Complexity, 2005-2006
James Lawson

1. (a) i. NPC the class of langauges that are in NP and are NP-hard

   ii. SAT: given a set of clauses, is there a satisfying assignment
   HP: given a graph, is there a hamiltonian path
   TSP(D): given a graph, is there a cycle that visits all nodes within budget b

   iii. Assume $\mathsf{NPC} \cap \mathsf{P} \neq \emptyset$
   Then there is some langauge L_0 that is NP-complete and in P.
   We already know that $\mathsf{P} \subseteq \mathsf{NP}$, so we need to show $\mathsf{NP} \subseteq \mathsf{P}$.
   Take some language $L \in \mathsf{NP}$.
   Since L_0 is NPC, we have $L \leqslant L_0$
   But since $L_0 \in \mathsf{P}$, we have $L \in \mathsf{P}$ (P is downwards-closed).
   Hence $\mathsf{NP} \subseteq \mathsf{P}$. So $\mathsf{P} = \mathsf{NP}$.

   (b)
   - We can find a reduction from HP to AHP.
   Let $f(G)$ add twos new nodes, x and p. Add edge (x, p) and edges (p, u) for all u in original G. Then G has HC iff $f(G)$ has HP starting at x.
   $(\Rightarrow)$ If G has HP $u..v$, then $f(G)$ has HP starting at x: $x, p, u...v$.
   $(\Leftarrow)$ If $f(G)$ has HP, it must start/end at x, because x is the only node with degree 1, so and path must go through p at the start/end to reach x. $u..vpx$. So G has HP $u...v$.
   If G has any degree 0 nodes, then it returns some no-instance.
   f can be computed in p-time and so AHP is NPC.
   - We can find a reduction from HP to THP.
   Let $f(G)$ add four new nodes, p, q, r, s.
   Add edge (p, q) and edges (q, u) for all u in original G.
   Add edge (r, s) and edges (u, r) for all u in original G. Label some node in the old graph as x. If G has any degree 0 nodes, then it returns some no-instance.
   Then G has HP iff $f(G)$ has HP starting that doesn't start or end at x.
   $(\Rightarrow)$ If G has HP $u..v$, then $f(G)$ does indeed have a HP $p, q, u...v, r, s$. And, any HP in $f(G)$ must start/end with p, s, and so cannot start/end with the node labelled x from the original graph.
   $(\Leftarrow)$ If $f(G)$ has HP, it must start/end at p/s, because these are is the only nodes in $f(G)$ with degree 1. So HP has the form $p, q, u...v, r, s$, and so G has HP $u...v$. f can be computed in p-time and so THP is NPC.

   (c) i. 2SAT: Given a set of clauses, each with 2 literals, is there a satisfying assignment?

ii. We can show that 2SAT is in P. Suppose that $2SAT \in \mathsf{NPC}$. We have some language in both NPC and P, so $\mathsf{NPC} \cap \mathsf{P} \neq \emptyset$, so by (a)(iii), we have $\mathsf{P} = \mathsf{NP}$. Contradicting our assumption that $\mathsf{P} \neq \mathsf{NP}$. So 2SAT cannot be NP-complete under this assumption.

iii.
- Add nodes: p, q, r and $\neg p, \neg q, \neg r$.
  Add edges:
  $\neg p \to q$, $\neg q \to p$
  $q \to r$, $\neg r \to \neg q$
  $q \to \neg r$, $r \to \neg q$
  $r \to \neg r$
  Take p. Check if there is a path from p to $\neg p$. There is not. So $v(p) = \mathtt{tt}$. Take q. Check if there is a path from q to $\neg p$. There is. check if path from $\neg p$ to p. There is not. So $v(q) = \mathsf{ff}$. Take r. check if there is a path from r to $\neg r$. There is. Check if path from $\neg r$ to r. There is not. So $v(r) = \mathsf{ff}$.
  Hence a satisfying assignment is $v(p) = \mathtt{tt}, v(q) = \mathsf{ff}, v(r) = \mathsf{ff}$.
- Add nodes: p, q, r and $\neg p, \neg q, \neg r$.
  Add edges:
  $p \to q$, $\neg q \to \neg p$
  $p \to \neg q$, $q \to \neg p$
  $\neg p \to r$, $\neg r \to p$
  $r \to \neg r$
  $\neg r \to \neg q$, $q \to r$
  $\neg q \to p$, $\neg p \to q$
  Take p. Check if there is a path from p to $\neg p$. There is. Check if there is a path from $\neg p$ to p. There is. We have a cycle from p to $\neg p$, so there is no satisfying assignment.

