Complexity, 2003-2004
James Lawson

1.(a) i. There is a nondeterministic TM, N, that decides L in polynomial time.

ii. $L_1 \leqslant L_2$ iff there is a map f such that, (i) $x \in L_1$ iff $f(x) \in L_2$ and (ii) f is a p-time computable function.

iii. L is NP-complete iff (i) L is NP-hard: if L' in NP then $L' \leqslant L$ (ii) L in NP.

iv. To show HC is NP-complete , consider a reduction HP $\leqslant$ HC.

$$f((V,E)) = \begin{cases} \bullet & \text{if } |V| = 1 \\ (V \cup \{\mathsf{p}\}, E \cup \{(u,\mathsf{p}) \mid u \in V\}) & \text{otherwise} \\ \quad \mathsf{p} \text{ not in } V & \end{cases}$$

To show: $G \in$ HP iff $f(G) \in$ HC.

- ($\Rightarrow$) Assume G has a HP. If $|V| = 1$, then $f(G) = \bullet$ which trivially has a HC. Otherwise, G has some HP $x..y$. $f(G)$ has HC p$x..y$p.
- ($\Leftarrow$) Assume $f(G) = (V', E')$ has a HC. If $|V'| = 1$, only first case matches, $|V| = 1$, so G trivially has a HP. Otherwise second case. $|V'| \geqslant 3$ and $f(G)$ has p. So $f(G)$ has some HC of the form p$x..y$p, so G must have HP $x..y$.

We can compute f in p-time (reading/writing to a adjacency matrix polynomial number of times) so f is a map that gives us HP $\leqslant$ HC. So HC is NP-hard. But HC is in NP so, we can guess string and check in polynomial time that it is indeed a valid HC. Hence HC is NP-complete.

(b) i. L is *Cook Reducible* to L' iff there is a p-time algorithm which decides L using an oracle for L'.

ii. Assume GISOM $\in$ P.
Consider a Cook Reduction from GAUTO to GISOM. We show there is a p-time algorithm which decides GAUTO using a GISOM oracle.
Algorithm:
For each node in G, delete node. call this graph G'. Ask the GISOM if there is an isomorphism from G' to G'.

2.(a) i. We can guess a path and check in logspace that is indeed a path from x to y. We use a counter to store the current node and another counter to count the number of nodes we have checked so far. For the current node, check that the next node in the path is adjacent to x'. Repeat up to n. If counter reaches n and we haven't reached y, halt and fail. If we reach y within n steps, succeed.

Each counter requires $\log(n)$ space and there are a fixed number of counters (independent of input size). We also only *read* from the input (read the path, read adjacency matrix) which has no charge in terms of space. This algorithm shows RCH is in NL.

ii. RCH is NL-complete. If RCH were in L, then this would imply NL = L which is believed to be unlikely.

(b) A k-tape TM with input-output has at most $C = Q.n.(|\Sigma|^{f(n)}.f(n))^{k-2}.2$ configs. We can apply the *reachability method*. For any machine in NSPACE, we can construct a machine M' that on input x, builds config graph $G(M,x)$ then runs RCH in $O(C^2)$ time, returning yes iff there is a path from init config to a yes state. This gives, $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{TIME}(k^{\log(n)+f(n)})$ when f is proper.

Let $f(n) = \log(n)$. Since $\log(n)$ is a proper function, we can apply the previous result. $k^{\log(n)+f(n)} = k^{\log(n)+\log(n)} = c^{\log(n)} = n^{c'}$.
Hence $\mathsf{NSPACE}(\log(n)) \subseteq \mathsf{TIME}(n^{c'})$, i.e. NL $\subseteq$ P.

(c) *Algorithm*: Use one counter $s = 0$ (to store the input's size). Read input tape from left to right while incrementing s until we reach the end.
If s is odd, then halt and fail.
Otherwise. Loop using two counters $l = 0$ and $r = s/2$.
Read the lth symbol and rth symbol of the input, if they are not equal, halt and fail. Otherwise increment l and r and repeat comparison.
Repeat loop until $l = s/2$. If l reaches $s/2$, Halt and succeed.

Reasoning: Reading from the tape has no charge. We use a fixed number of counters: s, l, r whose values are bounded by $\log(n)$. We can use a fixed number of work tapes for the comparison whose values are bounded by $\log(n)$. (eg use 2 tapes of size 1). Overall we use space $\leqslant m\log(n)$ where m is independent of input size, so this algorithm decides DOUBLE in logspace, hence DOUBLE $\in$ L.

