CS 221: Computational Complexity **Prof. Salil Vadhan**

Lecture Notes 3

February 1, 2010 **Scribe: Mike Ruberry**

Contents

1 Relations between resources and their complexity classes

1.1 Inclusions

The "hierarchy theorems" from the previous lecture showed that for "constructible" functions:[1]

$\mathbf{DTIME}(f(n)) \not\subseteq \mathbf{DTIME}(o(f(n)/\log(f(n)))$

$\mathbf{NTIME}(f(n)) \not\subseteq \mathbf{NTIME}(o(f(n)/\log(f(n)))$

$\mathbf{SPACE}(f(n)) \not\subseteq \mathbf{SPACE}(o(f(n)))$

Note that these results refer to comparing a resource with itself (more time vs. less time, more space vs. less space, etc.).

Much more difficult is comparing *different* resources to each other (eg. time vs. nondeterministic time, as in the **P** vs. **NP** question). Most of what we know about the relationship between time, space, and nondeterministic time is captured by the following:

Theorem 1 $\mathbf{DTIME}(f(n)) \subseteq \mathbf{NTIME}(f(n)) \subseteq \mathbf{SPACE}(f(n)) \subseteq \bigcup_c \mathbf{DTIME}(c^{f(n)})$ *for* $f(n) \geq \log(n)$

Proof:

1. $\mathbf{DTIME}(f(n)) \subseteq \mathbf{SPACE}(f(n))$ follows because at each step at most one unit of memory may be addressed.

[1] Non-constructible functions may exhibit strange behavior, such as $\mathbf{DTIME}(f(n)) = \mathbf{DTIME}(2^{2^{2^{f(n)}}})$

2. **NTIME**$(f(n)) \subseteq$ **SPACE**$(f(n))$ since all computation paths of length $f(n)$ may be attempted.

3. **SPACE**$(f(n)) \subseteq \bigcup_c$ **DTIME**$(c^{f(n)})$ for $f(n) \geq \log(n)$ because a space $f(n)$ machine has at most $c^{f(n)}$ distinct configurations on any input, and thus must halt within $c^{f(n)}$ steps if it ever halts. (A repeated configuration implies an infinite loop. The number of configurations is at most $|\Gamma|^{f(n)} \cdot |Q| \cdot n \cdot f(n)^{O(1)}$, where Γ is the tape alphabet, Q is the set of states, and $n \cdot f(n)^{O(1)}$ bounds the number of possible locations of the heads. The factor of n is why we need $f(n) \geq \log n$.) Thus, we can halt the machine after $c^{f(n)}$ steps (if it runs for longer, then it is in an infinite loop and will never halt.)

■

Figure 1 is an image of the inclusions of common complexity classes, using the above theorem. Given a complexity class **C**, its class of complements is defined as **co-C** $= \{\bar{L} : \ L \in C\}$, which is *not* the same as $\bar{\mathbf{C}}$. We don't know anything more about the relations between these classes beyond what follows from the above theorem, the hierarchy theorems, and translation results like those below. For example, we know that **L** $\neq$ **PSPACE** by the space hierarchy theorem, but it is open whether **L** $=$ **NP**. And we know that **P** $\neq$ **EXP**, but it is open whether **P** $=$ **PSPACE**.

1.2 Implications of class equality

Translation: equality translates upwards, inequality translates downwards.

Examples:

P $=$ **NP**	$\Rightarrow$ **NP** $=$ **co-NP**	(PS0)
P $=$ **NP**	$\Rightarrow$ **EXP** $=$ **NEXP**	(thm below)
L $=$ **P**	$\Rightarrow$ **PSPACE** $=$ **EXP**	(omitted, similar to thm below)

Theorem 2 **P** $=$ **NP** $\Rightarrow$ **EXP** $=$ **NEXP**

Proof: Assume **P** $=$ **NP**. Given $L \in$ **NEXP**, consider its exponentially "padded" version $L' = \{x01^{2^{|x|^c}} : x \in L\}$.

We argue that $L' \in$ **NP**:

1. Given x', reject if not of the form $x01^{2^{|x|^c}}$. This verification is performed in linear time.

2. Run the 2^{n^c} nondeterministic algorithm for L on x, which is exponentially shorter than x'. This is also performed in linear time.

This implies $L' \in$ **P** since, per our assumption, **P** $=$ **NP**. This implies $L \in$ **EXP**: we can decide whether $x \in L$ by running the poly-time algorithm for L' on $x' = x01^{2^{|x|^c}}$. ■

These translation/padding arguments are very general More generally, **DTIME**$(f(n)) =$ **NTIME**$(g(n)) \Rightarrow$ **DTIME**$(f(h(n))) =$ **NTIME**$(g(h(n)))$ for time constructible f, g, h.

