Lecture 18

Time-bounded computations

We now begin the final part of the course, which is on *complexity theory*. We'll have time to only scratch the surface—complexity theory is a rich subject, and many scientists around the world are engaged in a study of this field. Unlike formal language theory and computability theory, many of the central questions of complexity theory remain unanswered to this day.

The motivation for the subject of this lecture, which is on time-bounded computation, is simple: it is usually not enough to know that a language is decidable (or that a function is computable)—we need computations to be *efficient*. For instance, if we have a particular computational task that we would like to perform, and someone gives us a computational device that will complete this task only after running for one million years, it is practically useless.

It is typical that efficiency is equated with running time, and this is why we begin by considering the amount of time needed to perform computations. It is, however, possible to consider other notions of efficiency, by referring to things like memory usage, communication (in a distributed setting), power consumption, or a variety of other notions concerning resource usage.

18.1 Time complexity

We will start with a definition of the running time of a DTM, assuming that this DTM never runs forever.

Definition 18.1. Let M be a DTM with input alphabet Σ that halts on every input string $w \in \Sigma^*$. The *running time* of M is the function $t : \mathbb{N} \to \mathbb{N}$ defined as follows for each $n \in \mathbb{N}$:

$$t(n) = \begin{cases} \text{the maximum number of steps required for } M \\ \text{to halt, over all inputs } w \in \Sigma^* \text{ with } |w| = n. \end{cases} \tag{18.1}$$

One can define the running time for any Turing machine variant, but we will focus primarily on ordinary (one-tape) DTMs.

Deterministic time complexity classes

Next, for every function $f : \mathbb{N} \to \mathbb{N}$, we define a class of languages $\mathrm{DTIME}(f)$ representing those languages decidable in time $O(f(n))$.

Definition 18.2. Let $f : \mathbb{N} \to \mathbb{N}$ be a function. A language A is contained in the class $\mathrm{DTIME}(f)$ if there exists a DTM M that decides A and whose running time t satisfies $t(n) = O(f(n))$.

We define $\mathrm{DTIME}(f)$ in this way, using $O(f(n))$ rather than $f(n)$, because we are generally not interested in constant factors or in what might happen in finitely many special cases. One fact that motivates this choice is that it is usually possible to "speed up" a DTM by defining a new DTM, having a larger tape alphabet than the original, that succeeds in simulating multiple computation steps of the original DTM with each step it performs.

When it is reasonable to do so, we use the variable name n to refer to the input length for whatever language or DTM we are considering. So, for example, we may refer to a DTM that runs in time $O(n^2)$ or refer to the class of languages $\mathrm{DTIME}(n^2)$ with the understanding that we are speaking of the function $f(n) = n^2$, without explicitly saying that n is the input length.

We also sometimes refer to classes such as

$$\mathrm{DTIME}\big(n\sqrt{n}\big) \quad \text{or} \quad \mathrm{DTIME}\big(n^2 \log(n)\big), \tag{18.2}$$

where the function f that we are implicitly referring to appears to take non-integer values for some choices of n. This is done in an attempt to keep the expressions of these classes simple and intuitive, and you can interpret these things as referring to functions of the form $f : \mathbb{N} \to \mathbb{N}$ obtained by rounding up to the next positive integer. For instance, $\mathrm{DTIME}(n^2 \log(n))$ means $\mathrm{DTIME}(f)$ for

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ \lceil n^2 \log(n) \rceil & \text{otherwise.} \end{cases} \tag{18.3}$$

Because the definition of $\mathrm{DTIME}(f)$ refers only to $O(f(n))$ and not $f(n)$, you could just as easily modify Definition 18.2 to allow f to be a real-valued function—the resulting complexity class $\mathrm{DTIME}(f)$ would be the same for the examples mentioned above.

Example 18.3. The language $A = \{0^m 1^m : m \in \mathbb{N}\}$ is contained in DTIME(n^2). The example of a DTM for deciding this language from Lecture 12, for instance, runs in time $O(n^2)$ on inputs of length n.

It is, in fact, possible to do better: it is the case that $A \in \text{DTIME}(n \log(n))$. To see that A is contained in DTIME$(n \log(n))$, consider a DTM that repeatedly "crosses out" every other symbol on the tape, and compares the parity of the number of 0s and 1s crossed out after each pass over the input. Through this method, A can be decided in time $O(n \log(n))$ by making just a logarithmic number of passes over the portion of the tape initially containing the input.