2. (a) i.
- $\mathsf{LOGSPACE}$: The set of languages that can decided by a k-tape input-output turing machine in logspace.
- log-space reduction: $A \leqslant_{\log} B$ iff there is a map f such that $x \in A$ iff $f(x) \in B$ and f is computable in logspace.
- NL-complete: A language, L is nl-complete iff it is can be decided by a nondeterministic IOTM in logspace and has the property that for all L', if $L' \in \mathsf{NL}$ then $L' \leqslant_{\log} L$.

ii. Savitch's theorem: RCH can be decided in $O(\log(n)^2)$ space.
Consequence: $\mathsf{NL} \subseteq \mathsf{SPACE}(\log^2(n))$.

(b) Consider the reduction $\textsc{Rch} \leqslant_{\log} \textsc{DConn}$. Let

$$f((G,x,y)) = (V, E \cup \{(u,v) \mid v = x \vee u = y\}) \text{ where } G = (V,E)$$

To show $(G,x,y) \in$ RCH iff $f((G,x,y)) \in$ DCONN

- ($\Rightarrow$) Assume (G,x,y) in RCH.
  Then there is path $x \rightsquigarrow y$ in G', and in G' (f doesn't remove edges).
  Take $u, v \in G'$. There is edge $(u,x) \in E'$. and edge (y,v). Hence there is a path $u, x \rightsquigarrow y, v$ in G'. Any two nodes u, v have a path connecting, so G' is connected.
- ($\Leftarrow$) Assume $f((G,x,y))$ in DCONN.
  Since $f(G)$ is connected, every pair of nodes, u, v has path $u \rightsquigarrow v$. So there is path $x \rightsquigarrow y$ in G'. But, all edges in this path are also in G, (f only added incoming edges to x and outgoing edges from y - none of these are in the path $x \rightsquigarrow y$). Hence G has path $x \rightsquigarrow y$.

We can compute f in logspace. We use a counter to loop through all nodes, u, and add edges (u,x) and (y,x). Changing the adj matrix can be done in logspace and we used a fixed counter bounded by input size, so f in logspace.

(c) i. The *complement* of a language $\overline{L} = \Sigma^* - L$.
The class NL is the class $\{\overline{L} \mid L \in \mathsf{NL}\}$.
The S-I theorem tells us that $\mathsf{NL} =$ co-NL.

ii. $\overline{\text{D-CONN}}$: Given G are there nodes x, v where there is no path from x to v.
To show that $\overline{\text{D-CONN}}$ is in NL, first guess x and v, then use the algorithm for the S-I theorem to calculate, $N(x)$, the set of nodes reachable from x. On the final stage, any successful computation examines all nodes y, and indicates whether $y \in N(x)$. We use this to a see if any of the y's reachable in $n-1$ edges are are v. If none are, then we return yes. If one of them is then we would've returned no on discovering $y = v$.
Algorithm:

```
k = 0
s = 1
while k < n - 1
ycount = 0;
for y in Nodes(G)
  zcount++; foundy = false
  if (Rch(x,z,k))
    ycount++
    for z in Nodes (G):
      zcount++
      if !foundy and ((z,y) in G or y = z)
         ycount++; foundy = true;
         if (y = v) return = NO;
```

```
            if zcount < s
                return FAIL
        s = ycount
        k++
    return = YES;
```

iii. Since $\overline{\text{D-CONN}}$ is in co-NL, by the S-I theorem, D-CONN is in NL. But we have from (b), RCH $\leqslant_{\log}$ DCONN and that RCH is NL-complete. Hence DCONN is NL-complete.

