Topics for the Final Examination

- Design of automata and grammars; regular expressions
- Computability problems
- Decision problems
- Undecidability via reduction
- Recursive and Recursively Enumerable sets
- P and NP algorithms
- NP-completeness via Polynomial Time Computable Reduction
- **No pumping lemmas, no congruence lemmas, and no diagonalizations. You will be asked to prove undecidability via reduction and NP-completeness via polynomial time reduction.**

III. Prove that the following problem is computable.

$$f(x,y) = \begin{cases} 1 & \text{if there exist TMs } M_1, M_2 \text{ s.t. } x = [M_1],\ y = [M_2], \text{ and } L(M_1) \cap L(M_2) \neq \phi \\ \textit{undefined} & \text{otherwise} \end{cases}$$

check whether x & y are codes of TMs. If not go into a loop, ie. $f(x,y)$ is undefined

If so, let $x = [M_1]$ & $y = [M_2]$.

At any stage keep track of the ~~~~ sets of strings accepted by M_1 & M_2 upto-that instant. Let they be L_1 & L_2. Initially $L_1 = L_2 = \phi$.

At any stage i, bring in simulation of M_1 & M_2 on input i (as a binary string). Simulate all the pending processes by one additional step. If an M_i accepts an ~~~~ input j, include j in L_i. Remove (M_i, j) simulation process. Then check whether j is in the current L_{3-i}. If so, output 1 and halt.

Else, ~~~~ go to the next stage.

IV. Given TMs $M_1, M_2, \cdots, M_n$, prove that the following set is recursively enumerable:
$\{x \mid$ every M_i halts on input $x\}$.

Function f keeps track of the number of TMs that have so far halted on any particular input. For example, if $f(x) = k$ then at that stage k of the n TMs have halted on input x.

At any stage i; start simulations of $M_1, M_2, \ldots, M_n$ on input i as separate processes & set $f(i) = 0$. Simulate all the running processes by one more step. If any M_j halts on input l then remove this process & increment $f(l)$ by 1. Then check whether $f(l) = n$. If so output l.

Go to stage $i+1$.

VII. **Design a P algorithm for the following problem. Also, estimate the speed of the algorithm.**

Given a digraph G, a vertex u, and a value k, are there k distinct vertices $v_1, v_2, \cdots, v_k$ s.t. for every $i \in \{1, \cdots, k\}$, there is a path from u to v_i and there is a path from v_i to u?

A similar problem is done in class this year (2013).

VIII. **Design an NP algorithm for the following problem. Prove its correctness and estimate its speed.**

Given n digraphs, $G_1, G_2, \cdots, G_n$, each having n vertices, does there exist a $k \geq n/2$ s.t. at least $n/2$ of the digraphs contain a simple cycle of length k?

Guess $i_1, \cdots, i_k$. verify $i_1 < i_2 < \cdots < i_k$ & $k \geq \frac{n}{2}$.
For each i_j guess k vertices $v_{j_1}, \cdots, v_{j_K}$. Verify that $v_{j_1}, \cdots, v_{j_K}$ form a simple cycle in G_{i_j}.