(d) *Algorithm*: Guess some split $w_1 \cdot w_2 \cdot w_3 \cdot \ldots \cdot w_n$ where $w_i \in \{0,1\}^*$
Use one counter to store n. Use two counters i, j.
i points to beginning w_1 and j points to beginning of w_2.
Check w_1, w_2 in L by running $M(w_1), M(w_2)$ respectively.
If either $M(w_1), M(w_2)$ return no, halt and fail.
Compare w_1 with w_2 and check they are equal.
Now repeat, by incrementing i and j and checking w_2, w_3, instead.
Repeat until i reaches n (w_{n-1}, w_n checked). Halt and succeed.
Reasoning: To compare w_i and w_j, we use can one tape with 1 symbol We have a fixed number of counters bounded by $\log(n)$ and running $M(w_1), M(w_2)$ can be done in logspace. Hence this algorithm describes a nondeterministic machine that decides L^* in logspace. Hence $L^* \in$ NL.

3.(a) i. Use n^3 processors, and each calculates $A_{ik}B_{kj}$ in one step. Then use n^2 of processors to find sum $S_{ij} = \sum_k A_{ik}B_{kj}$. When computing S_{ij}, each processor will parallelise addition by using a further n processors. Create a knockout-tournament tree of additions of depth $O(\log(n))$. Overall time $= 1$ step for mult. $+ O(\log(n))$ steps for additions $= O(\log(n))$

ii. Take the adjacency matrix for A and square A successively $\lceil \log(n) \rceil$ times. This gives the transitive closure for A. By using the previous matrix mult. algorithm, we can square a matrix in $O(\log(n))$ steps. Overall time is num of times we square $\times$ steps for squaring $= O(\log^2(n))$.

(b) Create knockout tournament. Each match calculates total 0s and total 1s by performing parallel addition of the result of two matches from the previous parallel round, $(x+x', y+y')$. The final summation has some total (x,y) and we return yes iff $x = y$. Total work: $2(n-1)$. Parallel time: $O(\log(n))$

(c) i. L is in NC_j iff there is a logspace uniform family of circuits $\mathcal{C}$ that decide L in parallel time $O(\log^j(n))$ with $O(n^k)$ work.

ii. Because $L \in NC_j$:

- from family $\mathcal{C}$, there is a circuit C_1 that decides if $x_1 \in L$, a circuit C_2 that decides if $x_1x_2 \in L$, a circuit C_3 that decides if $x_1x_2x_3 \in L$, ... a circuit C_n that decides if $x_1x_2x_3...x_n \in L$. We perform $\wedge$ on all the outputs of C_1, C_2, ..., C_n and the result outputs tt iff for all $y \leq x$, $y \in L$. In other words, C'_n decides L'.
- C_1, C_2, ..., C_n have depths bounded by $O(\log^j(n))$. When we perform $\wedge$, we can connect them using a tournament-style knockout tree that has a depth of $\lceil \log(n) \rceil$. Overall, C'_n has depth $O(\log^j(n)) + \lceil \log(n) \rceil = O(\log^j(n))$
- C_1, C_2, ..., C_n have sizes bounded by $O(n^k)$. We use $n-1$ $\wedge$-gates for our knockout tree. Overall, C'_n has size: $nO(n^k) + n - 1 = O(n^{k'})$ for some k'.

Since L' is in $PT/WK(\log^j n, n^{k'})$, we have $L' \in NC_j$.

4.(a) i. RP is the class of languages with p-time Monte Carlo TMs (if $x \in L$ then TM outputs yes with probability $> \frac{1}{2}$, if $x \notin L$ then TM definitely outputs no).

ii. Suppose that $L_1, L_2 \in$ RP and are recognised by Monte Carlo machines M_1, M_2 respectively. We describe a machine M for $L_1 \cap L_2$. Run M_1 followed by M_2 and accept iff both M_1 and M_1 accepts. Clearly M is can be represented as a precise NDTM with polybound $p_1 + p_2$.
Let $\frac{1}{2} + \varepsilon_1$, $\frac{1}{2} + \varepsilon_2$ be probabilities of success, when $x \in L_1$, $x \in L_2$, for M_1, M_2 respectively, $\varepsilon_1, \varepsilon_2 > 0$. Then probability of success when $x \in$

L is $(\frac{1}{2}+\varepsilon_1)2^{p_1} \times (\frac{1}{2}+\varepsilon_2)2^{p_2} = (\frac{1}{2}+\varepsilon_1)(\frac{1}{2}+\varepsilon_2)2^{p_1+p_2} = \varepsilon 2^{p_1+p_2}$, for some $\varepsilon > 0$. Suppose we needed to run M, m times until a successful computation. Then $(1-(1-\varepsilon)^m)2^{p_1+p_2}$ of computations succeed. Then let M run m times such that $(1-(1-\varepsilon)^m) > \frac{1}{2}$. Then we have that if $x \in L$, then M succeeds with prob. $> \frac{1}{2}$.

