**CS 810: Introduction to Complexity Theory** 2/25/03

# Lecture 11: Mahaney's Theorem

Instructor: Jin-Yi Cai Scribe: Rachel Heck, Chris Kaiserlian, Asher Langton

In last lecture, we discussed Karp–Lipton theorem, which says that if SAT has polynomial size circuits, then the polynomial hierarchy collapses to the second level. In other words, if SAT Turing reduces to some sparse set S, then the hierarchy collapses to the second level. A natural question at this point is, what happens if SAT many-one reduces to a sparse set. Notice that this is a stronger assumption. Mahaney's theorem says that if SAT many-one reduces to a sparse set, the NP=P. The theorem also has implications in the context of Berman-Hartmanis conjecture. We will discuss it in next lecture. In this lecture, we prove Mahaney's theorem.

## 1 Mahaney's Theorem

**Definition 1.** *Let $\tau = \tau_1\tau_2\ldots\tau_k$ and $\sigma = \sigma_1\sigma_2\ldots\sigma_n$ be binary strings, with $k \leq n$. Then we say $\sigma$ is to the left of $\tau$ (denoted $\sigma \preceq_l \tau$) if either $\tau$ is a prefix of $\sigma$ or, after padding $\tau$ with $n-k$ 0s on the right, $\sigma$ precedes $\tau$ lexicographically. Formally, let $i = max\{j \mid \forall j' < j, \tau_{j'} = \sigma_{j'}\}$. Then $\sigma \preceq_l \tau$ if either $i > k$ or $i \leq k$ with $\sigma_i < \tau_i$.*

We can view the above definition pictorially as follows. Consider the binary tree of partial assignments to the $n$ variables. Let $\sigma$ and $\tau$ be two partial assignments. $\sigma$ is said to be the *left* of $\tau$, if $\sigma$ is below $\tau$ (i.e. $\tau$ is a prefix of $\sigma$) or $\sigma$ is in a branch to the left of $\tau$.
We next define a language called *left cut* of SAT. Let $\varphi$ be a formula and $\tau$ be a partial assignment. The pair $\langle\varphi, \tau\rangle$ is in $L_{\text{SAT}}$, if some assignment $\sigma$ to the left of $\tau$ satisfies $\varphi$. Formally,

**Definition 2.** *A leftcut of* SAT *is the set* $\text{L}_{\text{SAT}} = \{\langle\varphi, \tau\rangle \mid \varphi$ *is a formula on $n$ variables and* $\tau \in \{0,1\}^k, 0 \leq k \leq n,$ *such that* $\exists\sigma \in \{0,1\}^n$ *with* $\sigma \preceq_l \tau$ *and* $\varphi|_\sigma =$ *True.*$\}$

The crucial property of $L_{\text{SAT}}$ is the following. Let $\varphi$ be a formula and $\sigma$ and $\tau$ be two partial assignments, such that $\sigma \preceq_l \tau$. Then, if $\langle\varphi, \sigma\rangle$ is in $L_{\text{SAT}}$, then $\langle\varphi, \tau\rangle$ is also in $L_{\text{SAT}}$. Suppose we are given $\langle\varphi, \sigma\rangle$ and $\langle\varphi, \tau\rangle$ and we have to bet on one of these to be in $\text{L}_{\text{SAT}}$. Clearly, we should place our bet on $\langle\varphi, \tau\rangle$. The proof hinges on this property of $\text{L}_{\text{SAT}}$.

**Theorem 1.** *(Mahaney; proof by Ogiwara and Watanabe) For any sparse set $S$,* $\text{SAT} \leq_m^P S \iff \text{P} = \text{NP}$.

*Proof.* Let $S$ be a sparse set. The implication $\text{P} = \text{NP} \implies \text{SAT} \leq_m^P S$ is trivial, so we need only prove that $\text{SAT} \leq_m^P S \implies \text{P} = \text{NP}$.
Suppose $\text{SAT} \leq_m^P S$. We shall design a polynomial time algorithm to solve SAT. First of all, notice that $\text{L}_{\text{SAT}}$ is in NP. (Given $\langle\phi, \tau\rangle$, we guess a (full) truth assignment $\sigma$, then verify that $\sigma \preceq_l \tau$ and $\sigma$ satisfies $\phi$; if so accept, else reject.) From the assumption $\text{SAT} \leq_m^P S$, it follows that $\text{L}_{\text{SAT}} \leq_m^P S$, via some polynomial time computable function $f$.
We are ready to design a polynomial time algorithm for SAT. Let $\varphi$ be the input formula over $n$ variables and we need to check if it is satisfiable or not. If $\varphi$ is satisfiable, the algorithm would, in fact, output the (lexicographically) left-most satisfying assignment.
Let us consider the binary tree formed by assignments on $\varphi$. The root of this tree corresponds to the empty assignment (i.e. no variable is assigned a value). Nodes of the tree correspond to

partial assignments. We will do a breadth-first search on this tree, starting with the root. The tree has exponentially many nodes and we cannot hope to do a full search. Instead, as we go along, we will prune the tree and explore only parts of the tree. At any point of time, we will maintain only polynomially many nodes. We will ensure that, if $\varphi$ is satisfiable, the left-most satisfying assignment is not pruned away.

