CSE 531: Computational Complexity I **Winter 2016**

Lecture 8: L, NL, NSPACE closed under complement

January 29, 2016

Lecturer: Paul Beame *Scribe: Paul Beame*

1 L, NL, NL-completeness

We have complexity classes

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}.$$

Last class, we considered the notion of PSPACE-completeness and the potential for separations of P from PSPACE for which polynomial-time reductions are appropriate. We now consider potential separations of classes within P and from NP. For these complexity classes, the notion of polynomial-time mapping reductions is too coarse since it does not distinguish L from P. We need a finer notion of reduction.

Definition 1.1. *$A \subseteq \{0,1\}^*$ is* logspace mapping reducible *to $B \subseteq \{0,1\}^*$, $A \leq_L B$ iff there is function $f : \{0,1\}^* \to \{0,1\}^*$ computable in space $O(\log n)$ such that for all $x \in \{0,1\}^*$, $x \in A \Leftrightarrow f(x) \in B$.*

Note that this definition is more standard than the equivalent definition given in the text but it uses a write-only output tape that is not included in the space bound. Since any log-space computation runs in polynomial time the following is immediate.

Proposition 1.2. *If $A \leq_L B$ then $A \leq_P B$.*

Lemma 1.3. *1. If $A \leq_L B$ and $B \leq_L C$ then $A \leq_L C$.*

2. If $A \leq_L B$ and $B \in \mathsf{L}$ then $A \in \mathsf{L}$.

3. If $A \leq_L B$ and $B \in \mathsf{NL}$ then $A \in \mathsf{NL}$.

Proof. We prove part 1. The other parts follow along the same lines. Let f be the reduction showing that $A \leq_L B$ and M_f be the associated log-space TM computing f. Let g be the reduction showing that $B \leq_L C$ and M_g be the associated log-space TM computing g. As with the case of $\leq_P$, the reduction showing $A \leq_L C$ is $g \circ f$. However we cannot simply run M_f followed by M_g as we did for the case of $\leq_P$ because this would require that the output of $y = f(x)$ on which M_g is run be written on a tape that can be read and in general it is too long. Instead, the computation of $g \circ f$ proceeds by producing each bit of y only as it is needed and recomputing f to obtain other bits.

The simulation will maintain the work tapes of M_f and of M_g as well as an index i representing the position of the read head of M_g on inout y and an index j representing the position of the write head of M_f on its output tape. More precisely, the algorithm to compute $g \circ f$ is as follows: On input x, set $i = j = 0$ and run M_g. For each step of M_g, run M_f on input x ignoring all outputs except when $j = i$. Use the output at position i to determine the action of the next step of M_g and update i accordingly. The total space required is $O(\log n)$ plus that of M_f and M_g which is $O(\log n)$ overall. □

Definition 1.4. *1. B is* NL-hard *iff $A \leq_L B$ for all $A \in \mathsf{NL}$.*

2. B is NL-complete *iff (a) $B \in \mathsf{NL}$ and (b) B is NL-hard.*

Lemma 1.5. *$PATH$ is NL-complete.*

Proof. We already have shown that $PATH \in \mathsf{NL}$. We name the nondeterministic $O(\log n)$-space procedure that we used to guess and verify a path from s to t of length at most $i = n$ by $CheckPath_G(s, t, i)$. We will find this useful later.

It remains to show that $PATH$ is NL-hard. Let $A \in \mathsf{NL}$ and M_A be a normal form $O(\log n)$-space NTM that decides A. As before we have $x \in A$ if and only if there is a path in $G_{M_A,x}$ from C_0 to C_{accept}. Since M_A is logspace, nodes of $G_{M_A,x}$ can be specified by $O(\log n)$ bits each. The reduction simply is $x \mapsto [G_{M_A,x}, C_0, C_{accept}]$ which clearly is correct. It can be computed in $O(\log n)$ space since each edge simply requires checking, for each pair C, D of configurations of M_A, whether C yields D in one step of M_A. Therefore $A \leq_L PATH$. □

There are other path problems that are also important for complexity. We can also define
$1PATH = \{[G, s, t] \mid G$ is a outdegree 1 graph with a path from s to $t\}$
$UPATH = \{[G, s, t] \mid G$ is an undirected graph with a path from s to $t\}$

It is immediate $1PATH \in \mathsf{L}$ since the algorithm simply needs to follow the unique path in G from s and see if it encounters t. Somewhat surprisingly, the undirected graph case has similar complexity. This is a difficult theorem that is beyond the scope of this class. It shows that the distinction between L and NL is precisely the difference between undirected and directed graph reachability.