3. (a) i. A family of circuits is *uniform* iff there is a logspace bounded IOTM which, given input of 1^n, outputs circuit C_n - where $C_n(n) = \mathtt{tt}$ if $n \in S$, $C_n(n) = \mathtt{ff}$ if $n \notin S$, for any undecidable set S of natural numbers.

ii. NC_j is the class of languages that have some uniform family of circuits that decide the language in $O(\log^j(n))$ parallel time and $O(n^k)$ work for some k. NC is union over all j, $\mathsf{NC} = \cup_{j>0}\mathsf{NC}_j$.

iii. Suppose, n is a positive even with $|x| = 2k$ for some $k > 0$. Connect x_1 and x_{k+1} to an and-gate, x_2 to x_{k+2} to an and-gate, ..., x_k to x_{2k} to an and-gate. Then form a knockout-tournament with these k players, and connect them using and-gates to depth $\lceil \log(k) \rceil$. Output is $\mathtt{tt}$ iff $x_1 = x_{k+1}$, ..., $x_k = x_{2k}$ iff $x = x'x'$ for some $x' \in \{0,1\}^*$.
This circuit has depth $1 + \lceil \log(k) \rceil = O(\log(n))$ and size $k + k - 1 = O(n) = O(n^k)$ for $k = 1$. So C_n has $O(\log(n))$ depth and polynomial work. If n is odd let C_n be a circuit that simply outputs $\mathtt{ff}$ for all inputs. If $n = 0$, let C_n be a circuit that simply outputs $\mathtt{tt}$ for all inputs. Both of these circuits trivially have $O(\log(n))$ depth and polynomial work. So L has some uniform family of circuits that decide the language in $O(\log^1(n))$ parallel time and $O(n^k)$ work for some k. Hence $L \in \mathsf{NC}_1$.

iv. It would imply that all languages in P can be parallelized to have poly-log time with a polynomial number of processors, which seems unlikely. Some problems such as MAXFLOW seem inherently sequential. It would mean that $\mathsf{NC} = \mathsf{P}$, and that NC hierarchy of $\mathsf{NC}_1 \subseteq \mathsf{NC}_2 \subseteq ..$ would collapse, giving $\mathsf{NC} = \mathsf{NC}_j$ for all $j \geqslant 2$.

(b) i. Let M_f and M_g be the machines that compute f and g. Then the following machine M, computes $g(f(x))$ in logspace.
Assume the output tape for M_f is numbered from 0,1,2,...
Algorithm:
$i = 0$
Run M_f from the very beginning
until ith symbol written to its output tape

while(true):

execute next instruction of M_g (using current symbol for M_g's input tape as ith symbol on M_f's output tape).

if final state of M_g: return.

if M_g moved input tape head to the right:

$i = i+1$;

Run M_f from the very beginning until ith symbol written to its output tape

if M_g moved input tape head to the left:

$i = i-1$;

if $i < 0$: halt and succeed

Run M_f from the very beginning

until ith symbol written to its output tape The output tape of M_f is never larger than i and i is bounded by size of M_g's input tape. Given that M_g is a IOTM that computes g in logspace, M_g's input tape is bounded logarithmically by it's input size. So i can only be incremented logarithmically number of times. We have that $i \leqslant k\log(|f(x)|)$. Hence output tape of M_f is never larger $k\log(|f(x)|)$. Similarly, all the work tapes are bounded by $k\log(|f(x)|)$. M runs in $\log(|f(x)|)$ and computes $g(f(x))$. Hence $g(f(x))$ is logspace computable.

ii. Since f is uniform, there must be some circuit to calculate $f(x)$ that takes inputs of size x and gives outputs of size $|f(x)|$. Similarly, since g is uniform, there must be some circuit that that takes inputs of size $|f(x)|$ and gives outputs of size $|g(|f(x)|)|$. We can form a new logspace bounded machine which given input of 1^n, outputs this circuit, where $n = |x|$.

