IAS/PCMI Summer Session 2000
Clay Mathematics Undergraduate Program
Basic Course on Computational Complexity

Lecture 8: Complete Problems for Other Complexity Classes

David Mix Barrington and Alexis Maciel
July 26, 2000

In our first week of lectures, we introduced the following complexity classes and established this sequence of inclusions:

$$\text{AC}^0 \subseteq \text{NC}^1 \subseteq \text{L} \subseteq \text{NL} \subseteq \text{AC}^1 \subseteq \cdots \subseteq \text{NC} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXP}.$$

(Once again, all the circuit classes below L must be logspace uniform while those between NL and P can either be all logspace uniform or all P uniform.) We also pointed out that among all these inclusions, only one is known to be strict: the first one.

In the preceding two lectures, we have examined the concept of NP-completeness and argued its relevance to the question of whether P equals NP. The NP-completeness of a problem can also be viewed as evidence that the problem is not in P, but this depends on how strongly one believes that P $\neq$ NP. In this lecture, our goal will be to identify complete problems for some of the other complexity classes.

1. NL-Completeness

Before defining the notion of NL-completeness, we first need to decide on the kind of reducibility that would be appropriate. Polynomial-time reducibility would not be very useful because NL $\subseteq$ P, which implies that the reduction itself would be able to solve any problem in NL. As a consequence, every language in NL (except $\emptyset$ and Σ^*) would be NL-complete.

A better choice is a kind of reducibility that corresponds to a subclass of NL. We also want this subclass X to be closed under function composition so that if $A \leq_X B$ and $B \in X$, then $A \in X$. This also guarantees that the reducibility is transitive (if $A \leq_X B$ and $B \leq_X C$, then $A \leq_X C$) and allows us to show the NL-completeness of a problem by a reduction from some other problem known to be NL-complete (because if A is NL-complete, $A \leq_X B$, and $B \in$ NL, then B is NL-complete).

A good choice for NL is the notion of logspace reducibility, denoted "$\leq_{\mathrm{L}}$". We then say that a language is NL-complete if it is in NL and every other language in NL logspace reduces to it. This notion of completeness allows us, in particular, to investigate the relationship between L and NL.

As in the case of NP-completeness, it is not difficult to come up with an NL-complete problem. Given a string x of length n, a nondeterministic machine N and a string of f 1's, does N accept x in space $\log f$? We call this the NL *evaluation* problem.

Theorem 1 NL *evaluation is* NL-*complete.*

Proof To show that NL evaluation is in NL, simulate N on x while keeping a count of the amount of space used. If ever that count exceeds $\log f$, the machine stops and rejects. N accepts x in space $\log f$ following a certain sequence of choices if and only if the simulator accepts following that same sequence of choices. The simulator runs in space $O(\log f)$, which is logarithmic in the length of its input.

Showing that every language A in NL reduces to NL evaluation is easy. Suppose N decides A in $c \log n$ space. Given an input x of length n, the reduction simply outputs x, a description of N, and the string 1^{n^c}. This can easily be done in $O(\log n)$ space. □

Can we come up with a more natural NL-complete problem? It turns out that we have already seen such a problem and implicitly used its NL-completeness in our study of NL and space-bounded computation in general. We are referring of course to graph reachability.

Theorem 2 *Graph reachability is* NL-*complete.*

Proof We already know that graph reachability is in NL. If N is any nondeterministic machine running in space $O(\log n)$, then given an input x, the reduction simply outputs the graph of configurations of N on x. This can be done in logarithmic space. □

2. L-Completeness

One notion of reducibility that makes sense for the class L is that of logspace-uniform NC^1 reducibility. We could again come up with an artificial L-complete problem. Instead, we will show right away that a certain version of graph reachability is L-complete. This will be the out-degree one version of graph reachability, where we want to accept only those graphs that are of out-degree one and contain a path from s to t.

Theorem 3 *The out-degree one version of graph reachability is* L*-complete.*

Proof First note that verifying whether a directed graph is of out-degree one can be done in $O(\log n)$ space. Then, to determine whether there is a path from s to t, it is simply a matter of following the unique path out of s. Therefore the problem is in L.

Now consider an arbitrary deterministic machine M running in space $O(\log n)$. The reduction will be a logspace-uniform NC^1 circuit that given x, computes the graph of configurations of M on x. That graph will be of out-degree one because M is deterministic. For each pair of configurations, determining whether one configuration leads to the other can be done by examining a single symbol of x. A logspace machine can run through all the possible pairs of configurations and construct a simple circuit that accesses the appropriate input symbol. This shows that every problem in L reduces to the out-degree one version of graph reachability. $\square$

3. NC^1-Completeness

Since we are talking about logspace-uniform NC^1, we can define NC^1-completeness in terms of logspace-uniform AC^0 reductions. One problem that is clearly *hard* for NC^1, in the sense that every problem in NC^1 reduces to it, is the *Boolean formula value problem* (BFVP): given a Boolean formula ϕ and values for the variables of ϕ, does ϕ evaluate to 1? If C is a logspace-uniform NC^1 circuit, then there is logspace machine that given any input of length n will output the formula equivalent to C. That machine can then be modified to output instead an AC^0 circuit (actually, an NC^0 circuit) that given an input x, outputs x together with a formula equivalent to C. This logspace-uniform AC^0 circuit performs the required reduction.