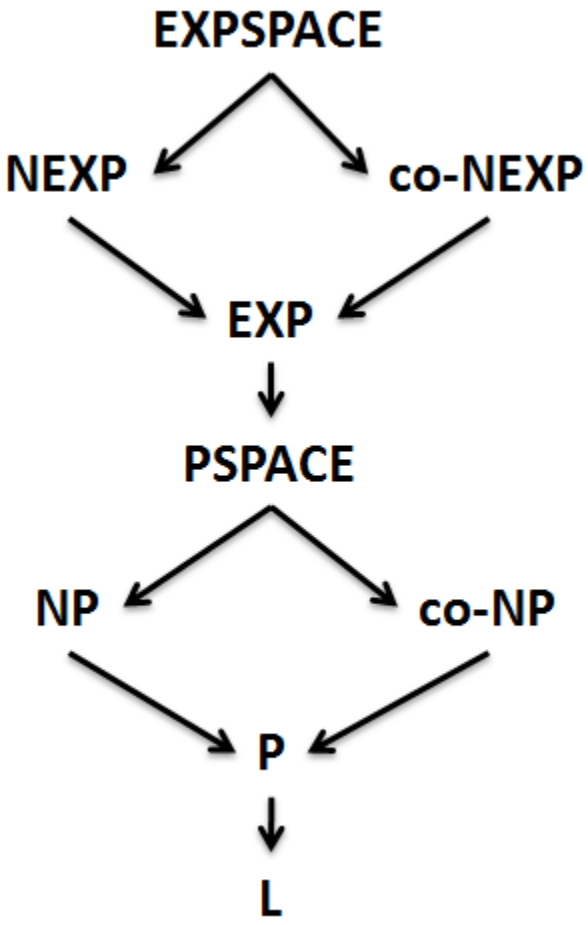

Figure 1: The complexity hierarchy. Beyond these separations questions of equivalence are open.

1.3 TM-specific results

Some slightly stronger results are known about fine relations between resources for the Turing Machine model:

$\mathbf{DTIME}(f(n)) \subseteq \mathbf{SPACE}(\frac{f(n)}{log(f(n))}) \subsetneq \mathbf{SPACE}(f(n))$ [Hopcroft-Paul-Valiant]

$\mathbf{DTIME}(f(n)) \subsetneq \mathbf{NTIME}(f(n))$ [Paul-Pippenger-Szemeredi-Trotter]

At first, the latter result seems quite close to the **P** vs. **NP** question, but it really only gives a tiny separation between deterministic and nondeterministic time (something like $\log^* n$), so the gap disappears as soon as we allow polynomial slackness (as in **P** vs. **NP**). These small gaps are also the reason that the results are model-dependent.

2 Review of NP, P vs. NP

Proposition 3 *$L \in \mathbf{NP}$ if and only if there exists a poly-time TM M and a polynomial p s.t. $x \in L \Leftrightarrow \exists u \in [0,1]^{p(n)} M(x,u) = 1$.*

Where M may be called the "verifier," and u is commonly referred to as the "witness," "certificate," "proof," or "solution."

If $\mathbf{P} = \mathbf{NP}$, the following are easy:

- search problems (finding a witness)
- optimization
- machine learning
- breaking cryptosystems
- finding mathematical proofs (finding them becomes polynomial in their length in a formal system)

Whereas if $\mathbf{P} \neq \mathbf{NP}$ every **NP**-hard problem $\notin \mathbf{P}$.

3 Reductions

$A \leq B$: is meant to capture that computational problem A is easier (no harder) than B. A and B might be languages, functions, or even search problems (where there is a *set* of valid answers on each input, see PS0).

Cook reduction ($A \leq_C B$): A can be solved in polynomial time given an "oracle" for B*.i.e.,* There exists a poly time "oracle TM" such that for every x, $M^B(x) = A(x)$. M can query oracle B on any input q; it writes and receives $B(q)$ back in one step (possibly on an oracle tape). In case B is a search problem, we require that $M^O(x) = A(x)$ for every function O that solves B (i.e. $O(q) \in B(q)$ for all q).

Otherwise known as "poly-time Turing" or "poly-time oracle" reductions.

Karp reduction ($A \leq_p B$): Applies when A and B are languages. Requires that there exists a poly-time computable function f such that $x \in A \Leftrightarrow f(x) \in B$.

Otherwise known as "poly-time mapping" or "poly-time many-one" reductions.

Logspace mapping reduction Analogous to a Karp reduction, but where f is computable in log space.

Some desiderata when considering reductions:

- Reductions should be transitive ($A \leq B, B \leq C \Rightarrow A \leq C$).
- If B is "easy," A should be "easy" in the same sense:

  for "easy" = **P**, Cook reductions suffice.

  for "easy" = **NP**, Karp is appropriate since **NP** is not closed under complement (**NP** $\stackrel{?}{=}$ **co-NP**).

  for "easy" = **L**, logspace reductions are suitable.

3.1 NP-completeness

Let **C** be a class of computational problems, B a computational problem, and $\leq_x$ a class of reductions.

