Lecture #9: Nondeterministic Time — Speedup, Emulation, and a Nondeterministic Time Hierarchy Theorem

Proof of the Nondeterministic Time Hierarchy Theorem

The goal of this document — which is ***for interest only*** (and not required reading) is to present a proof of the following.

Theorem (Nondeterministic Time Hierarchy Theorem)**.** *Let $f, g : \mathbb{N} \rightarrow \mathbb{N}$ be time constructible functions such that f and g are nondecreasing functions, $f(n) \geq n$ for all $n \in \mathbb{N}$, and $g(n+1) \in o(f(n))$. Then*

$$\text{NTIME}(g(n)) \subsetneq \text{NTIME}(f(n)).$$

A similar result — the "Deterministic Time Hierarchy Theorem" was proved using a Diagonalization argument, considering an algorithm that simulated an execution on an encoded Turing machine and then flipping the answer (by exchanging accepting and rejecting states) — so that a contradiction can be obtained when considering what happens when this algorithm receives (as input) an encoding of a Turing machine implementing the same algorithm.

This does not work, here, because exchanging accepting and rejecting states *does not*, necessarily, change the outcome of a nondeterministic computation.

In order to deal with, a technique sometimes called "lazy diagonalization" is introduced and used: A language $L_D \subseteq \{\mathtt{1}\}^\star$ is introduced, whose inputs can be thought of as *unary* representations of indices in a listing of nondeterministic Turing machines.

- The language is defined in such a way that, if $\text{NTIME}(g(n)) = \text{NTIME}(f(n))$, then it must be true that either

  $$\mathtt{1}^\ell, \mathtt{1}^{\ell+1}, \mathtt{1}^{\ell+2}, \ldots, \mathtt{1}^r$$

  must all belong to L_D, or none of these values belong to L_D, for positive integers ℓ and r.
- However, r is so much larger than ℓ that it is possible to deterministically check whether $\mathtt{1}^\ell$ belongs to L_D when $\mathtt{1}^r$ is given as input — and this can to be used to make sure that *exactly one* of $\mathtt{1}^\ell$ and $\mathtt{1}^r$ belongs to L_D.
- Since these statements cannot both hold, a contradiction is obtained, as needed to establish the complexity classes $\text{NTIME}(g(n))$ and $\text{NTIME}(f(n))$ must be different.

The Nondeterministic Time Hierarchy Theorem refers to two time-constructible functions f and g. A third function, $h : \mathbb{N} \to \mathbb{N}$, is also required to prove this result — it is used to define the values "ℓ" and "r" in the above summary.

Definition 1. Suppose that $f : \mathbb{N} \to \mathbb{N}$ is the time-constructible function mentioned in the Nondeterministic Time Hierarchy Theorem. Let $h : \mathbb{N} \to \mathbb{N}$ be defined as follows:

- $h(0) = 2$, and
- if $i \geq 0$ then $h(i+1) = c^{f(h(i))}$ where

$$c = \begin{cases} 2 & \text{if } i \leq 2, \text{ and} \\ 2^{\lceil \log_2 i \rceil} & \text{otherwise.} \end{cases}$$

Since $f(n) \geq n$ for all $n \in \mathbb{N}$, h is certainly a strictly increasing function of n. Indeed, it grows so rapidly that proving the following claim is not trivial.

Lemma 2. *Given a unary representation of a positive integer n such that $n \geq 3$, it is possible to compute the binary representation of the unique integer i such that $h(i) < n \leq h(i+1)$, deterministically, using a number of steps that is at most linear in $f(n)$.*

Proof. To begin, notice that $h(i)$ is a power of two for every integer $i \geq 0$, so that $h(i) < n \leq h(i+1)$ if and only if $\log_2 h(i) < \lceil \log_2 n \rceil \leq \log_2 h(i+1)$ — and there is only one integer i that satisfies this constraint. It is therefore sufficient to check this condition.

