Basics Principles of Computational Complexity

Paul Sacawa

MAT 477

February 11, 2014

Basic Notions of Symbolic Representation

We represent by Σ some finite set of symbols, called an input alphabet.

For our purposes, $\Sigma := \{0, 1\}$. The elements of the alphabet Σ will be our means of symbolically representing data in our TMs. We denote by Σ^* the set of finite length strings $\gamma = \sigma_1 \ldots \sigma_n$ of symbols of Σ ($\sigma_k \in \Sigma$), with $|\gamma| := \#$ of symbols in γ. These strings are our input and output strings.

Symbolic Representation of Decision Problems

By a language we means a subset A of the set of all strings , $A \subset \Sigma^*$,

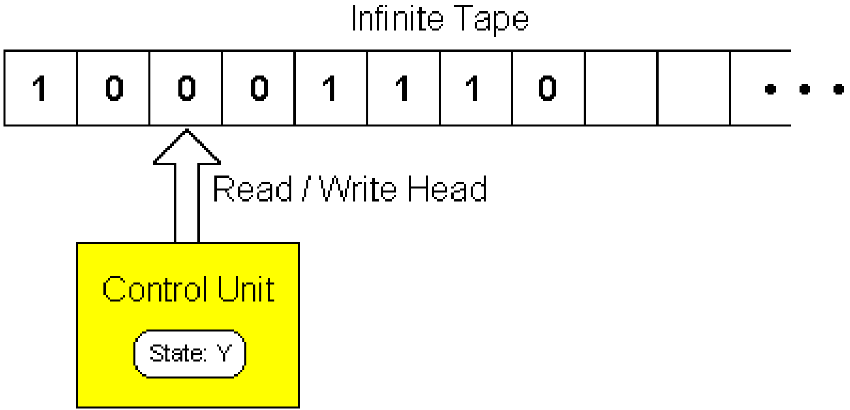
which we will think of as the computational problem of determining for

a given string $x \in \Sigma^*$, whether $x \in A$. For some property $\Psi(x)$ and

$A := \{x \in \Sigma^* : \Psi(x) \text{ holds}\}$, the problem of determining whether $x \in A$

is the problem of deciding whether x satisfies the property Ψ.

A Turing Machine will consist of a few components. We have an infinite sequence of cells where each cell contains a symbol of some alphabet (shortly, a memory tape). Here, the machine stores data. The tape has a head which at each step indicates one cell, which is the cell the machine is currently reading. We also have a set of states Q and a function δ telling the machine how to act at any step: Q and δ represent the 'code' of the machine.


Infinite Tape
1
0
0
0
1
1
1
0
Read / Write Head
Control Unit
State: Y

Our Computational Model: Turing Machines.

Def : A Turing Machine is a tuple $M := (Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}, \delta)$

containing the following data:

- a finite set of computation states $Q \supset \{q_0, q_{accept}, q_{reject}\}$;
- for us the input alphabet $\Sigma := \{0, 1\}$;
- for us the tape alphabet $\Gamma := \Sigma \cup \{\square\}$ consists of symbols that we allow to appear on the tape and $\square$ represents empty cells;
- a distinguished starting state $q_0 \in Q$;

- a distinguished accepting state $q_{accept} \in Q$ (when answer is "yes");
- a distinguished rejecting state $q_{reject} \in Q$ (when answer is "no");
- a transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{left, right, stay}\}$,

  shortly $dir := \{\text{left, right, stay}\}$.

Q and δ together form the program of M . To run M on input $x = x_1 \ldots x_r \in \Sigma^*$, we place x along the beginning of the tape, filling the rest of the tape with $\square$ characters. We then place the head at the first cell, set q_0 to be the state of the machine, and repeatedly follow δ :

If we are in state q , the symbol under the tape head is $s \in \Gamma$, and

$\delta(q, s) = (q', s', d')$, where $d' \in \text{dir}$. We then interpret this as an

instruction to rewrite the current symbol as s' , move to state q' , and

move the tape head in the direction dir. We repeat until we reach either

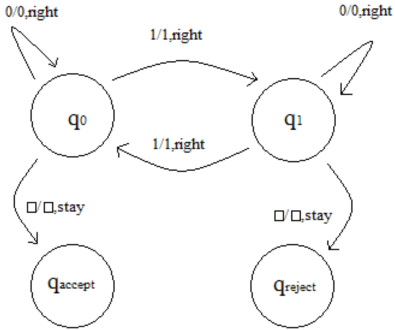
q_{accept} or q_{reject} , at which point we stop and say M has accepted or

rejected the input x based on the final state.

Def : Language $\mathcal{L}(M) := \{x \in \Sigma^* : M$ running on input x accepts$\}$.

Ex. We construct M s.th. $\mathcal{L}(M) = \{x \in \Sigma^*$: even number of 1's in $x\}$:

$Q = \{q_0, q_1, q_{acc}, q_{rej}\}$ and transitions δ are as below:



Turing Machines can compute functions $f : \Sigma^* \to \Sigma^*$. In this case, we run the TM in the exact same way, and when the computation ends, either accepting or rejecting, whichever is the case, we say the output $M(x)$ is the content of the tape up to the first $\Box$ character, i.e. is in Σ^*). The latter makes M to evaluate a function f on strings, i.e. $\forall x \in \Sigma^* : f(x) = M(x)$ hold.