After considering the previous example, it is natural to ask if one can do even better than $O(n \log(n))$ for the running time of a DTM deciding the language $A = \{0^m 1^m : m \in \mathbb{N}\}$. The answer is that this is not possible. This is a consequence of the following theorem (which we will not prove).

Theorem 18.4. *Let A be a language. If there exists a DTM M that decides A in time $o(n \log(n))$, meaning that the running time t of M satisfies*

$$\lim_{n \to \infty} \frac{t(n)}{n \log(n)} = 0, \tag{18.4}$$

then A is regular.

It is, of course, critical that we understand the previous theorem to be referring to ordinary, one-tape DTMs. With a two-tape DTM, for instance, it is easy to decide some nonregular languages, including $\{0^m 1^m : m \in \mathbb{N}\}$, in time $O(n)$.

Time-constructible functions

The complexity class DTIME(f) has been defined for an arbitrary function of the form $f : \mathbb{N} \to \mathbb{N}$, but there is a sense in which most functions of this form are uninteresting—because they have absolutely nothing to do with the running time of any DTM.

There are, in fact, some choices of functions $f : \mathbb{N} \to \mathbb{N}$ that are so strange that they lead to highly counter-intuitive results. For example, there exists a function f such that

$$\text{DTIME}(f) = \text{DTIME}(g), \qquad \text{for } g(n) = 2^{f(n)}; \tag{18.5}$$

even though g is exponentially larger than f, they both result in exactly the same deterministic time complexity classes. This doesn't necessarily imply anything important about time complexity, it's more a statement about the strangeness of the function f.

For this reason we define a collection of functions, called *time-constructible functions*, that represent well-behaved upper bounds on the possible running times of DTMs. Here is a precise definition.

Definition 18.5. Let $f : \mathbb{N} \to \mathbb{N}$ be a function satisfying $f(n) = \Omega(n \log(n))$. The function f is said to be *time constructible* if there exists a DTM M that operates as follows:

1. On each input 0^n the DTM M computes the binary representation of $f(n)$, for every $n \in \mathbb{N}$.
2. M runs in time $O(f(n))$.

It might not be clear why we would define a class of functions in this particular way, but the essence is that these are functions that can serve as upper bounds for DTM computations. That is, a DTM can compute $f(n)$ on any input of length n, and doing this doesn't take more than $O(f(n))$ steps—and then it has the number $f(n)$ written in a convenient form so that it could use this number to limit some subsequent part of its computation (perhaps the number of steps for which it runs during a second phase of its computation).

As it turns out, just about any reasonable function f with $f(n) = \Omega(n \log(n))$ that you are likely to care about as a bound on running time is time constructible. Examples include the following:

1. For any choice of an integer $k \geq 2$, the functions

$$f(n) = n^k \qquad \text{and} \qquad f(n) = k^n \tag{18.6}$$

   are time constructible.

2. For any choice of an integer $k \geq 1$, the functions

$$f(n) = \begin{cases} 0 & \text{if } n = 0 \\ \lceil n^k \log(n) \rceil & \text{otherwise} \end{cases} \tag{18.7}$$

   and

$$f(n) = \lceil n^k \sqrt{n} \rceil \tag{18.8}$$

   are time constructible.

3. If f and g are time-constructible functions, then the functions

$$h_1(n) = f(n) + g(n), \quad h_2(n) = f(n)g(n), \quad \text{and} \quad h_3(n) = f(g(n)) \tag{18.9}$$

   are also time constructible.

18.2 The time-hierarchy theorem

What we will do next is to discuss a fairly intuitive theorem concerning time complexity. A highly informal statement of the theorem is this: more languages can be decided with more time. While this is indeed an intuitive idea, it is not obvious how a formal version of this statement is to be proved. We will begin with a somewhat high-level discussion of how the theorem is proved, and then state the strongest-known form of the theorem (without going through the low-level details needed to obtain the stronger form).

We will restrict our attention to ordinary one-tape DTMs whose input alphabet is $\Sigma = \{0, 1\}$, and let us suppose that we have fixed an encoding scheme over Σ for DTMs having just this particular input alphabet. That is, for each DTM M having input alphabet Σ, the encoding $\langle M \rangle$ is a string over the alphabet Σ. (We're free to use an encoding scheme for all DTMs if we want, but we will regard encodings of DTMs having input alphabets differing from Σ as if they are invalid encodings.)

