Complexity Theory
Lecture 11

Anuj Dawar

University of Cambridge Computer Laboratory

Easter Term 2014

http://www.cl.cam.ac.uk/teaching/1314/Complexity/

Inclusions

We have the following inclusions:

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

where $\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{TIME}(2^{n^k})$

Moreover,

$$\mathsf{L} \subseteq \mathsf{NL} \cap \mathsf{co\text{-}NL}$$

$$\mathsf{P} \subseteq \mathsf{NP} \cap \mathsf{co\text{-}NP}$$

$$\mathsf{PSPACE} \subseteq \mathsf{NPSPACE} \cap \mathsf{co\text{-}NPSPACE}$$

Reachability

Recall the Reachability problem: given a *directed* graph $G = (V, E)$ and two nodes $a, b \in V$, determine whether there is a path from a to b in G.

A simple search algorithm solves it:

1. mark node a, leaving other nodes unmarked, and initialise set S to $\{a\}$;
2. while S is not empty, choose node i in S: remove i from S and for all j such that there is an edge (i, j) and j is unmarked, mark j and add j to S;
3. if b is marked, accept else reject.

NL Reachability

We can construct an algorithm to show that the Reachability problem is in NL:

1. write the index of node a in the work space;
2. if i is the index currently written on the work space:
   (a) if $i = b$ then accept, else
   guess an index j ($\log n$ bits) and write it on the work space.
   (b) if (i, j) is not an edge, reject, else replace i by j and return to (2).

Complementation

A still more clever algorithm for Reachability has been used to show that nondeterministic space classes are closed under complementation:

If $f(n) \geq \log n$, then

$$\mathsf{NSPACE}(f(n)) = \text{co-}\mathsf{NSPACE}(f(n))$$

In particular

$$\mathsf{NL} = \text{co-}\mathsf{NL}.$$

Logarithmic Space Reductions

We write

$$A \leq_L B$$

if there is a reduction f of A to B that is computable by a deterministic Turing machine using $O(\log n)$ workspace (with a *read-only* input tape and *write-only* output tape).

Note: We can compose $\leq_L$ reductions. So,

$$\text{if } A \leq_L B \text{ and } B \leq_L C \text{ then } A \leq_L C$$

NP-complete Problems

Analysing carefully the reductions we constructed in our proofs of NP-completeness, we can see that SAT and the various other NP-complete problems are actually complete under $\leq_L$ reductions.

Thus, if SAT $\leq_L A$ for some problem A in L then not only P = NP but also L = NP.

P-complete Problems

It makes little sense to talk of complete problems for the class P with respect to polynomial time reducibility $\leq_P$.

There are problems that are complete for P with respect to *logarithmic space* reductions $\leq_L$.

One example is CVP—the circuit value problem.

- If CVP $\in$ L then L $=$ P.
- If CVP $\in$ NL then NL $=$ P.

CVP

CVP - the *circuit value problem* is, given a circuit, determine the value of the result node n.

CVP is solvable in polynomial time, by the algorithm which examines the nodes in increasing order, assigning a value `true` or `false` to each node.

CVP is complete for P under L reductions.

That is, for every language A in P,

$$A \leq_L \mathsf{CVP}$$

Reachability

Similarly, it can be shown that Reachability is, in fact, NL-complete.

For any language $A \in \mathsf{NL}$, we have $A \leq_L$ Reachability

$\mathsf{L} = \mathsf{NL}$ if, and only if, Reachability $\in \mathsf{L}$

Note: it is known that the reachability problem for *undirected* graphs is in L.

Provable Intractability

Our aim now is to show that there are languages (*or, equivalently, decision problems*) that we can prove are not in P.

This is done by showing that, for every *reasonable* function f, there is a language that is not in $\mathsf{TIME}(f)$.

The proof is based on the diagonal method, as in the proof of the undecidability of the halting problem.

Constructible Functions

A complexity class such as $\mathsf{TIME}(f)$ can be very unnatural, if f is.

We restrict our bounding functions f to be proper functions:

Definition

A function $f : \mathbb{N} \to \mathbb{N}$ is *constructible* if:

- f is non-decreasing, i.e. $f(n+1) \geq f(n)$ for all n; and
- there is a deterministic machine M which, on any input of length n, replaces the input with the string $0^{f(n)}$, and M runs in time $O(n + f(n))$ and uses $O(f(n))$ *work space.*

Examples

All of the following functions are constructible:

- $\lceil \log n \rceil$;
- n^2;
- n;
- 2^n.

If f and g are constructible functions, then so are $f + g$, $f \cdot g$, 2^f and $f(g)$ (this last, provided that $f(n) > n$).

Using Constructible Functions

NTIME(f) can be defined as the class of those languages L accepted by a *nondeterministic* Turing machine M, such that for every $x \in L$, there is an accepting computation of M on x of length at most $O(f(n))$.

If f is a constructible function then any language in NTIME(f) is accepted by a machine for which all computations are of length at most $O(f(n))$.

Also, given a Turing machine M and a constructible function f, we can define a machine that simulates M for $f(n)$ steps.

Inclusions

The inclusions we proved between complexity classes:

- $\mathsf{NTIME}(f(n)) \subseteq \mathsf{SPACE}(f(n))$;
- $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{TIME}(k^{\log n + f(n)})$;
- $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{SPACE}(f(n)^2)$

really only work for *constructible* functions f.

The inclusions are established by showing that a deterministic machine can simulate a nondeterministic machine M for $f(n)$ steps.

For this, we have to be able to compute f within the required bounds.

Time Hierarchy Theorem

For any constructible function f, with $f(n) \geq n$, define the f-bounded *halting language* to be:

$$H_f = \{[M], x \mid M \text{ accepts } x \text{ in } f(|x|) \text{ steps}\}$$

where $[M]$ is a description of M in some fixed encoding scheme.

Then, we can show

$H_f \in \mathsf{TIME}(f(n)^2)$ and $H_f \notin \mathsf{TIME}(f(\lfloor n/2 \rfloor))$

Time Hierarchy Theorem

For any constructible function $f(n) \geq n$, $\mathsf{TIME}(f(n))$ is properly contained in $\mathsf{TIME}(f(2n+1)^2)$.