4.(a) i. A *Monte Carlo Turing Machine* for L is a precise polynomial time NDTM with 2 choices at each step that if $x \in L$, returns yes for $> \frac{1}{2}$ of its computation, and if $x \notin L$, returns no all of its computations. The class RP is the class of languages that have Monte Carlo Turing Machines.

ii. $\mathsf{P} \subseteq \mathsf{RP} \subseteq \mathsf{NP}$

$\mathsf{RP} \subseteq \mathsf{NP}$. Every language $L \in \mathsf{RP}$ has some Monte Carlo Turing Machine M. Clealy if $x \in L$, then M has some computation that leads returns to yes. So M' nondeterminstically decides L in p-time so $L \in \mathsf{NP}$.

$\mathsf{P} \subseteq \mathsf{RP}$. Every language $L \in \mathsf{P}$ has some determinstic Turing machine M that decides L. M can be transformed into a p-time precise nondeterminstic TM, M', with 2 choices (adding states for padding) that doesn't modify the outputs of M. Since M decides L, trivially, this will give yes for $> \frac{1}{2}$ of the computations when $x \in L$ and returns no for all computations when $x \notin L$. Hence M' is a Monte Carlo Turing Machine for L.

iii. Let $L_1, L_2 \in \mathsf{RP}$ be recognised by Monte Carlo TMs, M_1, M_2. We describe

machine M for L. There are $n+1$ splits to check.
Run S_1, S_1, ... S_{n+1}, where $S_i \equiv$ running $M_1(x[0..i))$ then $M_2(x[i..n))$. Return yes if any of the S_i has two successes. Return no if none of the S_i had double success. Each S_i is a binary tree of computations with $2^{p_1(i)+p_2(n-i)}$ comps. Overall M has $2^{\sum_{i=0}^{n}[p_1(i)+p_2(n-i)]}$ comps $= 2^{p_3(n)}$ for some polynomial p_3.
Suppose if $x \in L_1$ then M_1 returns yes for exactly $\frac{1}{2}+\varepsilon_1$ fraction of total comps. Suppose if $x \in L_2$ then M_2 returns yes for exactly $\frac{1}{2}+\varepsilon_2$ fraction of total comps. Then each S_i returns yes for exactly $(\frac{1}{2}+\varepsilon_1)(\frac{1}{2}+\varepsilon_2)$ fraction of the total S_i comps. So M returns yes for exactly $[(\frac{1}{2}+\varepsilon_1)(\frac{1}{2}+\varepsilon_2)]^{n+1} = \varepsilon$ of the $2^{p_3(n)}$ comps, for some ε. Suppose, given x we need to ran M for a total m times accept if it gave least one yes. Overall we accept a fraction $> 1-(1-\varepsilon)^m$ of comps. We can build an M' that runs m times such that $1-(1-\varepsilon)^m > \frac{1}{2}$. M' is a Monte Carlo TM for L_2, so $L_2 \in \mathsf{RP}$.

(b) 1. A shuffles the adjacency matrix.
2. B asks whether non-adjacent nodes x, y are in independent set.
3. Process repeats.

Suppose the independent set was invalid. Probability B catches A in the worst case is $\frac{1}{C(k,2)}$. If this process is repeated d times, the probability A escapes is

$$\left(1-\frac{1}{\frac{1}{2}k(k-1)}\right)^d$$

We would like the probability of A escaping after polynomial repetitions to be to be exponentially small. Notice that

$$\left(1-\frac{1}{\frac{1}{2}k(k-1)}\right)^{\frac{1}{2}k(k-1)} < \frac{1}{e} < \frac{1}{2}$$

since $k \leqslant n$, we have $\frac{1}{2}k(k-1)n \leqslant n^3$. So for $k > 1$:

$$\left(1-\frac{1}{\frac{1}{2}k(k-1)}\right)^{n^3} < \left(1-\frac{1}{\frac{1}{2}k(k-1)}\right)^{\frac{1}{2}k(k-1)n} < \frac{1}{2^n}$$

So if we repeat the process $p(n) = n^3$ times (polynomial), then the changes of A getting away with a lie would be $< \frac{1}{2^n}$ (exponentially small).