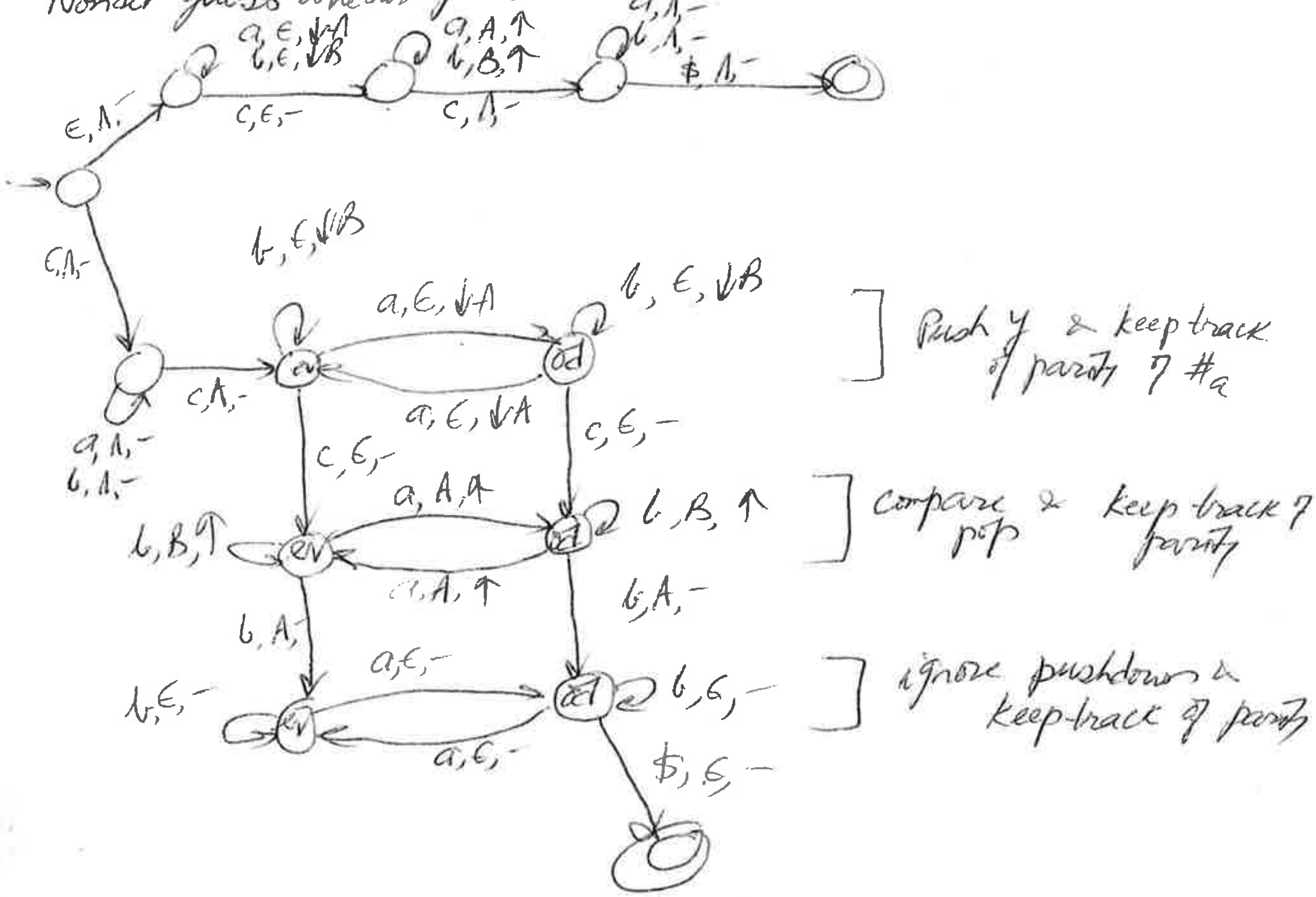
Correctness: easy.
Speed: Testing the distinctness of $v_{j_1}, \cdots, v_{j_K}$ crudely requires $O(k^2)$ steps $= O(n^2)$ steps. Overall $O(n^3)$.

600.271 Automata & Computation Theory
Final Examination
May 10, 2012
In-class, Closed Book
Time: 2 hrs 30 mins.

All the subproblems carry equal weight.

I. Design a nondeterministic pda for the language:
$\{xcycz \mid x, y, z \in \{a,b\}^*, y = x^R \text{ or } (z \neq y^R \text{ and } \#_a yz \text{ is an odd integer})\}$.

Nondet guess whether $y = x^R$ or the 2nd condition holds & verify.



