Theory of Computation (236343)

Eyal Kushilevitz

July 20, 1997

Lecture #1: March 2nd 1997

Introduction: The course is divided into two parts dealing with the following two fundamental questions:

1. What problems can be solved by computers ?
2. What problems can be solved *efficiently* by computers ?

We will give formal definitions, and will develop various techniques that will help us answering such questions in a mathematically sound manner.

Turing Machines: We define the model of Turing machines which will serve as our formal model of computation, as well as several related notions (such as configurations, the function f_M computed by a Turing machine M, etc.)

In the tutorial, examples will be given for how to program Turing machines.

Lecture #2: March 9th 1997

We define the notion of equivalence between two models of computation. We proved that the model of Turing machine (as defined in the previous lecture) is equivalent to the model of k-tape Turing machine.
In the next two tutorials, more examples will be given for models equivalent to Turing machines. In particular this will be shown for the model of Random Access Machine (RAM).

We then formulated *Church thesis*, that says that every "reasonable" model of computation is equivalent to the model of Turing machine.

Lecture #3: March 16th 1997

We define the classes $\mathcal{R}$ and $\mathcal{RE}$ and show some basic properties of these classes. For example, $\mathcal{R}$ is closed under complement, union, intersection, and concatenation; $\mathcal{RE}$ is closed under union.
We also define encodings of Turing machines and configurations into 0/1 strings. Using these encodings we show the existence of a universal Turing machine.
Finally, by a counting argument we show that there are languages not in $\mathcal{RE}$.

Lecture #4: March 26th 1997

We defined the languages $\mathcal{H}P$ ("halting problem") and L_u ("the universal language"). We proved that $\mathcal{H}P \in \mathcal{RE}$ but not in $\mathcal{R}$. We also proved that if a language L and its complement $\bar{L}$ are both in $\mathcal{RE}$ then $L \in \mathcal{R}$. This implies that $\overline{\mathcal{H}P} \notin \mathcal{RE}$. Similarly we showed that $L_u \in (\mathcal{RE} \setminus \mathcal{R})$ and $\overline{L_u} \notin \mathcal{RE}$.
By using these results we showed additional examples of languages not in $\mathcal{R}$. These include the languages L_{Σ^*} and L_{EQ}.

Lecture #5: March 30th 1997

We say that a language L_1 is reducible to a language L_2, and denote $L_1 \leq L_2$, if there exists a *computable function* $h : \Sigma^* \to \Sigma^*$ such that $x \in L_1 \Leftrightarrow h(x) \in L_2$. We proved basic properties of reductions such as:

$$L_2 \in \mathcal{R}, L_1 \leq L_2 \Rightarrow L_1 \in \mathcal{R}$$

$$L_2 \in \mathcal{RE}, L_1 \leq L_2 \Rightarrow L_1 \in \mathcal{RE}$$

and the transitivity of reductions. We then show how to use reductions to prove for certain languages L that $L \notin \mathcal{R}$. In particular we proved:

Rice Theorem: Let S be a non-trivial property of $\mathcal{RE}$ languages (i.e., there exist $L \in \mathcal{RE}$ that satisfies the property S and $L' \in \mathcal{RE}$ that does not satisfies S). Let $L_S \triangleq \{\langle M\rangle | L(M) \in S\}$. Then, $L_S \notin \mathcal{R}$.

Lecture #6: April 6th 1997

We surveyed the techniques by which we can prove that a language L is not in $\mathcal{RE}$. One possibility is to show that its complement $\bar{L}$ is in $\mathcal{RE} \setminus \mathcal{R}$ (we use this to prove that $L_\emptyset \notin \mathcal{RE}$). Another option is to use reductions from languages already known not to be in $\mathcal{RE}$ (we use this to demonstrate a language, L_{Σ^*}, such that both it and its complement are not in $\mathcal{RE}$).
We discussed the ways by which we can prove that functions (rather than languages) are not computable. For a given function f we defined a language $L_f = \{(x,y) | f(x) = y\}$ and proved that f is computable iff $L_f \in \mathcal{RE}$. Moreover, if f is defined for every x then f is computable iff $L_f \in \mathcal{R}$. This allows using the tools developed in the context of languages to prove impossibility of computing functions.

Lecture #7: April 13th 1997

Given a two argument relation $S \subseteq \Sigma^* \times \Sigma^*$ we defined the corresponding search problem (given x find y such that $(x,y) \in S$) and decision problem (given (x,y) decide whether $(x,y) \in S$). We proved that if the decision problem is in $\mathcal{RE}$ then the search problem is solvable, and that for certain relations the converse is false.