C-hard B is **C**-*hard with respect to* $\leq_x$ if for every $A \in \mathbf{C}$, we have $A \leq_x B$

C-complete B is **C**-*complete with respect to* $\leq_x$ if

[1] $B \in \mathbf{C}$

[2] B is **C**-hard with respect to $\leq_x$.

Circuit satisfiability is a canonical **NP**-complete problem, and Boolean circuit size has a close relationship with the time a Turing machine requires to compute a problem.

Boolean circuit a Boolean circuit of n inputs, m outputs, and a "fan-in" of k is a directed acyclic graph with n sources (inputs without incoming edges), m sinks (outputs without outgoing edges) and all non-source vertices v labelled with a Boolean function f_v of arity equal to the indegree of v, which is required to be at most k.

Unless otherwise stated, $k = 2$.

The "size" of a Boolean circuit is the number of nodes in it.

Boolean circuits naturally define mappings $C : \{0,1\}^n \Rightarrow \{0,1\}^m$.

Theorem 4 *Let M be a time $t(n)$ TM, where $\lceil \log t(n) \rceil$ is space constructible. There exists a sequence, $C_1, C_2, \ldots$ of Boolean circuits, where C_n has n inputs and $t(n)$ outputs, such that:*

1. $\forall x \in \{0,1\}^n,\ C_n(x) = M(x)$ *(where $M(x)$ may be padded to $t(n)$ bits)*
2. $|C_n| = \text{poly}(t(n)) =: S(n)$ *(this bound may be improved to $S(n) = O(t(n) \cdot \log(t))$, see below)*

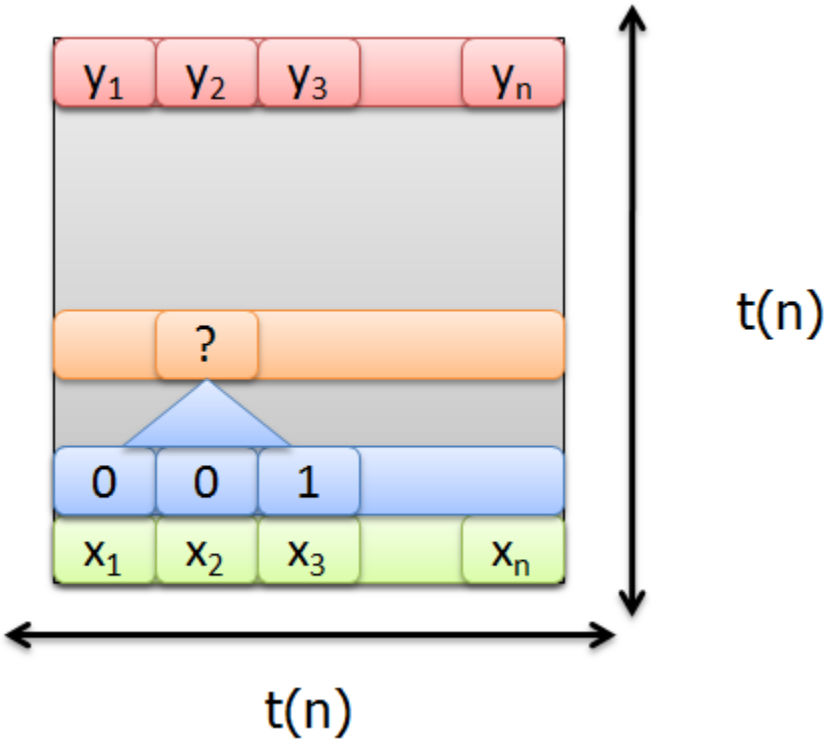

Figure 2: The tableau of a Turing machine computing a Boolean circuit. The Turing machine mimics the circuit's computation iteratively, using only local results and the circuit's Boolean function to determine the next value.

3. *("uniformity") There exists a TM M', running in time $O(S(n))$ and space $O(\log S(n))$ s.t. $M'(1^n) = \lfloor C_n \rfloor$.*

Proof: The "sloppy" or "loose" version is sketched here. This version demonstrates a polynomial bound on the circuit size. A discussion follows about reducing this.

For a single tape TM, consider the tableau of M's computation on inputs of length n (see Figure 2). Each cell will be represented by a constant number of gates in the circuit (which encode the tape cell contents, whether or not the head is present at that cell, and the state of the TM if the head is present). Then we can create subcircuits, f_M, which are constant-sized functions ensuring correctness of the TM computation based on the transition function: each cell is determined by three cells in the row immediately below it. Any Boolean function, and in particular f_M can be computed by a C_M of constant size.

Thus, circuit C_n basically consists of many copies of C_M, so M' simply keeps track of the row and column when generating C_n. $T(n)$ must be constructible in order to determine the dimensions of the tableau. ■

The above results in a blow-up of $(t(n)^2)^2$. The first squaring is the conversion to a single tape machine, and the second is due to the square tableau. By moving to a multi-tape machine and making the TM oblivious, both may be eliminated. However, the transition to an oblivious TM requires a logarithmic slowdown, and so the final blow-up is $(t(n) \cdot \log(t(n)))$ (details are found in the book).