Notice, as well, that it follows by the above definition that $\log_2 h(0) = 1$ and, for $i \geq 1$,

$$\log_2 h(i+1) = \begin{cases} f(h(i)) & \text{if } i \leq 2, \\ \lceil \log_2 i \rceil \cdot f(h(i)) & \text{if } i > 2. \end{cases} \tag{1}$$

Now consider the algorithm shown in Figure 1. This computes binary representations of values $h_i = h(i)$ and $j_i = f(h(i))$ as part of its processing.

To see that this algorithm computes the integer i such that $h(i) < i \leq h(i+1)$, note that the integer d computed at lines 6–8 is equal to $\log_2 c$, where c is the integer used in the recursive definition of $h(i+1)$ from $h(i)$ — so that

$$\log_2 h(i+1) = d \times f(h(i)) = d \times j_i$$

if $h_i = h(i)$ and $j_i = f(h_i) = f(h(i))$.

Since $(\log_2 h(i+1))/d \in \mathbb{N}$, this implies that the test at line 9 passes if and only if i is the smallest integer such that $\log_2 n \leq \log_2 h(i+1)$, that is, if and only if $h(i) < n \leq h(i+1)$, as required. It also implies that $h_{i+1} = h(i+1)$ is correctly computed at line 11.

On input 1^n where $n \geq 3$:

1. Compute the binary representation of n. Then compute the binary representation of $\lceil \log_2 n \rceil$ — the length of the binary representation of n if n is not a power of two and one less than this, otherwise.
2. i := 0; h_i := 2
3. `while (true) {`
4.   Use the binary representation of h_i to compute the unary representation 1^{h_i} of h_i.
5.   Use the unary representation of h_i to compute the binary representation of $j_i = f(h_i)$.
6.   `if` $(i \leq 2)$ `{`
7. d := 1

    `} else {`

8. d := $\lceil \log_2 i \rceil$

    `}`

9.   `if` $(\lceil \lceil \log_2 n \rceil / d \rceil \leq j_i)$ `{`
10.   `return` i

    `}`

11.   Compute the binary representation of $h_{i+1} = 2^{d \times j_i}$
12. i := $i + 1$

  `}`

Figure 1: Computation of Integer i Such That $h(i) < n \leq h(i)$

Now consider the time used to execute this algorithm. Steps 1 and 2 can be carried out using $O(n)$ steps and this is certainly in $O(f(n))$.

To continue, let us consider the cost to execute the loop at lines 3–12. Let us consider the (total) cost of all but the *final* execution of the loop body — so that i is too small, and $n > h(i+1)$. To begin, consider a single execution of the loop body when this is the case.

- Step 4 can be carried out using $O(h(i))$ steps.
- Since f is time-constructible, step 5 can be carried out using $O(f(h(i))$ steps.
- Steps 6–8 can be carried out using $O(i)$ steps, and this is certainly in $O(h(i))$.
- The computation of $\lceil \lceil \log_2 n \rceil / d \rceil$, required for step 9, requires $O((\log n)^2)$ steps. Once a binary representation of $\lceil \lceil \log_2 n \rceil / d \rceil$ is available, the rest of step 9 can be completed using $O(\log_2 f(h(i)))$ steps.
- Step 11 requires the multiplication of a pair of integers whose binary representations each have length in $O(\log_2 h(i+1))$ to compute the binary representation of $h(i+1)$ —

followed by conversion from a binary representation of this value to a unary representation of it.

This can be carried out using $O(h(i+1))$ steps.

- It follows that the cost of everything except the computation of $\lceil \lceil \log_2 n \rceil / d \rceil$ has cost in $O(h(i+1))$.

This allows us to bound the total cost of all these executions of the loop body as follows.

- Note next that h is growing *at least* exponentially with its input — and it is not hard to use this to argue that

$$\sum_{j=1}^{i} h(j) \in O(h(i)).$$

  Thus the *total* cost of all these operations is in $O(h(i+1))$ and this is in $O(n)$, since $h(i+1) \leq n$ for the value of i currently being considered.

- Since h is growing at least exponentially with its input there are $O(\log_2 n)$ executions of step 9, so the total cost of all computations of $\lceil \lceil \log_2 n \rceil / d \rceil$, for various values of d, is in $O((\log_2 n)^3) \subseteq O(n)$.
- It now follows that the total cost of all steps except those in the *last* execution of the loop body is in $O(n)$, and this certainly in $O(f(n))$.