Now, suppose that a time-constructible function $f : \mathbb{N} \to \mathbb{N}$ has been selected, and define a DTM K as follows:

On input $w = \langle M \rangle 01^k$, where M is a DTM with input alphabet Σ and $k \in \mathbb{N}$:

1. Compute $t = f(|w|)$.
2. Simulate M on input w for t steps.
3. If M has rejected during this simulation, then *accept*, otherwise *reject*.

It is not immediately clear what the running time is for K, because that depends on precisely how the simulation of M is done—different ways of performing the simulation could of course lead to different running times. For the time being, let us take $g : \mathbb{N} \to \mathbb{N}$ to be the running time of K, and we'll worry later about how specifically g relates to f.

Next, let us think about the language $\mathrm{L}(K)$ decided by K. It is obvious that $\mathrm{L}(K) \in \mathrm{DTIME}(g)$, because K itself is a DTM that decides $\mathrm{L}(K)$ in time $g(n)$. What we will show is that $\mathrm{L}(K)$ cannot possibly be decided by a DTM that runs in time $o(f(n))$.

Assume toward contradiction that there does exist a DTM M that decides $\mathrm{L}(K)$ in time $o(f(n))$. Because the running time of M is $o(f(n))$, we know that there must exist a natural number n_0 such that, for all $n \geq n_0$, the DTM M halts on all inputs of length n in strictly fewer than $f(n)$ steps. Choose k to be large enough so that the string $w = \langle M \rangle 01^k$ satisfies $|w| \geq n_0$, and (as always) let $n = |w|$. Because M halts on input w after fewer than $f(n)$ steps, we find that

$$w \in \mathrm{L}(K) \;\Leftrightarrow\; w \notin \mathrm{L}(M). \tag{18.10}$$

The reason is that K simulates M on input w, it completes the simulation because M runs for fewer than $f(n)$ step, and it answers *opposite* to the way M answers (i.e., if M accepts, then K rejects; and if M rejects, then K accepts). This contradicts the assumption that M decides $\mathrm{L}(K)$. We conclude that no DTM whose running time is $o(f(n))$ can decide $\mathrm{L}(K)$.

It is natural to wonder what the purpose is for taking the input to K to have the form $\langle M \rangle 01^k$, as opposed to just $\langle M \rangle$ (for instance). The reason is pretty simple: it's just a way of letting the length of the input string grow, so that the asymptotic behavior of the function f and the running time of M take over (even though we're really interested in fixed choices of M). If we were to change the language, so that the input takes the form $w = \langle M \rangle$ rather than $\langle M \rangle 01^k$, we would have no way to guarantee that K is capable of finishing the simulation of M on input $\langle M \rangle$ within $f(|\langle M \rangle|)$ steps—for it could be that the running time of M on input $\langle M \rangle$ exceeds $f(|\langle M \rangle|)$ steps, even though the running time of M is small compared with f for significantly longer input strings.

What we have proved is that for any choice of a time-constructible function $f : \mathbb{N} \to \mathbb{N}$, it holds that

$$\mathrm{DTIME}(h) \subsetneq \mathrm{DTIME}(g) \tag{18.11}$$

whenever $h(n) = o(f(n))$, where g is the running time of K (which depends somehow on f). If you work very hard to make K run as efficiently as possible, the following theorem can be obtained.

Theorem 18.6 (Time-hierarchy theorem)**.** *If $f, g : \mathbb{N} \to \mathbb{N}$ are time-constructible functions for which it holds that $f(n) = o(g(n)/\log(g(n)))$, then*

$$\mathrm{DTIME}(f) \subsetneq \mathrm{DTIME}(g). \tag{18.12}$$

The main reason that we will not go through the details required to prove this theorem is that optimizing K to simulate a given DTM as efficiently as possible gets very technical. For the sake of this course, it is enough that you understand the basic idea of proving this theorem through the diagonalization technique (along similar lines to the proof that DIAG is not Turing-recognizable, and to the proof that $\mathcal{P}(\mathbb{N})$ is uncountable).

From the time-hierarchy theorem, one can conclude the following down-to-earth corollary.

