Lecture #8: Introduction to Nondeterministic Computation Proofs of Claims

This document includes proofs of the nontrivial claims, concerning nondeterministic computation, that were given in the notes for Lecture #8. This is ***for interest only***, since all the proofs, given here, are also sketched in the lecture notes.

A Useful Lemma

The following ***lemma*** will be used repeatedly when proving this, and later results — because many of the definitions used in this course imply that the number of steps used for some computation is "in $O(f(n))$", rather than "at most $f(n)$", when the input has length n.

Lemma. *Let $f, g : \mathbb{N} \to \mathbb{N}$ such that $g \in O(f)$. Then there exist a positive constant c_1 and a nonnegative constant c_0 such that*

$$g(n) \leq c_1 f(n) + c_0$$

for all $n \in \mathbb{N}$.

Proof. Let $f, g : \mathbb{N} \to \mathbb{N}$ such that $g \in O(f)$. Then it follows, by the definition of "$O(f)$", that there exists a constant N_0 and a *positive* constant c such that

$$g(n) \leq c f(n)$$

for all $n \in \mathbb{N}$ such that $n \geq N_0$.

Now let $c_1 = c$ and let

$$c_0 = \max\left(\max_{\substack{m \in \mathbb{N} \\ 0 \leq m \leq N_0 - 1}} \left(g(m) - cf(m)\right), 0 \right).$$

Then c_1 is a positive constant, since it is given that c is, and c_0 is a nonnegative constant by the above definition. Now, if $0 \leq N_0 - 1$ then

$$\begin{aligned} g(n) &= cf(n) + (g(n) - cf(n)) \\ &= c_1 f(n) + (g(n) - cf(n)) && \text{(since } c_1 = c\text{)} \\ &\leq c_1 f(n) + c_0 && \text{(by the definition of } c_0\text{, since } n \in \mathbb{N} \text{ and } 0 \leq n \leq N_0 - 1\text{).} \end{aligned}$$

On the other hand, if $n \in \mathbb{N}$ and $n \geq N_0$ then

$$\begin{aligned} g(n) &\leq cf(n) && \text{(since } n \geq N_0\text{, as noted above)} \\ &= c_1 f(n) && \text{(since } c_1 = c\text{)} \\ &\leq c_1 f(n) + c_0 && \text{(since } c_0 \geq 0\text{).} \end{aligned}$$

Thus $g(n) \leq c_1 f(n) + c_0$ for all $n \in \mathbb{N}$, as claimed. □

Proof of Claim #1

As the following result indicates, the two notions of "nondeterministic time" complexity classes are equivalent when the functions used to bound running time are well behaved:

Claim 1. *Let $f : \mathbb{N} \to \mathbb{N}$ be a time-constructible function such that $f(n) \in \Omega(n)$. Then*

$$\text{NTIME}(f) = \text{NTIME}_V(f).$$

Proof. Let $f : \mathbb{N} \to \mathbb{N}$ be a time-constructible function such that $f(n) \in \Omega(n)$. Consider a language $L \subseteq \Sigma^\star$ for some alphabet Σ.

- Suppose, first, $L \in \text{NTIME}(f)$.
  - Then there exists a *nondeterministic* Turing machine

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

    that decides L using time in $O(f(n))$. It follows, by the above lemma, that there exist a positive constant c_1 and a nonnegative constant c_0 such that, if M is executed on a string $\omega \in \Sigma^\star$ then the depth of the computation tree, modelling this computation, is at most $c_1 f(|\omega|) + c_0$.
  - Let k be the number of tapes that M has, so that k is a positive integer.
  - Since M is a *fixed* Turing machine its transition function is a *fixed* function, and there exists a *positive constant* $U \in \mathbb{N}$ such that, for every state $q \in Q$ and for all symbols $\sigma_1, \sigma_2, \ldots, \sigma_k \in \Gamma$, $\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)$ is a *subset* of $Q \times (\Gamma \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\})^k$ with size at most U.