What is not clear, however, is whether BFVP in is NC^1. Given a formula ϕ and an input x, we cannot simply say "use the NC^1 circuit equivalent to ϕ on input x". We need a *single* circuit that can be used to evaluate *every* formula. Although we won't prove it here, it is known that BFVP is in logspace-uniform NC^1 and thus BFVP is complete for logspace-uniform NC^1.

In Lecture 5 of the Advanced Course, we saw that every language in NC^1 can be decided by a family of polynomial-length programs over the group S_5. This gives us another NC^1-complete problem. The *word problem over* S_5 is defined as follows: given a sequence of elements $x_1, \ldots, x_n$ and another element y, all from S_5, determine whether the product $x_1 \cdots x_n$ is equal to y.

Theorem 4 *The word problem over* S_5 *is complete for logspace-uniform* NC^1 *under logspace-uniform* AC^0 *reductions.*

Proof The product of n elements from any finite group can be easily computed by a simple binary tree of depth $O(\log n)$. Which implies that the word problem over any finite group is in logspace-uniform NC^1.

Now let L be any language in logspace-uniform NC^1. Then, as shown in the Advanced Course, L is decided by a logspace-uniform family of polynomial-length programs over S_5. Given an input x, we can construct, in logspace, an AC^0 circuit (actually, an NC^0 circuit) that outputs the yield of the appropriate program together with the permutation (12345). Since the program evaluates to (12345) precisely when x is in L, this circuit reduces L to the word problem over S_5. $\square$

As noted in the Advanced Lectures, polynomial-length programs over any non-Abelian group can compute all of NC^1. It follows that the word problem for any non-Abelian simple group is complete for NC^1.

4. PSPACE-Completeness

For the class PSPACE, we go back to polynomial-time reductions (although what follows also holds under finer notions of reducibility). We define here a problem that is PSPACE-complete under polynomial-time reductions and leave the proof of its completeness to the exercises. The key ideas needed in this proof will come up again in Lecture 11 when we relate PSPACE to the polynomial-time hierarchy.

A formula is *fully quantified* if all its variables are bound by some quantifier. (Such a formula is also called a *sentence*.) A formula is in *prenex normal form* if it is a sequence of quantifiers followed by a quantifier-free formula. If ϕ is a formula with no quantifiers that involves only the variables x_1 x_2 and x_3, then $\exists x_1 \forall x_2 \exists x_3 \phi(x_1, x_2, x_3)$

is an example of a formula that is both fully quantified and in prenex normal form. The *quantified Boolean formula problem* (QBF) is to determine whether a given fully quantified Boolean formula in prenex normal form is true or false.

Another way to look at this problem is to view it as a more general form of the formula satisfiability problem (SAT). With SAT, we are given a formula $\phi(x_1, \ldots, x_n)$ and want to know whether $\exists x_1 \ldots \exists x_n \phi(x_1, \ldots, x_n)$ is true. With QBF, we simply allow a more general type of quantification.

It is not hard to see that QBF is in PSPACE. If the formula is of the form $\exists x_1 \psi$, then recursively evaluate ψ with x_1 replaced by 0 and with x_1 replaced by 1. Accept if either of these last two formulas is true. In the case of the $\forall$ quantifier, accept if both are true. The depth of the recursion is the number of quantifiers. For each level of the recursion, we need to store the value of one variable, except at the last level, where we evaluate a formula in polynomial-time. In total, this algorithm uses a polynomial amount of space.

As mentioned earlier, the exercises ask you to show that QBF is PSPACE-complete. A key idea is the technique we used in simulation of polynomial-time machines by circuits of polynomial size. A second idea is the reduction of CIRCUIT SAT to SAT.

5. Exercises

1. Consider the following problem: given a sequence of $n \times n$ Boolean matrices and two indices i and j, what is the value of entry i, j in the product of the matrices? Define the product of two Boolean matrices as we did earlier: $C = A \times B$ if $C_{ij} = \bigvee_{k=1}^{n} A_{ik} B_{kj}$. Show that this iterated matrix multiplication problem is NL-complete.

2. Show that graph reachability is NL-complete under finer notions of reducibility, i.e., reducibilities that correspond to subclasses of L. How far can you go?

3. Given a directed graph G and a node s, say that *G has a fork from s* if there is a path from s to some node t that has out-degree greater than 1. Show that the problem of determining whether G has a fork from s is complete for L.

4. A directed graph is *levelled* if its vertices can be divided into sets $l_1, \ldots, l_d$ such that every edge in the graph goes from a vertex at one level to a vertex at the next level. The *width* of such a graph is the size of the largest level. For every number k, consider the following problem: given a levelled graph of width k, a

node $s \in l_1$ and a node $t \in l_d$, is there a path from s to t? Assume that the vertices of the graph are presented level by level. Show that this problem is NC^1-complete if $k \leq 5$.

5. Show that the circuit evaluation problem is complete for P under logspace reductions.

6. Show that planar circuit evaluation is P-complete. This is the circuit value problem but restricted to planar circuits, i.e., circuits that can be drawn in the plane with no wire crossings. Assume that the input contains an embedding of the circuit in the plane, so you can easily verify whether the given circuit is planar. (Hint: Design a planar subcircuit for simulating the crossing of two wires.)

7. Show that QBF is PSPACE-complete.