Now consider the cost of the steps in the *last* execution of the body of the loop.

- Since $h_i = h(i) < n$, step 4 can be carried out using $O(n)$ steps.
- Since f is time-constructible and nondecreasing, step 5 can be carried out using $O(f(h(i))$ steps and, since $h(i) \leq n$, this is in $O(f(n))$.

  It follows that binary representation of $j_i = f(h(i))$ has length in $O(f(n))$.

- Thus — since $\lceil \lceil \log_n \rceil / d \rceil$ can be computed cheaply and certainly has length in $O(f(n))$ too, the test at line 9 can also be carried out using $O(f(n))$ steps.
- The `return` statement is certainly inexpensive and is the last step executed.
- Thus the cost of the final execution of the body of the loop is in $O(f(n))$ as well, as required to establish the claim. □

To continue, we will try to use ***diagonalization*** to construct some language L_D, that is in NTIME$(f(n))$ but not in NTIME$(g(n))$. This is complicated because we cannot just "flip" the answer for a nondeterministic computation, like we can with deterministic computation — this would generally still have strings being accepted by nondeterministic Turing machines when we do not want them to be (why?).

The trick here is to use a technique sometimes ***lazy diagonalization***. Rather than ensuring that the wrong answer is given by a too-fast machine on a specific input it suffices to ensure that it must make a mistake on one of a large *set* of input strings instead.

With that noted, the language $L_D \subseteq \{1\}^\star$ that is decided by a nondeterministic Turing machine implementing the algorithm shown in Figure 2 on page 6.

Lemma 3. $L_D \in \mathsf{NTIME}(f(n))$.

Proof. Consider the nondeterministic algorithm in Figure 2, on page 6, which decides L_D.

- It follows by Lemma 2 that the integer i can be discovered in time $O(f(n))$. Since i is logarithmic in n it is not hard to show that the unpadded encoding $\mu_i \in \Sigma_{\mathsf{UTM}}^\star$, of the nondeterministic Turing machine M_i, can be computed using time in $O(n) \subseteq O(f(n))$ as well.
- If "tapes for a future simulation" were set up when step 1 was carried out then step 2 can be completed in constant time.
- Since c is so small, a binary representation of c^2 can certainly be computed using deterministic time in $O(n)$. Step 3 can be carried out using a linear sweep over the unpadded encoding μ_i of M_i — whose length is in $O(\log_2 n)$ — so this step can certainly be carried out using time in $O(n) \subseteq f(n)$ too.
- Since f is a time-constructible function step 4 can be carried out — deterministically — using $O(f(n))$ steps too.
- It is certainly easy to check the test at line 5 using time in $O(n) \subseteq O(f(n))$.
- The value ℓ, mentioned in step 7, is computed "along the way" when i is computed in step 1. If it is remembered, at this point, then step 7 can be carried out using a number of steps in $O(\log n)$, since the length of a binary representation of ℓ is not significantly longer than $\log_2 n$.
- Finally, the time needed to carry out either of steps 6 or 8 can be shown to be in $O(f(n))$ because of the use of the binary counter to terminate simulations if they would need *more than* $f(n)$ steps.

Thus this algorithm uses $O(f(n))$ steps in the worst case. By definition, it accepts every string in L_D and it rejects every string in $\{1\}^\star$ that is not in L_D, as needed to establish the claim. $\square$

The proof of the following is the place where ***lazy diagonalization*** is being employed: The analysis at the end implies that any "too-fast" nondeterministic Turing machine, accepting a

On input $\mathtt{1}^n$ where $n \geq 3$:

1. Compute the integer $i \geq 0$ such that $h(i) < n \leq h(i+1)$ — remembering the positive integer c such that $h(i+1) = c^{f(h(i))}$. Let

$$M_i = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

   and let k be the number of tapes used by M_i. Make sure that the (unpadded) encoding $\mu_i \in \Sigma^{\star}_{\text{UTM}}$ of M_i has been written onto a tape for later use.