III. Prove that the following problem is decidable.

1. Given $[M_1]$ and $[M_2]$, M_1 a dfa and M_2 a dlba, is $L(M_1)$ a finite set and $L(M_1) \cap L(M_2) \neq \phi$?

Testing $L(M_1)$ is a finite set.

~~Minimizing the number of states of M_1. The only property we need is that "useless" states~~

Let M_1 have s_1 states.

Claim: $L(M_1)$ is finite iff M_1 doesn't accept any string of length in the range $[s_1, 2s_1]$.

Now generate all strings of length s_1, then s_1+1, then s_1+2, ..., then $2s_1$ & check whether at least one of them is accepted by M_1. If so $L(M_1)$ is not finite. If not, $L(M_1)$ is finite.

Knowing $L(M_1)$ is finite, to test $L(M_1) \cap L(M_2) \neq \phi$.

Let M_2 have s states and t tape symbols. We have shown that if M_2 accepts a string of length n, then it must accept within snt^n steps. So any given string can be tested for acceptance by M_2.

Now we generate ϵ, all strings of length 1, then length 2, then length ..., then length s_1. For each string, test whether it is accepted by both M_1 & M_2. If there is such a string, output 'yes', else output 'no' & halt.

V. Prove the undecidability of the following problem.

1. Given $[M_1], [M_2], [M_3]$, do there exist x_1, x_2, x_3 such that $x_1 < x_2 < x_3$ and for $i = 1, 2, 3$, Turing machine M_i halts on input x_i? (Hint: Make use of the undecidability of the blank tape halting problem.)

BTHP $\leq_m$ this problem

Typical ind: $[M]$ $\quad$ $[M_1], [M_2], [M_3]$

Given $[M]$, we ~~transform~~ transform it to $[M_1], [M_2], [M_3]$ s.t. TM M halts on BT iff there exist x_1, x_2, x_3 s.t. $x_1 < x_2 < x_3$ & TM M_i halts on x_i.

~~$[M]$~~ We choose $x_1 = 1$, $x_2 = 11$, $x_3 = 111$

TM M_1: halts on input 1, & doesn't halt on any other input.

TM M_2: halts on input 11 & doesn't halt on any other input

TM M_3: doesn't halt on any input other than 111. On input 111, it erases the input & simulates M on blank tape.

Note that our goal is achieved.

Also, the transformation from $[M]$ to $[M_1], [M_2], [M_3]$ is computable.

VII. Design a P algorithm for one of the following problems. Estimate its speed.

1. Given a directed graph G with n vertices, is there an ordering of its vertices into $v_{k_1}, v_{k_2}, \cdots, v_{k_n}$ such that for every $1 \leq i < j \leq n$ there is an edge from v_{k_i} to v_{k_j}?

2. Given an nfa M and a string x, is $x \in L(M)$? (Conversion of M into dfa is not possible since this can take an exponential number of steps.)

1. Let the adjacency matrix be A.
Find an i s.t. the ith row is all 1's except $A(i,i)$.
Make i to be k_1. Eliminate ith row & ith column.
Repeat the process to find $k_2, \ldots, k_n$.
__Crude implementation__: Each iteration requires scanning all the entries of A once; i.e. $O(n^2)$ steps. This is done $O(n)$ times. Overall algo: $O(n^3)$ steps.

2. Let $x = a_1 a_2 \ldots a_n$.
For each i, we compute ~~the~~ S_i, the set of states M can reach on input $a_1 a_2 \ldots a_i$.
Start with $S_0 = \{q_0\}$.
Computation of S_{i+1}: For each $q \in S_i$, place all states from q on input a_{i+1}

Remove duplicates.
At the end check whether S_n contains a final state.
__Speed__: $|S_i| \leq n$. From each state in S_i on input a_{i+1}, M can reach at most n states. So the number of steps needed to compute $S_{i+1} = O(n^2)$.
Overall $O(n^3)$.