Consider strings of the form $\langle \varphi, \sigma \rangle$, where $\sigma$ is some partial assignment for the input formula $\varphi$. Length of these strings is at most $|\varphi| + n$. As the reduction $f$ runs in polynomial time, its output on these strings can be at most polynomial in $n$. Let this number be $l$. The sparse set can have at most $N$ strings of length $\leq l$, where $N$ is polynomial in $l$, and it turn polynomial in $n$. As we explore the binary tree, level by level, we will maintain at most $N$ nodes, at any point of time.

First, we explore the tree until we reach a level $k$, where number of nodes $2^k > N$. At this point, we start pruning. We run the reduction $f$ on all these $2^k$ partial assignments (with $\varphi$ as the formula) and obtain $2^k$ output strings. We first look for duplicates. Let $\sigma_1$ and $\sigma_2$ be two partial assignments at level $k$ such that $f(\sigma_1) = f(\sigma_2)$. We throw away the right-most assignment among these two (i.e. if $\sigma_1 \preceq_l \sigma_2$, then throw away $\sigma_2$, else throw away $\sigma_1$). After removing all duplicates, if more than $N$ nodes survive, we throw away the left-most partial assignment at this level (i.e. we will not explore that subtree any further). We keep removing the left-most nodes until we have only $N$ nodes. Clearly, we will have at most $N$ nodes that survive. We continue to explore only these subtrees, in a similar fashion. That is, suppose $r \leq N$ nodes survive at level $k$. In level $k+1$, we consider their $2r$ children. If $2r \leq N$, we can move on to level $k+2$. If not, we run the reduction $f$ on these $2r$ nodes, and obtain $2r$ output strings. We first eliminate duplicates and then the left-most nodes, if necessary, until we have at most $N$ nodes. Continuing this way, finally we will reach the leaf level of the tree. Here, all the nodes correspond to full truth assignments of $\varphi$. We will have at most $N$ surviving full assignments. For each of these assignments, we check if any of them satisfy $\varphi$. If so $\varphi$ is satisfiable. Otherwise, it is not satisfiable.

Clearly, we maintain at most $N$ nodes at any level of the tree and $N$ is polynomial in $n$. Depth of the tree in $n$. Thus the algorithm runs in polynomial time. Suppose $\varphi$ is satisfiable. We shall argue that the left-most satisfying assignment will survive at the leaf-level. We do this by induction on level number. Consider level $k$ and let number of nodes surviving at this level be $r$. By induction, assume that the left-most satisfying assignment didn't get pruned away (i.e. it is the descendant of one of the $r$ surviving nodes). We consider the $2r$ children at level $k+1$. If $2r > N$, we start pruning. In this case, we run the reduction $f$ on these $2r$ partial assignment. We first remove duplicates. Suppose $f(\sigma_1) = f(\sigma_2)$ and $\sigma_2$ is to the right of $\sigma_1$. In this case, we throw away $\sigma_2$. We claim that, the left-most satisfying assignment is not a descendant of $\sigma_2$. By contradiction, suppose the left-most satisfying assignment $t$ is a descendant of $\sigma_2$. Then, clearly, $\langle \varphi, \sigma_2 \rangle \in \mathrm{L_{SAT}}$. As $f(\sigma_1) = f(\sigma_2)$ and $f$ is a reduction from $\mathrm{L_{SAT}}$ to $S$, $\langle \varphi, \sigma_1 \rangle \in \mathrm{L_{SAT}}$. So, there is some truth assignment $t'$ to the left of $\sigma_1$. Note that $t$ is to the right of $\sigma_1$. Thus, $t'$ is to the left of $t$ and hence, $t$ cannot be the left most satisfying truth assignment: a contradiction. Thus, throwing away duplicates is a correct procedure. After removing duplicates, if $d > N$ nodes survive, we removed the left-most node, say $\sigma$. Let these nodes be $\sigma_1, \sigma_2, \ldots, \sigma_d$. We argue that this is also a correct procedure. By contradiction, suppose the left-most satisfying assignment $t$ is a descendant of $\sigma$. Then, $t$ is to the left of all the $d$ surviving nodes. Thus, all of $\langle \varphi, \sigma_1 \rangle, \langle \varphi, \sigma_2 \rangle, \ldots \langle \varphi, \sigma_d \rangle$ are in $L_{\mathrm{SAT}}$. So, $f(\langle \varphi, \sigma_1 \rangle), f(\langle \varphi, \sigma_2 \rangle), \ldots f(\langle \varphi, \sigma_d \rangle)$ are all in $S$. Note that, all these strings are distinct and there $d > N$ of them. But, by our assumption of sparseness of $S$, there can be at most $N$ strings in $S$ (at this appropriate length). This contradiction shows that throwing away the left-most node is also a correct procedure. $\square$