Introduction to Logic in Computer Science: Autumn 2006

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam

Plan for Today

We are going to cover the following topics:

- More on relationships between complexity classes
- Complements of complexity classes
- Completeness with respect to a complexity class: what are the "most difficult" problems within a given class?

Regarding relationships, we sketch proofs for some important (difficult) results. Concerning completeness, we just introduce the definitions; the real stuff is for next week.

Remark: Any results stated in the sequel assume that all functions $f : \mathbb{N} \to \mathbb{N}$ are "proper complexity functions" ($\rightsquigarrow$ last week).

The Time Hierarchy Theorem

Not all complexity classes are the same (we weren't sure before!):

Theorem 1 *We have* $\mathbf{TIME}(f(n)) \subset \mathbf{TIME}((f(2n+1))^3)$ *for any function* $f : \mathbb{N} \to \mathbb{N}$ *with* $f(n) \geq n$.

Proof idea: Define a *time-bound halting problem* as follows:

$$H_f = \{M; x \mid \text{machine } M \text{ accepts } x \text{ after} \leq f(|x|) \text{ steps}\}$$

Then prove the following lemmas:

(1) $H_f \in \mathbf{TIME}((f(n))^3)$: Roughly, each step can be simulated in quadratic time; so the whole thing takes cubic time (see book).

(2) $H_f \notin \mathbf{TIME}(f(\lfloor \frac{n}{2} \rfloor))$
Proof by contradiction: Suppose there exists a machine M_{H_f} deciding H_f in $f(\lfloor \frac{n}{2} \rfloor)$. Define machine D_f such that

$D_f(M)$ says "yes" *iff* $M_{H_f}(M; M)$ says "no" (else "no")

Then wonder what the output for $D_f(D_f)$ would be ... ✓

P and EXPTIME

Recall the theorem: $\mathbf{TIME}(f(n)) \subset \mathbf{TIME}((f(2n+1))^3)$.

Corollary 1 *We get* $\mathbf{P} \subset \mathbf{EXPTIME}$.

Proof: Apply the Time Hierarchy Theorem to $f(n) = 2^n$, yielding $\mathbf{TIME}(2^n) \subset \mathbf{TIME}(2^{(2n+1)\cdot 3})$. The result then follows from $\mathbf{P} \subseteq \mathbf{TIME}(2^n)$ and $\mathbf{TIME}(2^{(2n+1)\cdot 3}) \subseteq \mathbf{EXPTIME}$. ✓

Savitch's Theorem

Theorem 2 (Savitch) $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{SPACE}(f^2(n))$ *for any function* $f : \mathbb{N} \to \mathbb{N}$ *with* $f(n) \geq \log n$.

Proof idea: First show that REACHABILITY $\in \mathbf{SPACE}(\log^2 n)$:

- Let PATH(x, y, t) be the problem: is there a path $\leq t$ from x to y?
  Algorithm: Recursively try all nodes z and check if both PATH$(x, z, t/2)$ and PATH$(z, y, t/2)$ hold.
  Analysis: Recursion depth is at most $\log t$. Storing one node takes $\log n$ space (binary representation). The claim then follows from REACHABILITY $=$ PATH(x, y, n).

Now take any problem $\in \mathbf{NSPACE}(f(n))$. The machine solving it can have at most $c^{f(n)}$ states. Solve REACHABILITY for the corresponding graph of $\leq c^{f(n)}$ nodes $\Rightarrow$ problem $\in \mathbf{SPACE}(f^2(n))$. ✓

Corollary 2 *We get* $\mathbf{PSPACE} = \mathbf{NPSPACE}$.

Compare this with the situation for *time:* we don't know whether $\mathbf{P} = \mathbf{NP}$, but strongly suspect *not*, namely $\mathbf{P} \subset \mathbf{NP}$.

Summary of Complexity Class Relationships

This is what we know so far:

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE} \subseteq \mathbf{EXPTIME}$$
$$\mathbf{P} \subset \mathbf{EXPTIME}$$

Hence, one of the $\subseteq$'s above must actually be strict, but we don't know which. Most experts believe they are probably all strict. In the case of $\mathbf{P} \subset^{?} \mathbf{NP}$, the answer is worth \$1.000.000.

Complements

- Let P be a class of decision problems. The *complement* $\overline{P}$ of P is the set of all instances that are *not* correct solutions for P.

  If you think of a class of problems as a *language*, then this is the usual notion of complementation.

