Complexity Theory
Lecture 12

Anuj Dawar

University of Cambridge Computer Laboratory
Easter Term 2013

http://www.cl.cam.ac.uk/teaching/1213/Complexity/

Complexity Classes

We have established the following inclusions among complexity classes:

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

Showing that a problem is NP-complete or PSPACE-complete, we often say that we have proved it intractable.

While this is not strictly correct, a proof of completeness for these classes does tell us that the problem is structurally difficult.

Similarly, we say that PSPACE-complete problems are harder than NP-complete ones, even if the running time is not higher.

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 $\mathsf{TIME}(f(n))$.

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 $\mathsf{TIME}(f(n))$ can be very unnatural, if $f(n)$ is.

We restrict our bounding functions $f(n)$ 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

$\mathsf{NTIME}(f(n))$ 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 $\mathsf{NTIME}(f(n))$ 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

$H_f \in \mathsf{TIME}(f(n)^3)$ 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 $\mathsf{TIME}(f(2n+1)^3)$.

Strong Hierarchy Theorems

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

Space Hierarchy Theorem
For any pair of constructible functions f and g, with $f = O(g)$ and $g \neq O(f)$, there is a language in $\mathsf{SPACE}(g(n))$ that is not in $\mathsf{SPACE}(f(n))$.

Similar results can be established for nondeterministic time and space classes.

Consequences

- For each k, $\mathsf{TIME}(n^k) \neq \mathsf{P}$.
- $\mathsf{P} \neq \mathsf{EXP}$.
- $\mathsf{L} \neq \mathsf{PSPACE}$.
- Any language that is EXP-complete is not in P.
- There are no problems in P that are complete under linear time reductions.

Descriptive Complexity

Descriptive Complexity is an attempt to study the complexity of problems and classify them, not on the basis of how difficult it is to *compute* solutions, but on the basis of how difficult it is to *describe* the problem.

This gives an alternative way to study complexity, independent of particular machine models.

Based on *definability in logic*.

Graph Properties

As an example, consider the following three decision problems on *graphs*.

1. Given a graph $G = (V, E)$ does it contain a *triangle*?
2. Given a directed graph $G = (V, E)$ and two of its vertices $s, t \in V$, does G contain a *path* from a to b?
3. Given a graph $G = (V, E)$ is it *3-colourable*? That is,

   is there a function $\chi : V \to \{1, 2, 3\}$ so that whenever $(u, v) \in E$, $\chi(u) \neq \chi(v)$.

Graph Properties

1. Checking if G contains a triangle can be solved in *polynomial time* and *logarithmic space*.

2. Checking if G contains a path from a to b can be done in *polynomial time*.

   Can it be done in *logarithmic space*?

   Unlikely. It is NL-complete.

3. Checking if G is 3-colourable can be done in *exponential time* and *polynomial space*.

   Can it be done in *polynomial time*?

   Unlikely. It is NP-complete.

Logical Definability

In what kind of formal language can these decision problems be *specified* or *defined*?

The graph $G = (V, E)$ contains a triangle.

$$\exists x \in V\, \exists y \in V\, \exists z \in V (x \neq y \wedge y \neq z \wedge x \neq z \wedge E(x,y) \wedge E(x,z) \wedge E(y,z))$$

The other two properties are *provably* not definable with only first-order quantification over vertices.

Second-Order Quantifiers

3-Colourability and *reachability* can be defined with quantification over *sets of vertices*.

$$\begin{aligned}
\exists R \subseteq V\, \exists B \subseteq V\, \exists G \subseteq V\quad & \\
& \forall x (Rx \vee Bx \vee Gx) \wedge \\
& \forall x (\neg(Rx \wedge Bx) \wedge \neg(Bx \wedge Gx) \wedge \neg(Rx \wedge Gx)) \wedge \\
& \forall x \forall y (Exy \to (\neg(Rx \wedge Ry) \wedge \\
& \qquad\qquad \neg(Bx \wedge By) \wedge \\
& \qquad\qquad \neg(Gx \wedge Gy)))
\end{aligned}$$

$$\forall S \subseteq V (a \in S \wedge \forall x \forall y ((x \in S \wedge E(x,y)) \to y \in S) \to b \in S)$$

Descriptive Complexity

Any property of graphs that is expressible in *first-order logic* is in L.

A property of graphs is definable in *existential second-order logic* if, and only if, it is in NP.

Is there a logic, intermediate between first and second-order logic that expresses exactly graph properties in P?

The End

Please provide *feedback*, using the link sent to you by e-mail.