CSC 2401 Introduction to Computer Complexity - Fall 2001
Lecture #9, Thursday, Octoer 11, 2001
Lecturer: Allan Borodin
Scribe: Pixing Zhang

Lecture #9

9.1 Enumeration Set

Before we begin todays main topic, NP and the proof that CIRCUIT-SAT is NP-complete, we want to respond to a question that came up relating to the first assignment. Namely, the question is whether or not the class of time constructible functions is "enumerable". (If it were enumerable, then it would be easy to diagonalize over it.)

For a class C of functions(languages), C is *enumerable* if there exists some listing $\{R_1($ say machine $M_1), R_2(M_2), ...\}$ such that:

- each $R_i(M_i)$ represents(for example, M_i computes) a member of C.
- every member of C is represented at least once in the list.

For example,

- $C = \{$ Partial Computable Functions $\}$
- $C = \{L : \{0,1\}^* \to \{0,1\} | \text{s.t. } L \in DTIME(cn^3), \text{ where } c \text{ is some constant}\}$

are enumerable sets.

In fact, all complexity class are enumerable. (We could see this by appending a time counter to each TM M_i to form a new machine M_i' and then the listing of the M' machines shows that the class is enumerable.)

When we talk about a complexity class that is defined as a countable union of complexity classes such as the polynomial time class P, we can either appeal to the fact that a countable collection of enumerable sets is still enumerable or equivalently we can see how we could make a one-one correspondence $N \times N \leftrightarrow N$.

For example, $M'_{\langle 0,0\rangle}, M'_{\langle 0,1\rangle}, ...M'_{\langle c,i\rangle}$ could be enumerating P by letting $M'_{\langle c,i\rangle}$ denote machine M_i with a n^c counter.

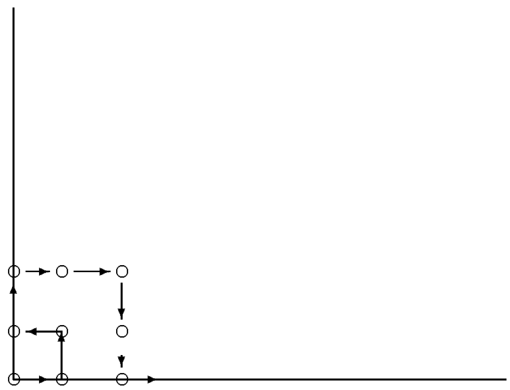

Figure 1: Pairing

Now let us consider the following question: Is the class of all time constructible functions enumerable? Let us simplify and formalize its definition as following: $C = \{f | f : N \rightarrow N, f(1^n)$ is computable in time $2 \cdot f(1^n)$. The question is: Is C enumerable?

Although it is plausible to say yes based on the similarity to say the class of languages acceptable in time $2n$, the answer is NO. By using a diagonalization, we could always construct a new (larger) time constructible function from the other time constructible functions.

9.2 An Alternative NP definition

When we talk about NP, we know that it is the class of languages acceptable in nondeterministic polynomial time. But when we prove our first NP-completeness result and more generally when we want to show that a language is in NP, , it might be more convinient to use the following definition than the original one:

$L \in NP$ iff $L = \{x | \exists y, |y| \leq q(|x|)$ and $R(\langle x, y \rangle)$ is true, $q(n)$ is some polynomial$\}$, where R is a binary predicate that can be decided within some polynomial time.

Intuitively speaking, $x \in L$ iff x has a short *certificate* y which can be efficiently verified (by the *verification predicate*) R.

For the SAT problem, given a formula x, assignment y is a certificate.

For the graph $3-coloring$ problem, a graph's certificate is a $3-coloring$.

We need to argue that this definition is equivalent to the orignial defintion. In one direction (showing the new definition implies the old), the NTM works by guessing a y, then verifing $\langle x, y\rangle$ in polynomial time. Conversely, suppose the Turing machine has at most d choices per move and it is bounded by at most $p(n)$ steps. The path on computation tree is described by a string over {0,1,2,...,d} of length $p(n)$. Once this string is told, the path to follow is determined.

Using this characterization, it is usually easy to check if a language is in NP although for some NP languages defining the concept of a certificate and verification is not obvious (e.g. for "primes" in contrast to "composites").

9.3 Boolean Circuits and Some Notation

A boolean circuit is a a labelled DAG where the nodes are labelled by boolean gates(e.g. $\neg, \vee, \wedge$) and the edges represent connections between the gtaes. There is a set of input gates and we usually assume just one output gate. The above three gate types are called a basis in that any boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ can be represented (computed) by some boolean circuit using these gates.

We can view propositional formulas as a special case of boolean circuits in which each gate (other than an input gate and the output gate) has fan-out 1. An input gate has arbitrary fan-out and the output gate has fan-out 0. For a given set of connectives (eg $\neg, \vee, \wedge$) we define formulas inductively: Each propositional variable is a formula; If f_1, f_2 are formulas, so are $f_1 \vee f_2, f_1 \wedge f_2, \neg f_1$.

DNF(CNF)-Disjunction Normal Form(respectively, Conjunction Normal Form) are defined over literals of the form $x_i, \bar{x_i}$, the terms (clauses) are conjunctions (disjunctions) of literals and the formula are disjunctions (conjunctions) of terms (clauses). DNF and CNF are duals of each other. It is easy to see that every formula has an DNF and CNF(but not 3CNF,3DNF-3CNF(3DNF) has at most(exactly) 3 literals per clauses.)