Fact (Sanity Check). The decision problems $A \in \Sigma^*$ solvable in polytime by TMs and functions $f : \Sigma^* \to \Sigma^*$ computable in polytime by TMs are exactly those computable in polytime by programs in common computer languages. We therefore let ourselves think of TMs as programs written in a 'sane' programming language, and make arguments about what TMs can do without appealing to the formal definition.

Formal Definition of Complexity Class P .

For a $M \in TM$ and $x \in \Sigma^*$, we define

$t_M(n) := \max_{x \in \Sigma^*, |x|=n} \#$ steps M takes to compute with input x .

Def : For $f : \mathbb{N} \to \mathbb{N}$, let

$TIME(f) := \{A \subset \Sigma^* : \exists M \text{ s.th. } \mathcal{L}(M) = A \text{ and } t_M(n) = O(f(n))\}$.

Def : $\mathbf{P} := \bigcup_{k \in \mathbb{N}} TIME(n^k)$.

So, informally **P** is the class of problems solvable in polytime.

Formal Definition of Complexity Class NP .

NP contains languages A for which $x \in A$ can be proven in polytime,

in other words, the languages for which it is possible to find a polytime

$V \in TM$ s.th. $x \in A$ iff there is a polynomial length string $c \in \Sigma^*$

s.th. with input $\langle x, c \rangle$ our machine V accepts, i.e. the role of V is to

verify a potential certificate c of the fact that x is in A .

$$\mathbf{NP} := \left\{ \begin{array}{c} A \subset \Sigma^*\text{: } \exists \text{ polytime } V \in TM,\ k \in \mathbb{N} \text{ s.th.} \\ x \in A \iff \exists c \in \Sigma^* t : |c| \leq |x|^k \text{ and } V(x, c) \text{ accepts} \end{array} \right\} .$$

Def : SAT is the problem of determining, given a formula Φ built from

variables $Var = \{v_1, v_2, v_3 \ldots\}$ and connectives $\vee :=$ or, $\wedge :=$ and,

$\neg :=$ negation, if there is a truth assignment $\tau : Var \to \{True, False\}$

that makes $\Phi[\tau]$ true (shortly, 'satisfying' truth assignment). Formally,

$$\textbf{SAT} := \{\langle \Phi \rangle : \Phi \text{ is a satisfiable sentential formula}\} \ ,$$

where $\langle \Phi \rangle$ is a string in Σ^* representing here Φ , or later other data

SAT $\in$ **NP**: consider a machine V which takes a truth assignment τ of the variables of Φ and checks whether it makes $\Phi[\tau]$ true. It can be done in polytime (as on page 11). Formally, take as an input take as an input $x = \langle \Phi \rangle$ and a certificate $c = \langle \tau_\Phi \rangle$ representing some truth assignment of the variables of Φ . Then $V \langle \Phi, \tau \rangle$ shall verify whether $\Phi[\tau]$ is true and 'accept' only in that case.

$\langle \Phi \rangle \in \mathbf{SAT} \iff \exists \tau : \Phi(\tau) = \text{True} \iff \exists \tau : V \langle \Phi, \tau \rangle$ accepts.

Hierarchy of Complexity Classes: $\mathbf{P} \subset \mathbf{NP}$.

E.**g**. $\mathbf{P} \subset \mathbf{NP}$. For $A \in \mathbf{P}$, let M be the polytime TM with $\mathcal{L}(M) = A$,

and let simply define a verifier $V(x, y) = M(x)$. The algorithm of V is to

ignore the input y and just run $M(x)$ in polytime and return that result.

If M runs in $O(n^k)$ time, then so will $V(x, y)$, because it runs the same

algorithm, simply ignoring the certificate input y (and it is easy to

ignore the end of the input in $O(n)$ time).

Cook Reducibility: a 'Hardness' ordering on Languages.

Def : For languages $A, B \subset \Sigma^*$, write $A \leq_p B$ if there is a polytime computable function $f : \Sigma^* \to \Sigma^*$ such that

$$x \in A \iff f(x) \in B$$

We say solving A is reducible to solving B by means of f . As we will say, this shows that B is at least as hard as solving A , so "$\leq_p$" is an ordering by 'hardness' of solving up to polytime.

Lemma. "$\leq_p$" is a preorder. **Proof.** A simple and direct calculation.

Lemma. If $A \leq_p B$, and $B \in \mathbf{P}$, then also $A \in \mathbf{P}$.

Proof. Take the following polytime algorithm for A : given $x \in \Sigma^*$,

compute $f(x)$ in polytime and decide whether $f(x) \in B$, also computes

in polytime. Then our algorithm records respectively "yes" or "no".

Def : For a language $H \subset \Sigma^*$, say H is **NP**-hard if for all $A \in \mathbf{NP}$,

we have A is Cook-reducible to H : $\forall\, A \in \mathbf{NP}$ holds $A \leq_p H$.

So, informally C is as 'hard' as any problem of **NP** .

Def : For a language $C \subset \Sigma^*$, say C is **NP**-complete (shortly **NPC**)

if $C \in \mathbf{NP}$ and C is **NP**-hard, i.e. $\mathbf{NPC} := \mathbf{NP} \cap \mathbf{NP}$-hard .

MainTheorem (Cook, Lewin) (1971). There exists an **NP**-complete

problem and, moreover, **SAT** is **NPC** .

We saw already that $\mathbf{SAT} \in \mathbf{NP}$, so now for any $A \in \mathbf{NP}$, we

need a polytime machine M such that $x \in A \iff M(x) \in \mathbf{SAT}$.