- Example: SAT is the problem of checking whether a given formula of propositional logic is satisfiable. The complement of SAT is checking whether a given formula is *not* satisfiable (which is equivalent to checking whether its negation is valid).

- For any complexity class $\mathcal{C}$, we define $\mathbf{co}\mathcal{C} = \{\overline{P} \mid P \in \mathcal{C}\}$.

- Example: **coNP** is the class of problems for which a negative answer can be verified in polynomial time.

Problems and Languages

Sometimes a small shift in terminology can make things clearer ...

A class of problems can be understood as a *language* over some given alphabet: each problem instance can be encoded as a word over the alphabet, and the language is the set of all words corresponding to problem instances with a positive answer.

For instance, for REACHABILITY, we can describe any graph $G = (V, E)$ together with two chosen vertices as a string $\subseteq \{0, 1\}^*$. The language corresponding to REACHABILITY would then be the set of words encoding an instance of this class of problems for which the correct answer would be "yes".

The complement of a language L (class of problems) with respect to an alphabet Σ is then easily defined as $\overline{L} = \Sigma^* \setminus L$.

A complexity class $\mathcal{C}$ is said to be *closed* under complementation *iff* $L \in \mathcal{C}$ implies $\overline{L} \in \mathcal{C}$ (similar for union and intersection).

Results for Complements

The following result follows from the fact that a deterministic algorithm for a given problem can be turned into a deterministic algorithm for the complement of that problem by simply swapping "yes" and "no" in the output.

Proposition 1 $\mathcal{C} = \mathbf{co}\mathcal{C}$ *for any deterministic complexity class* $\mathcal{C}$*.*

For example, we have $\mathbf{P} = \mathbf{coP}$. But nobody knows whether $\mathbf{NP} =^{?} \mathbf{coNP}$ (people tend to think not).

An important theorem that we just state here without proof:

Theorem 3 (Immerman-Szelepscényi) *If* $f(n) \geq \log n$*, then* $\mathbf{NSPACE}(f(n)) = \mathbf{coNSPACE}(f(n))$*.*

That is, non-deterministic space is closed under complementation. An example would be $\mathbf{NPSPACE} = \mathbf{coNPSPACE}$.

Reductions

To compare the hardness of different classes of problems, we introduce the concept of reduction.

Problem A *reduces* to problem B if we can translate any instance of A into an instance of B that we can then feed into a solver for B to obtain an answer to our original question (of type A).

If the translation (reduction) process itself is not too complex, then we can rightfully claim that problem B *is at least as hard* as problem A. This is because a B-solver can solve any instance of A, and maybe many more problems.

There are several ways of making this notion precise ...

Polynomial-Time Reductions

The important bit is that it has to be possible that the reduction be computed in *polynomial time* (this makes sense if we are interested in complexity classes **P** and above). Two variants:

- *Karp reduction* (aka. *many-one reduction*): Reduce an A-instance to a B-instance in polynomial time and then solve it using the B-solver (one call at the very end of the reduction).
- *Turing reduction* (aka. *Cook reduction*): Solve an A-instance in polynomial time using a polynomial number of B-oracles.

We'll usually gloss over the details (as is common practice in the literature) and not explicitly state which type of reduction we use.

Remark: Papadimitriou uses logarithmic-space rather than polynomial-time reductions in his book ... more powerful as he explains, but also less standard.

Hardness and Completeness

Let $\mathcal{C}$ be a complexity class.

- A language L is *$\mathcal{C}$-hard* iff any $L' \in \mathcal{C}$ is polynomial-time reducible to L. That is, the $\mathcal{C}$-hard problems include the very hardest problems inside of $\mathcal{C}$, and even harder ones.
- A language L is *$\mathcal{C}$-complete* iff L is $\mathcal{C}$-hard and $L \in \mathcal{C}$. That is, these are the hardest problems in $\mathcal{C}$, and only those.

Example: As we shall see later, SAT is an example for an **NP**-complete problem. That means, it is as hard as any other problem in **NP**; any other such problem can be reduced to SAT.

Summary

Today's material is covered in Chapters 7 and 8 of Papadimitriou's book.

- Review of definition of complexity classes and simple relationships between them (mostly previous set of slides).
- Time Hierarchy Theorem and Savitch's Theorem
- Complements and Immerman-Szelepscényi Theorem
- Polynomial-time reductions between different problems
- Completeness with respect to a complexity class

What next?

- We are going to see actual examples for decision problems that are complete with respect to specific complexity classes.
- For each such class we will have to prove completeness *once* from first principles; all subsequent results rely on reductions.

C.H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.