Corollary 18.7. *For all $k \in \mathbb{N}$ with $k \geq 1$, it holds that*

$$\mathrm{DTIME}(n^k) \subsetneq \mathrm{DTIME}(n^{k+1}). \tag{18.13}$$

18.3 Polynomial and exponential time

We'll finish off the lecture by introducing a few important notions based on deterministic time complexity.

First, let us define two complexity classes, known as P and EXP, as follows:

$$\mathrm{P} = \bigcup_{k \geq 1} \mathrm{DTIME}(n^k) \quad \text{and} \quad \mathrm{EXP} = \bigcup_{k \geq 1} \mathrm{DTIME}\bigl(2^{n^k}\bigr). \tag{18.14}$$

In words, a language A is contained in the complexity class P if there exists a DTM M that decides A and has *polynomial running time*, meaning a running time that is $O(n^k)$ for some choice of $k \geq 1$; and a language A is contained in the complexity class EXP if there exists a DTM M that decides A and has *exponential running time*, meaning a running time that is $O\bigl(2^{n^k}\bigr)$ for some choice of $k \geq 1$.

As a very rough but nevertheless useful simplification, we often view the class P as representing languages that can be *efficiently* decided by a DTM, while EXP contains languages that are decidable by a *brute force* approach. These are undoubtedly over-simplifications in some respects, but for languages that correspond to "natural" computational problems that arise in practical settings, this is a reasonable picture to keep in mind. We'll have more to say about these (and other) complexity classes in the next couple of lectures.

By the time-hierarchy theorem, it holds that $\mathrm{P} \subsetneq \mathrm{EXP}$. In particular, if we take $f(n) = 2^n$ and $g(n) = 2^{2n}$, then the time hierarchy theorem establishes the middle (proper) inclusion in this expression:

$$\mathrm{P} \subseteq \mathrm{DTIME}(2^n) \subsetneq \mathrm{DTIME}(2^{2n}) \subseteq \mathrm{EXP}. \tag{18.15}$$

A simple example of a language in the class P is the graph reachability problem from Lecture 14:

$$\left\{ \langle G, u, v \rangle \,:\, \begin{array}{l} \text{there exists a path from vertex } u \text{ to} \\ \text{vertex } v \text{ in the undirected graph } G \end{array} \right\}. \tag{18.16}$$

The DTM for this language that we discussed in that lecture decides the language in polynomial time (assuming we use any of the graph encoding schemes mentioned in Lecture 13, or something similar). Another example is this language:

$$\left\{ \langle M, w, 0^t \rangle \,:\, \begin{array}{l} M \text{ is a DTM, } w \text{ is a string, } t \in \mathbb{N}, \\ \text{and } M \text{ accepts } w \text{ within } t \text{ steps} \end{array} \right\}. \tag{18.17}$$

This language is similar to one described in Lecture 15, except that the number of steps t for which the DTM M is to run is input in unary. (If t was given in

binary, we would have a language in EXP but not in P.) Finally, it is interesting to note that every context-free language is in P, although we will not have time to discuss a method through which to prove this. There are of course many, many other interesting examples of languages contained in P—the ones just mentioned represent only a few examples.

Finally, let us observe that one may consider not only languages that are decided by DTMs having bounded running times, but also functions that can be computed by time-bounded DTMs. It will be enough for the purposes of the remaining lectures of this course to consider the class of *polynomial-time computable functions*, which are functions that can be computed by a DTM with running time $O(n^k)$ for some fixed positive integer k.

It will be convenient for the purposes of the next few lectures that we extend the notion of computable functions slightly by allowing for functions having different input and output alphabets. In particular, if Σ and Γ are (possibly different) alphabets, then we say that a function of the form

$$f : \Sigma^* \to \Gamma^* \tag{18.18}$$

is computable if there exists a DTM $M = (Q, \Sigma, \Delta, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$ whose yields relation satisfies

$$(q_0, \sqcup)\, w \vdash_M^* (q_{\text{acc}}, \sqcup)\, f(w) \tag{18.19}$$

for every string $w \in \Sigma^*$. The only change in the definition from the one in Lecture 16 is that we do not require that $f(w)$ is a string over the input alphabet of M. (Naturally, the tape alphabet Δ of M must include the symbols in Γ that appear in any string in the range of f in order for this relation to be satisfied.)