We defined the notion of *efficient* computation. Specifically we defined the class POLY as the class of functions that can be computed by a polynomial-time Turing machine and the class $\mathcal{P}$ as the class of languages that can be recognized by a polynomial-time Turing machine. We argued that the choice of *polynomial-time* as the definition for *efficiency* is a good one (yet not perfect) since it gives a good indication on what can be efficiently solved in reality and since it allows developing a "clean" theory.
We returned to the relationships between computing functions and recognizing languages. For a given function f we defined a language $L'_f = \{(x,y) | y \text{ is a prefix of } f(x)\}$ and proved that $f \in \text{POLY}$ iff $L'_f \in \mathcal{P}$.

Lecture #8: May 4th 1997

We started the discussion of efficient solutions for search problems. We stated the following major open problem: is it true for every two argument, polynomially-bounded relation $S \subseteq \Sigma^* \times \Sigma^*$ that if the decision problem of S is efficiently solvable then the search problem corresponding to S is also efficiently solvable.
We defined the class $\mathcal{NP}$ of all languages L for which there exists a polynomially-bounded, efficiently decidable relation R_L such that $L = \{x | \exists y, (x,y) \in R_L\}$. We proved that the above open problem can be stated equivalently as "Is $\mathcal{P} = \mathcal{NP}$?".
We defined the model of *nondeterministic* Turing machines and proved that this model is equivalent to the standard model of (deterministic) Turing machines.

Lecture #9: May 18th 1997

We gave an alternative definition for $\mathcal{NP}$ as the class of all languages that can be recognized by polynomial-time *nondeterministic* Turing machines. We proved the equivalence of the two definitions of $\mathcal{NP}$.
We defined the notion of *polynomial reductions* and presented simple properties of them (e.g., if $L_1 \leq_p L_2$ and $L_2 \in \mathcal{P}$ then $L_1 \in \mathcal{P}$; transitivity).
We defined the notion of $\mathcal{NP}$*-completeness* – L is $\mathcal{NP}$-complete if (1) $L \in \mathcal{NP}$ and (2) for every $L' \in \mathcal{NP}$, $L' \leq_p L$. We proved simple properties of $\mathcal{NP}$-completeness (e.g., if L is $\mathcal{NP}$-complete then $\mathcal{P} = \mathcal{NP}$ iff $L \in \mathcal{P}$; if $L \in \mathcal{NP}$ and for some $\mathcal{NP}$-complete L_2 we have $L_2 \leq_p L$ then L is $\mathcal{NP}$-complete). Finally, we proved the the *bounded halting language* is $\mathcal{NP}$-complete.

Lecture #10: May 25th 1997

In this class we proved the $\mathcal{NP}$-completeness of several languages. We started by stating Cook's theorem that L_{SAT} (satisfiability of CNF formulae) is $\mathcal{NP}$-complete (the proof will be given in the next class). We then used reductions to prove that all of the following languages are also $\mathcal{NP}$-complete: L_{3SAT} (satisfiability of 3CNF formulae), L_{VC} (Vertex Cover), L_{SC} (Set Cover), and L_{01IP} (0-1 Integer Programming).

Lecture #11: June 1st 1997

In this lecture we proved Cook's theorem (i.e., that L_{SAT} is $\mathcal{NP}$-complete). In contrast to L_{SAT} and L_{3SAT} we proved that L_{2SAT} is in $\mathcal{P}$.

Lecture #12: June 8th 1997

In this lecture we surveyed, mainly via examples, the possible approaches for coping with $\mathcal{NP}$-complete problems (and other problems not known to be in $\mathcal{P}$). These possibilities include (a) restricting the possible inputs; (b) allowing close-to-optimal solutions, when viewing the problems as optimization problems (so-called approximation algorithms; this option will be discussed in details in the next lecture); (c) average-case analysis of the running-time; and (d) probabilistic algorithms. Each of the possibilities may be useful for some problems and not for others.

Lecture #13: June 15th 1997

In this lecture we considered approximation algorithms for solving $\mathcal{NP}$-hard optimization problems. In particular we showed an algorithm for the minimal vertex-cover problem which finds a solution which is at most twice of the optimal, and an algorithm for the minimal set-cover problem which finds a solution which is at most $O(\log n)$ of the optimal. On the other hand, we showed that there are optimization problems that remain hard even if we allow close-to-optimal solutions.

Lecture #14: June 22nd 1997

In this lecture we introduced the notion of space complexity and in particular the class $\mathcal{PSPACE}$ of languages that can be recognized by polynomial-space Turing machines. We proved simple properties of this class (such as, $\mathcal{P} \subseteq \mathcal{PSPACE}$, $\mathcal{NP} \subseteq \mathcal{PSPACE}$, and $\mathcal{PSPACE} \subseteq \mathcal{R}$). Then, we defined the notion of $\mathcal{PSPACE}$-completeness and we proved that the language TQBF is $\mathcal{PSPACE}$-complete.