Theorem 1.6 (Reingold 2005)**.** $UPATH \in \mathsf{L}$.

So far, we have used just one notion of reduction when we have defined completeness for a complexity class. In general one can associate many notions of reduction for a given complexity class and can talk about a problem being complete for a class C under reductions of type $\leq'$. (Though to be useful, the notion $\leq'$ should not be too powerful.)

Thus the Cook-Levin Theorem shows that SAT is complete for NP under $\leq_P$ reductions. (Also sometimes referred to as $\leq_P$-complete for NP.) With the more refined notion $\leq_P$ we can also define the notions of *complete for* NP *under* $\leq_L$ *reductions* (or *logspace-complete for* NP) and *complete for* P *under* $\leq_L$ *reductions* (*logspace-complete for* P).

Returning to the simulations of Turing machine computations by circuits and the reductions given in the proof of the Cook-Levin theorem, one can see that the construction of the circuit is very local, requiring only indices of time and Turing machine tape position. It follows that we have the following results using the old constructions.

Theorem 1.7 (Cook)**.** *SAT is complete for NP under $\leq_L$ reductions.*

Theorem 1.8 (Ladner)**.** *$CIRCUIT\text{-}VALUE$ is complete for P under $\leq_L$ reductions.*

2 Nondeterministic Space is Closed Under Complement

In analogy with coNP we can also define $\mathsf{coNL} = \{\overline{\mathsf{L}} \mid \mathsf{L} \in \mathsf{NL}\}$. More generally $\mathsf{coNSPACE(S(n)} = \{\overline{\mathsf{L}} \mid \mathsf{L} \in \mathsf{NSPACE(S(n))}\}$.

Just as $PATH$ is NL-complete, the following language
$\overline{PATH} = \{[G, s, t] \mid$ directed graph G does not have a path from s to $t\}$
is a complete problem for coNL. (Strictly speaking, $\overline{PATH}$ should contain inputs that are not well-formed but they can be easily detected in deterministic logspace so we ignore them.)

Theorem 2.1 (Immerman-Szelepscenyi)**.** $\mathsf{NL} = \mathsf{coNL}$.

Proof. It suffices to show that $\overline{PATH} \in \mathsf{NL}$. On input $[G, s, t]$ let $G = (V, E)$ and compute $n = |V|$ We begin with an assumption and then we will clear that assumption.

Suppose that the algorithm has access to the exact value N, the number of vertices of G reachable from s. The idea for showing that there is no path to t will be to guess and verify N *other* vertices of G that have paths from s and hence t does not.

The algorithm is as follows:

$count \leftarrow 0$
for all $v \in V - \{t\}$
    Guess whether v is reachable from s
    if Guess="yes" then
        if $CheckPath_G(s, v, n)$ then
            $count \leftarrow count + 1$
        else

```
                reject
            endif
        endif
end for
if count = N then accept
```

It clearly only needs to retain $count$, N, v, and n, plus the space for $CheckPath_G$ which is clearly $O(logn)$ space in total.

It remains to nondeterministically compute N. We write V_i for the set of nodes of G reachable from s in at most i steps and $N_i = |V_i|$. Clearly $V_0 = \{s\}$ and $N_0 = 1$. We now how to compute N_i from N_{i-1} for $i \geq 1$. The general idea to confirm for each vertex whether or not it is in V_i. This is easy to do for elements of V_i. To confirm that it is not in V_i the algorithm checks that it is not adjacent to any element of V_{i-1}. It needs the count N_{i-1} to ensure that it has considered every element of V_{i-1}.

```
count ← 0
for all v ∈ V
    Guess whether v is in V_i
    if Guess="yes" then
        if CheckPath_G(s, v, i) then
            count ← count + 1
        else
            reject
        endif
    else
    if Guess="no" then
    oldcount ← 0
    for all u ∈ V
        Guess' whether u is in V_{i-1}
        if Guess'="yes" then
            if CheckPath_G(s, v, i − 1) then
            oldcount ← oldcount + 1
            if (u, v) ∈ E then reject;
            else
                reject
            endif
        endif
    end for
    if oldcount ≠ N_{i-1} then reject;
end for
N_i ← count
```

The algorithm requires only $count$, $oldcount$, u, v, i, N_{i-1} and N_i in addition to the space of $CheckPath_G$. After each iteration, i is incremented. At the end, $N = N_n$. $\square$

Corollary 2.2. *For all space constructible* $S(n) \geq \log_2 n$*,* $\mathsf{NSPACE(S(n))} = \mathsf{coNSPACE(S(n))}$.

