Time and Space Classes and P
Exposition by William Gasarch

1 Deterministic Turing Machines

Turing machines are a model of computation. It is believed that anything that can be computed can be computed by a Turing Machine. The definition won't look like much, and won't be used much; however, it is good to have a rigorous definition to refer to.

Def 1.1 A *Turing Machine* is a tuple $(Q, \Sigma, \delta, s, h)$ where

- Q is a finite set of states. It has the states s, q_{acc}, q_{rej}.
- Σ is a finite alphabet. It contains the symbol #.
- $\delta : Q - \{q_{acc}, q_{rej}\} \times \Sigma \to Q \times \Sigma \cup \{R, L\}$
- $s \in Q$ is the start state, q_{acc} is the accept state, q_{rej} is the reject state.

We use the following convention:

1. On input $x \in \Sigma^*$, $x = x_1 \cdots x_n$, the machine starts with tape

$$\#x_1x_2 \cdots x_n\#\#\#\# \cdots$$

that is one way infinite.

2. The head is initially looking at the x_n.
3. If $\delta(q, \sigma) = (p, \tau)$ then the state changes from q to p and the symbol σ is overwritten with τ. The head does not move.
4. If $\delta(q, \sigma) = (p, L)$ then the state changes from q to p and the head moves Left one square. overwritten with τ. The head does not move. (Similar for $\delta(q, \sigma) = (p, R)$.
5. If the machine is in state h then it is DONE.
6. If the machine halts in state q_{acc} then we say M ACCEPTS x. If the machine halts in state q_{rej} then we say M REJECTS x.

Important Note: We can code Turing machines into numbers in many ways. The important think is that when we do this we can, given a number i, extract out which Turing Machine it corresponds to (if it does not correspond to one then we just say its the machine that halts in one step on any input). Hence we can (and will) say things like

- Let $M_1, M_2, M_3, \ldots$ be a list of all Turing Machines.
- Run $M_i(x)$. This is easy- given i, we can find M_i, — that is, find the code for it, and then run it on x.

Def 1.2 A set A is DECIDABLE if there is a Turing Machine M such that

$$x \in A \rightarrow M(x)\text{ACCEPTS}$$

$$x \notin A \rightarrow M(x)\text{REJECTS}$$

2 Time and Space Classes

Def 2.1 Let $T(n)$ be a computable function (think of it as increasing). A is in $DTIME(T(n))$ if there is a MULTITAPE TM M that decides A and also, for all x, $M(x)$ halts in time $\leq T(|x|)$. *Convention:* By $DTIME(T(n))$ we really mean $DTIME(O(T(n))$. They are actually equivalent by having your TM just take bigger steps.

Note that this is unfortunately machine dependent. It is possible that if we allow 2-tapes instead of one it would change how much you can do. We won't have to deal with this much since we will usually define classes in terms of multi-tape machines, and we will allow some slack on the time bound, like: $DTIME(n^{O(1)})$.

It is known that a Multitape $DTIME(T(n))$ machine can be simulated by (1) a 1-tape $DTIME(T(n)^2)$ TM, and also0 (2) a 2-tape $DTIME(T(n)\log T(n))$ TM.

Def 2.2 Let $S(n)$ be a computable function (think of it as increasing). A is in $DSPACE(S(n))$ if there is a TM M that decides A and also, for all x, $M(x)$ only uses space $S(|x|)$. *Convention:* By $DSPACE(S(n))$ we really mean $DSPACE(O(S(n))$. They are actually equivalent by having your TM just take bigger steps. *Convention:* When dealing with space classes we will have an input tape which is read-only and a separate worktape. When dealing with space-bounded TMs computing functions we will also have a write-only output tape.

It is known that a Multitape $DSPACE(S(n))$ machine can be simulated by a 1-tape $DSPACE(S(n))$ TM.

Def 2.3 Let $T(n)$ be a computable function (think of it as increasing). A is in $NTIME(T(n))$ if there is a Nondet TM M that decides A and also, for all x, $M(x)$, on any path, halts in time $\leq T(|x|)$. *Convention:* By $NTIME(T(n))$ we really mean $NTIME(O(T(n))$. They are actually equivalent by having your TM just take bigger steps.

Def 2.4 Let $S(n)$ be a computable function (think of it as increasing). A is in $NSPACE(S(n))$ if there is a Nondet TM M that decides A and also, for all x, $M(x)$, on any path, only uses space $\leq S(|x|)$. *Convention:* By $NSPACE(S(n))$ we really mean $NSPACE(O(S(n))$. They are actually equivalent by having your TM just take bigger steps.

Def 2.5 For all of the definitions below, 1-tape and multitape are equivalent. This is important in the proof of the Cook-Levin theorem and later in the proof that a particular lang is EXPSPACE complete and hence not in P.

1. $P = DTIME(n^{O(1)})$.
2. $NP = NTIME(n^{O(1)})$. This is equivalent of just using 1-tape TM's. (This is equivalent to our quantifier definition.)
3. $EXP = DTIME(2^{n^{O(1)}})$.
4. $NEXP = NTIME(2^{n^{O(1)}})$.
5. $L = DSPACE(O(\log n))$.
6. $NL = NSPACE(O(\log n))$.
7. $PSPACE = DSPACE(n^{O(1)})$.
8. $EXPSPACE = DSPACE(2^{n^{O(1)}})$.
9. $NEXPSPACE = NSPACE(2^{n^{O(1)}})$.

3 Decidable Sets that are NOT in P

Are there any decidable sets that are NOT in P? Yes. We prove that here.

We first need a representation of P.

Let $M_1, M_2, \ldots$ be the set of all Turing Machines Recall that we code TM's in to numbers such that, given i, one can actually run M_i. We can also assume that every TM appears on the list infinitely often since there can always be dummy states. We will also assume that these are YES-NO machines, that is, if they stop then they have either YES or NO written on the output tape.

Let $N_1, N_2, \ldots$ be defined as follows:

$N_i(x)$ runs $M_i(x)$ for $|x|^i$ steps (it may stop before then). If at this point it has stopped and said YES or stopped and said NO then thats fine, no need to change it. If $M_i(x)$ has not stopped within $|x|^i$ steps then we have $N_i(x)$ write NO.

KEY: If a language L is in P then there is some N_i such that N_i decides L.

We now give an algorithm that will define a language that is NO in P.

1. Input(x). If $x \notin 0^*$ then OUTPUT NO AND STOP.

2. (Can assume $x = 0^i$ for some i.) Run $M_i(0^i)$. (Note that this is guaranteed to halt within n^i steps.) Let b be the output (so b is either YES or NO)

3. If $b = YES$ then output NO and halt. If $b = NO$ then output YES and halt.

Let L be the set of strings that this algorithm says YES on. This is clearly decidable.

We claim that L is not in P. Assume $L \in P$. Then there is some i such that N_i decides L. But L and N_i differ on the input 0^i. Hence L is not decided by N_i. Contradiction.