2. `if` $(|\Sigma| \neq 1)$ `{ reject }`
3. If there exists a state $q \in Q$ and symbols $\sigma_1, \sigma_2, \ldots, \sigma_k \in \Gamma$ such that

$$|\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)|^2 \geq c$$

   then ***reject***.

4. Compute the binary representation of $f(n)$ and initialize a binary counter to have this value.
5. `if` $(n < h(i+1))$ `{`
6. Apply the nondeterministic universal Turing machine described in the lecture notes, with an encoding of M_i, $\mathtt{1}^{n+1}$, and $1.5 \times g(n+1)$ as input, to *try to* decide whether there is an accepting computation for M_i on input $\mathtt{1}^{n+1}$ with length at most $1.5 \times g(n+1)$ — but using the binary counter to keep track of the number of steps taken ***by this nondeterministic universal Turing machine***.

   If the nondeterministic universal Turing machine has still not halted after at most $f(n)$ of its own steps then ***reject***.

   Otherwise ***accept*** if the nondeterministic universal Turing machine has accepted its input, and ***reject*** otherwise.

   `} else {` // *Note:* $n = h(i+1)$.

7. $\ell := h(i)$
8. *Try to use exhaustive search* — by checking *every* possible sequence of guessed moves to determine whether there exists an accepting computation of M_i on input $\mathtt{1}^{\ell+1}$ with length at most $1.5 \times g(\ell+1)$ — but using the binary counter from step 4 to make sure that this simulation does not include more than $f(n)$ steps.

   If the simulation has still not halted after $f(n)$ steps then ***reject***. If it halted and an accepting computation with length at most $g(\ell+1)$ was found then ***reject***. Otherwise ***accept***.

   `}`

Figure 2: Algorithm Deciding a Language L_D

language in $\{1\}^\star$, must make a mistake about membership about membership of 1^n in L_D, for some value n such that $h(i)+1 \le n \le h(i+1)$, for some $i \in \mathbb{N}$.

Lemma 4. *$L_D \notin$ NTIME$(g(n))$.*

Proof. By contradiction. Suppose $L_D \in$ NTIME$(g(n))$.

- Then there exists a nondeterministic Turing machine $\widehat{M}$ that decides L_D using time in $O(g(n))$.
- The Nondeterministic Linear Speedup Theorem, given in the notes for Lecture #9, can be used to conclude that there is a nondeterministic Turing machine M, that decides L, such that the computation tree for M on an input $\omega \in \{1\}^\star$ has depth at most $1.5 \cdot g(n)$, when $n = |\omega|$, for every nonempty string $\omega \in \{1\}^\star$.
- Consider what happens when the algorithm in Figure 2, that decides L_D, is applied to a string 1^n for which the corresponding nondeterministic Turing machine M_i (considered in the algorithm) corresponds to a sufficiently long ***padded*** encoding of M.
- The algorithm does not reject 1^n at step 2, because corresponding nondeterministic Turing machine M_i has an input alphabet with size one.
- For $n \ge 3$ let c_n be the value for "c" at step 1. Even though it grows extremely slowly, it is possible to argue that
$$\lim_{n \to +\infty} c_n = +\infty.$$
  Since the maximum size of any set included in M's transition function is a constant (not depending on the length of a padded encoding of M) this can be used to argue that for sufficiently large integer n, with the corresponding nondeterministic Turing machine M_n being a padded version of M, 1^n will not be rejected at line 3, either.
- Consider the use of the nondeterministic Turing machine in step 6, if this is reached and executed. Recall that the number of steps used here, to simulate each move of M, depends only on M (and not the length of its padded encoding).