Proof. The argument is a padding argument similar to the one by which we previously showed that if $clP = \mathsf{NP}$ then $\mathsf{EXP} = \mathsf{NEXP}$. Let $k(n) = 2^{S(n)}$. Suppose that $A \in \mathsf{NSPACE(S(n))}$ and let M_A be the space $O(S(n))$ NTM for A. Then define $A_{pad} = \{(x, 1^{k(|x|)}) \mid x \in A\}$. Since $\log_2 k(n) = S(n)$, we can decide A_{pad} by the following NTM M': on input $y = (x, 1^i)$, use the space constructibility of S to check that $i = k(|x|)$ and, if so, run M_A on input x. M_A uses space at most $cS(|x|)$ for some constant c. By construction, this is $O(\log |y|)$ so M' runs in space $O(\log n)$, hence $A_{pad} \in \mathsf{NL}$. By the above theorem $A_{pad} \in \mathsf{coNL}$, i.e. $\overline{A_{pad}} \in \mathsf{NL}$. Therefore there is an $O(\log n)$-space NTM M'' that decides $\overline{A_{pad}}$. The NTM for $\overline{A_{pad}}$ on input x now uses the space constructibility of $S(n)$ to compute $k(|x|)$ and acts as if it has appended $1^{k(|x|)}$ onto x and simulates M'' on input $(x, 1^{k(|x|)})$. Note that it cannot actually add the 1's because there are two many of them (and one cannot change the input) but instead it maintains a counter of the head position of M'' when it is not on x and returns value 1 for each such position when needed. The space of the algorithm is $O(S(n))$ as required. $\square$

3 The Polynomial-time Hierarchy

So far we have considered NP problems such as

$$INDSET = \{[G, k] \mid G \text{ has an independent set of size } \geq k\}.$$

However, this does not characterize all reasonable related problems such as

$$EXACT\text{-}INDSET = \{[G, k] \mid \text{ the largest independent set of } G \text{ has size } = k\}.$$

In other words,

$$[G, k] \in EXACT\text{-}INDSET \Leftrightarrow ([G, k] \in INDSET \ \wedge \ [G, k+1] \notin INDSET)$$

In particular, if $Indep(U, G)$ denotes the predicate that U is an independent set of G, then we can say that $[G, k] \in EXACT\text{-}INDSET$ if and only if

$$\exists U \subseteq V \ (|U| = k \wedge Indep(U, G)) \ \wedge \ \forall U' \subseteq V \ (|U'| = k + 1 \rightarrow Indep(U, G)).$$

Similarly we can define

$$MINDNF = \{[\varphi, k] \mid \varphi \text{ is a DNF that has an equivalent DNF of size } \leq k\}.$$

We can again express membership in terms of a quantified formula

$$\exists \varphi' \forall x \in \{0,1\}^n (|\varphi'| \leq k \wedge DNF(\varphi) \wedge DNF(\varphi') \wedge (\varphi(x) = \varphi'(x))).$$

These are problems that have both NP and coNP aspects.

Definition 3.1. *Define the class of languages Σ_2^P to be the class of languages $A \subseteq \{0,1\}^*$ such that there are polynomial bounds p_1 and p_2 and a polynomial-time computable verifier V such that*

$$x \in A \iff \exists y_1 \in \{0,1\}^{p_1(|x|)} \forall y_2 \in \{0,1\}^{p_2(|x|)}\, V(x, y_1, y_2) = 1.$$

We can define $\Pi_2^P = \{\overline{L} \mid L \in \Sigma_2^P\}$. There is a similar definition of Π_2^P as the set of languages A such taht

$$x \in A \iff \forall y_1 \in \{0,1\}^{p_1(|x|)} \exists y_2 \in \{0,1\}^{p_2(|x|)}\, V(x, y_1, y_2) = 1.$$

With these definitions we see that $MINDNF \in \Sigma_2^P$ and, since the quantified parts of $EXACT\text{-}INDSET$ are independent of each other, $EXACT\text{-}INDSET \in \Sigma_2^P \cap \Pi_2^P$.

We will discuss more of the hierarchy in the next class.