(b) i. SAT: given a set of clauses, is there a satisfying assignment that makes each clause evaluate to true
FSAT: given a set of clauses, find a satisfying assignment that makes each clause evaluate to true

ii. Let $\varphi(x_1, x_2, x_3, ..., x_n)$ be our formula in CNF.
We can solve FSAT by following alg:
Algorithm:
Ask oracle if φ is sat. If no halt. If yes continue.
Ask oracle if $\varphi(\mathsf{tt}, x_2, x_3, ..., x_n)$ is sat. If yes, assign $h(x_1) = \mathsf{tt}$. If no $h(x_1) = \mathsf{ff}$. Ask oracle if $\varphi(h(x_1), \mathsf{tt}, x_3, ..., x_n)$ is sat. If yes assign $h(x_2) = \mathsf{tt}$. If no $h(x_2) = \mathsf{ff}$. Repeat until all variables assigned. We only use oracle $n+1$ times (polynomial times) so we have computed FSAT in p-time.

iii. F3COL: Given a graph G, find an assignment of colours to the nodes of G in such a way that no two adjacent nodes have the same colour, and no more than three colours are used.
Let $G(v_1, v_2, v_3, ..., v_n)$ be our graph.
Given a F3COL oracle, we can solve F3COL by following alg:
Fix three colours R, G, B. Ask oracle if $G(v_1, v_2, v_3, ..., v_n)$ is 3-colourable. If no halt. If yes continue.

Modify G and add e RGB K_3 cliques, where e is the number of edges in G. Let $G(\mathsf{R}, v_2, v_3, ..., v_n)$ denote modifying G (with cliques) where, for each edge, e, that v_1 connects with, add two new edges to nodes G and B in corresponding RGB-clique for e. Similarly, let $G(\mathsf{R}, \mathsf{G}, v_3, ..., v_n)$ denote G_1 with but adding for each edge, e, that v_2 connects with add two new edges to nodes R and B in corresponding RGB-clique for e. And so on, for n colour assignments.
In general let $G(..., \mathsf{C}_i, ...)$ where $\mathsf{C}_i \in \{\mathsf{R}, \mathsf{G}, \mathsf{B}\}$ denote, taking the previous graph and modifying it so that for each edge, e, that v_i connects with, we add two new edges to nodes $\neq \mathsf{C}_i$ in corresponding RGB-clique for e.

Let χ be our colouring. Set $\chi(v_1) = \mathsf{R}$.
Ask oracle if $G(\mathsf{R}, \mathsf{R}, v_3, ..., v_n)$ is 3-colourable.
If yes, then set $\chi(v_2) = \mathsf{R}$. Else ask oracle if $G(\mathsf{R}, \mathsf{G}, v_3, ..., v_n)$ is 3-colourable.

If yes, then set $\chi(v_2) = \mathsf{G}$, else $\chi(v_3) = \mathsf{B}$.

Ask oracle if $G(\mathsf{R}, \chi(v_2), \mathsf{R}, ..., v_n)$ is 3-colourable.
If yes, then set $\chi(v_3) = \mathsf{R}$ Else ask oracle if $G(\mathsf{R}, \chi(v_2), \mathsf{G}, ..., v_n)$ is 3-colourable.
If yes, then set $\chi(v_3) = \mathsf{G}$, else $\chi(v_3) = \mathsf{B}$.

Ask oracle if $G(\mathsf{R}, \chi(v_2), \chi(v_3), ..., \mathsf{R})$ is 3-colourable.
If yes, then set $\chi(v_n) = \mathsf{R}$, Else, ask oracle if $G(\mathsf{R}, \chi(v_2), \chi(v_3), ..., \mathsf{G})$ is 3-colourable.
If yes, then set $\chi(v_n) = \mathsf{G}$, else $\chi(v_n) = \mathsf{B}$.
We ask the oracle $O(n)$ times and graph modifications can be done in p-time. Hence we have a p-time algorithm.