  Since $g(n+1) \in o(f(n))$ this implies that — for sufficiently large n — the binary counter never runs down to zero, and the simulation ends before 1^n is either accepted or rejected. In other words, 1^n is accepted by M_i if and only if 1^{n+1} would be too.
- Consider the deterministic simulation at step 8, if *this* is reached and executed. Since
$$|\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)|^2 < c$$
  the number of sequences of guessed moves with length at most $1.5 \cdot g(\ell+1)$ is at most linear in
$$\left(\sqrt{c}\right)^{1.5 \times g(\ell+1)}$$

The total number of steps used in the simulation is now bounded by the product of some fixed value depending on M — but not on the length of a padded encoding — and

$$\left(\sqrt{c}\right)^{1.5\times g(\ell+1)} \times g(\ell+1) \in o(c^{g(\ell+1)}).$$

Since $g(n+1) \in o(f(n))$, $g(\ell+1) < f(\ell)$ for sufficiently large ℓ and, indeed, the total number of steps used by this simulation is less than

$$c^{f(\ell)} = c^{f(h(i))} = h(i+1) = n \leq f(n).$$

Thus the binary counter is never run down to zero in this case either, so *this* simulation is also run to completion.

- Consider what this implies about the language L_D when n is so large that all the above conditions are satisfied: On input n, either step 6 or 8 is reached and the input string is either accepted or rejected before the binary counter runs down to zero.
  - We may assume that this is true for ***every*** integer n such that $h(i) < n \leq h(i+1)$ — noting that a padded version of the same nondeterministic Turing machine M is considered by the algorithm when $\mathtt{1}^n$ is received as input for *any* integer n in this range.
  - *Case:* $\mathtt{1}^{h(i)+1} \notin L_D$. When the algorithm is run on input $\mathtt{1}^{h(i)+1}$ it reaches step 6 and simulates the execution of M on input $\mathtt{1}^{h(i)+2}$. Since the timer does not run down one can see by an inspection of this step that $\mathtt{1}^{h(i)+2} \notin L_D$ — because the algorithm would ***accept*** $\mathtt{1}^{h(i)+1}$, otherwise.

    Indeed, considering the behaviour of the algorithm on inputs $\mathtt{1}^k$ for $k = h(i) + 2, h(i)+3, \ldots, h(i+1)-1$ — and seeing that step 6 is reached in every case — one can see that

    $$L_D \cap \{\mathtt{1}^{h(i)+1}, \mathtt{1}^{h(i)+2}, \ldots, \mathtt{1}^{h(i+1)-1}, \mathtt{1}^{h(i+1)}\} = \emptyset.$$

  - *Case:* $\mathtt{1}^{h(i)+1} \in L_D$. When the algorithm is run on input $\mathtt{1}^{h(i)+1}$ it reaches step 6 and simulates the execution of M on input $\mathtt{1}^{h(i)+2}$. Since the timer does not run down one can see by an inspection of this step that $\mathtt{1}^{h(i)+2} \in L_D$ — because the algorithm would ***reject*** $\mathtt{1}^{h(i)+1}$, otherwise.

    Indeed, considering the behaviour of the algorithm on inputs $\mathtt{1}^k$ for $k = h(i) + 2, h(i)+3, \ldots, h(i+1)-1$ — and seeing that step 6 is reached in every case — one can see that

    $$\{\mathtt{1}^{h(i)+1}, \mathtt{1}^{h(i)+2}, \ldots, \mathtt{1}^{h(i+1)-1}, \mathtt{1}^{h(i+1)}\} \subseteq L_D.$$

  - Thus

    $$\mathtt{1}^{h(i)+1} \in L_D \iff \mathtt{1}^{h(i+1)} \in L_D.$$

- However, if the algorithm is executed on input $1^{h(i+1)}$ then step 8 is reached and executed. Since the binary counter never runs down one can see by an inspection of *this* step that
$$1^{h(i)+1} \in L_D \iff 1^{h(i+1)} \notin D.$$
- Since both of these cannot be true at the same time a ***contradiction*** has been obtained, as needed to establish the claim. ☐

Proof of the Nondeterministic Time Hierarchy Theorem. Since g is a nondecreasing function and $g(n+1) \in o(f(n))$, $g(n) \in o(f(n))$ as well, and it certainly follows that NTIME$(g(n)) \subseteq$ NTIME$(f(n))$. The result is now a straightforward consequence of Lemmas 3 and 4. ☐