For each formula, we could build a binary tree from bottom up. Each gate of the circuit has one or two inputs and one output. And the tree's depth is the formula's depth. For circuits, gates may have more than one output. We call the number of the inputs to a gate as the **fan-in** of the gate; the number of the outputs as the **fan-out** of the gate. If a formula or circuit only contains $\vee, \wedge$, we call it monotone formula (circuit) and it computes a monotone function.

Today we will prove that the **Circuit Sat** problem is NP-complete. First let us give its definition:

$\{C | C$ is a circuit on some number m of input variables $z_1, .., z_m$, s.t. $\exists$ a setting $z_i \to b_i \in \{0,1\}$, for which C output 1(true)$\}$

$Circuit - SAT$ is NP-complete wrt. $\leq_{logspace}$ reduction. In the following section, we will prove it.

9.4 Circuit SAT problem is NP Complete

Circuit problem was not the choice of Cook to show the first natural NP-complete problem. But proving NP hardness for this problem is somewhat easier than for the SAT probem.

Moreover,from the NP-complete of Circuit-SAT problem, it is easy to construct the following transformation.

$3 - SAT = \{f | f$ is a satisfiable 3CNF formula$\}$.

$Circuit - SAT \leq_{logspace} 3CNF - SAT$; also known as $3SAT$.

It follows that $SAT \leq 3SAT$. Note that this does not mean that for every formula F we can find an equivalent 3CNF formula F'! Rather we can transform F to a 3 CNF formula F' such that F is satisfiable iff F' is satisfiable. satisfiable iff SAT is.)

Theorem: Circuit-SAT is NP-complete

In the first step, we need to prove that $Circuit - SAT$ is in NP. This is obvious, since we just take the assignments to it as the certificate.

In the second step, we prove that every language $L \in NP$ can be transformed to $Circuit - SAT$ problem.

Let $L = \{x | \exists y, |y| \leq lg(|x|), \exists M : M(\langle x, y \rangle) = 1$ iff $R(x, y) = true\}$ M runs in time poly in $|\langle x, y \rangle| = poly(|x|)$(Since $y = poly(|x|)$.)

Let M is a one-tape machine with alphabet set Γ. We could construct a circuit as following to simulate how the machine works on input $\langle x, y \rangle$ within logspace reduction.

We have $x_1, x_2, ..., x_n, y_1, ..., y_{q(n)}$ as the input of the circuit. The depth of the circuit is $p(n)$ which is the running time of machine M. Each configuration of machine's computation corresponds to one level of the circuit. We can encode a configuration as a string of symbols of length $p(n)$ over the alphabet $\Gamma \bigcup (\Gamma \times Q)$. These symbols could then be be encoded in binary.

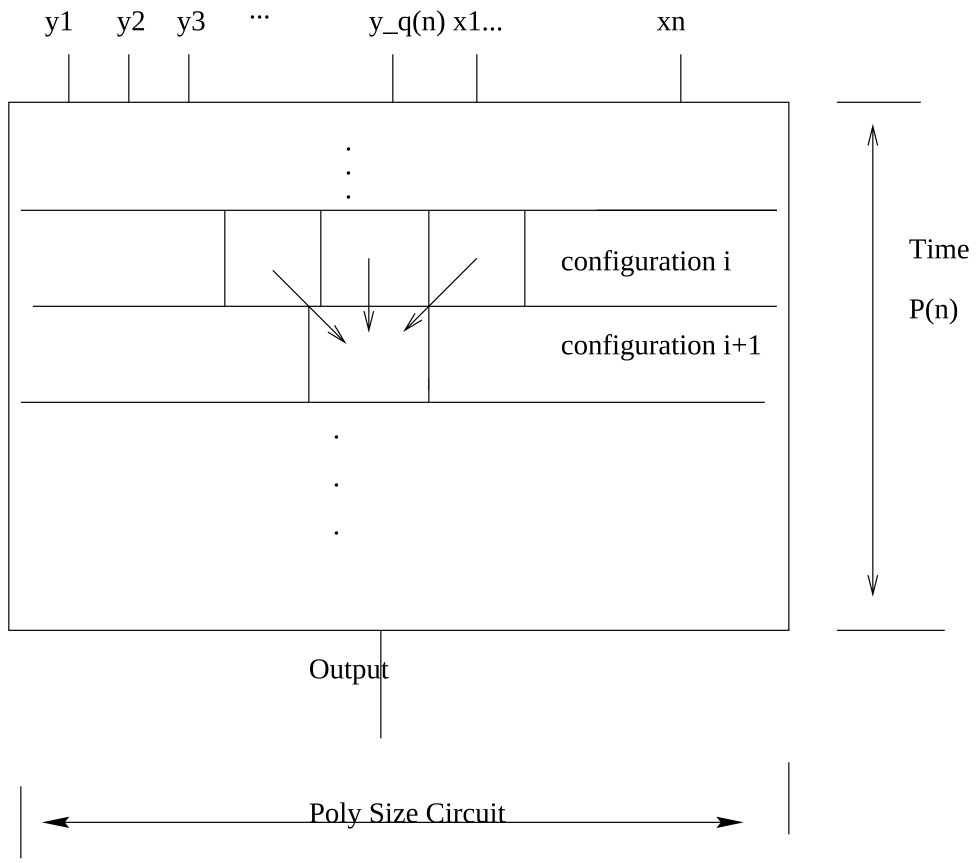

Figure 2: Simulating NP by Circuit SAT

Because of the very local nature of Turing machine computation steps, every symbol in level $i+1$ is determined by three symbols in level i.

It is easy to see that this transformation can be computed in $LogSpace$.