in SAT$*

Variables for the formula: $x_{i,j,s} 1 \leq j, j \leq n^k; s \in Q \cup \Gamma \cup \{\#\}$

The variable s represents all possible symbols that could be written on the tape (i.e. tape alphabet).

$$x_{i,j,s} = \begin{cases} 1 & \text{if cell}[i,j] = \text{s;} \\ 0 & \text{otherwise.} \end{cases}$$

Formula: $\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept} \wedge \phi_{move}$

First, ϕ_{cell} will represent the fact that every cell contains exactly one symbol. Second, ϕ_{start} will ensure that the assignment to the variables encodes the correct start state for the machine (i.e. first line of the table). Third, ϕ_{accept} will guarantee that one of the lines represents an accept state. Finally, ϕ_{move} will verify that the table is both valid and consistent. Validity is determined by every sequence of configurations performing a proper transition. The size of each ϕ value is bounded by size $\leq O(n^{2k})$.

$\phi_{cell} = \bigwedge 1 \leq i, j \leq n^k (\bigvee s \in c\ x_{i,j,s}) \wedge (\bigwedge s \neq s' \in c(\overline{x_{i,j,s}} \vee \overline{x_{i,j,s'}}))$

Example: $(x_{1,1,a} \vee x_{1,1,b} \vee x_{1,1,\#} \vee x_{1,1,\_}) \wedge \ldots$ represents the fact that something must exist in cell 1,1.

$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_{start}} \wedge x_{1,3,w_1} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge x_{1,n+3,\_} \wedge \ldots \wedge x_{1,n^k,\#}$

$\phi_{accept} = \bigvee 1 \leq i \leq n^k (\bigvee 1 \leq j \leq n^k x_{i,j,q_{accept}})$

$\phi_{move} = \bigwedge 1 \leq i, j \leq n^k (\bigvee$ legal windows $x_{i,j,a} \wedge x_{i,j+1,b} \wedge x_{i,j+2,c} \wedge x_{i+1,j,d} \ldots)$

Example alphabet and potential states:

$\Gamma = \{a, b, \_\}$

$Q = \{q_1, q_2\}$

Example transitions:

$\delta(q_1, a) = \{(q_1, b, R)\}$

$\delta(q_1, b) = \{(q_2, , L), (q_2, a, R)\}$

On a Turing machine, computation is local, so the idea is that the validity of a window can also be checked locally. As such, provided are some legal window examples:

a	a	a
a	a	a

a	a	a
q_1	a	a

q_1	b	a
a	$\_\_$	a

The total number of possible windows is some constant value (i.e. $|C|^6$).

Claim 12.6 *ϕ is satisfiable $\leftrightarrow$ there exists a valid accepting tableau $\leftrightarrow$ $w \in B$*

Proof: Suppose ϕ is satisfiable. Let $\{x_{i,j,s}\}$ be a satisfying assignment.

First direction: If there exists a valid accepting tableau $\rightarrow$ ϕ is satisfiable.

The satisfying assignment is the assignment that corresponds to setting all the cells. Thus, ϕ_{cell} is satisfied since all the cells contain exactly one symbol. Since the table is a valid accepting tableau, ϕ_{start} must be initialized correctly. Finally, ϕ_{move} is satisfied because the tableau has all legal windows.

Opposite direction: If ϕ is satisfiable $\rightarrow$ there exists a valid accepting tableau.

The table can be filled in according to the assignment. The start state is guaranteed to be valid, because q_{start} is satisfied if the formula is satisfied. Legal windows enforce the fact that a cell cannot be written to unless the tape head occupies that cell. Additional tape heads also cannot be created since the legal windows would disallow it. The three additional cell entries being copied while shifting the window downward must also be valid since the initial state was valid and all transitions are valid. Thus, if ϕ is satisfiable, then all the windows are legal, which implies that all states are a result of a sequence of valid configurations.

In summary, we started with a language in NP, which had a non-deterministic Turing machine that decided it. When the execution of the machine was encoded with the tableau, the machine accepted B if and only if there was a valid accepting tableau. Thus, the sequence of configurations ends somewhere in an accept state. Given some input w, the formula ϕ was produced such that ϕ is satisfiable if and only if there was an accepting tableau on input w. This formula also produced the mapping of w in polynomial time. Finally, since this was verified to hold for every B in NP, this implies that for every language $B \in NP$, $B \leq_p$ SAT. Therefore, it may be concluded by definition that SAT is NP-complete. ■