– Ordering the states in Q and the symbols in the tape alphabet Γ (as is needed when describing an *encoding* of M as a string) one can also *order* the elements of $Q \times (\Gamma \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\})^k$ included in the value of each transition function. That is, for all $q \in Q$ and $\sigma_1, \sigma_2, \dots, \sigma_k \in \Gamma$, one can write

  $$\delta(q, \sigma_1, \sigma_2, \dots, \sigma_k) = \{\chi_0, \chi_1, \chi_2, \dots, \chi_{m-1}\}$$

  where $m = |\delta(q, \sigma_1, \sigma_2, \dots, \sigma_k)|$, and where $\chi_i \in Q \times (\Gamma \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\})^k$ for $0 \le i \le m-1$. Indeed, this can be done in such a way that χ_i can be determined from q, $\sigma_1, \sigma_2, \dots, \sigma_k$, and i.[1]

– Now let

  $$\Sigma_C = \{\tau_0, \tau_1, \dots, \tau_{U-1}\}$$

  be some alphabet with size U, for U as given above, whose elements are also ordered in some way.

– For each string $\omega \in \Sigma^\star$ with length $n \in \mathbb{N}$, every string $\nu \in \Sigma_C^\star$, such that $|\nu| \le c_1 f(n) + c_0$, can now be thought of as specifying some *path*, down from the root, in the *computation tree* for the execution of $\mathcal{M}$ on input ω.

– With that noted, consider a deterministic Turing machine, with input alphabet $\widehat{\Sigma} = \Sigma \cup \Sigma_C \cup \{\#\}$, that implements the algorithm shown in Figure 1 on page 4. Recall that, since the function $f : \mathbb{N} \to \mathbb{N}$, is time-constructible, there exists a deterministic Turing machine M_f (that computes functions) that takes the unary representation 1^n of a natural number $n \in \mathbb{N}$ and computes the binary representation of $f(n)$ as output, using $O(f(n))$ steps in the worst case. This is the Turing machine (with ℓ tapes) that is applied in step 5 of this algorithm.

– A string $\mu \in \widehat{\Sigma}^\star$ is only accepted by this algorithm if it reaches and passes the test, leading to acceptance, in the step at line 8. Since steps 1 and 2 must have been carried out before this, with the test at line 2 leading to rejection having *failed*, an examination of the details of steps 2–8 should suffice to confirm that μ is only accepted if $\mu = \omega\#\nu$ for some string $\omega \in \Sigma^\star$ and for some string $\nu \in \Sigma_C^\star$. Furthermore, since

  * the string on the third tape is the binary representation of $c_1 f(n) + c_0$ at the end of the fifth step, where $n = |\omega|$,
  * this counter's value is decremented every time step 6 is executed without rejecting the input,
  * a symbol on the first tape is examined during step 8, before the step at line 6 is reached again

[1]Since M is a *fixed* nondeterministic Turing machine, the *finite control* of the Turing machine $\widehat{M}$ that is being described in this proof, can be used to take care of this. This is more of a concern when one is describing an implementing a "universal nondeterministic Turing machine", instead.

On input $\mu \in \Sigma^\star \ldots$

1. While the symbol visible on the first tape is in Σ, copy this symbol onto the second tape — except that the first of these symbols should be "marked" (with a symbol $\dot{\sigma}$ used instead of a symbol $\sigma \in \Gamma$), so the leftmost cell on the second tape can be located, later on — and write 1 onto the third tape. Move ***right*** on the first, second and third tapes without moving on other tapes or changing the contents of any tape except the second and third.
2. If the symbol visible on the first tape is not # then ***reject***. Otherwise move ***right*** on the first tape without changing the contents of any tape or moving on any other tape.
3. While the symbol visible on the second tape is not a "marked" symbol (so that the tape head is not, yet, at the leftmost cell on this tape, move ***left*** on both the second and third tapes, without moving on other tapes or changing the contents of any tape.
4. Replace the "marked" symbol $\dot{\sigma}$ on the second tape with the corresponding symbol $\sigma \in \Gamma$ without moving on other tapes or changing the contents of any other tape.
5. Carry out the execution of the deterministic Turing machine M_f, using tapes $3, 4, \ldots, \ell + 2$ as M_f's tapes. If M_f's output tape stores the binary representation of an integer b, after this, replace this with the binary representation of $c_1 b + c_0$.

   Note: If this step was reached then μ begins with a substring ω# for some string $\omega \in \Sigma^\star$, and the second tape now stores ω, with infinitely many blanks, to the right, and with the tape head resting on the leftmost cell of the tape. If $|\omega| = n$ then the output tape of $\mathcal{M}_f$ now stores the binary representation of $c_1 f(n) + c_0$, with the tape head resting on the leftmost cell of the tape as well. Suppose that the output tape is tape #h (so that $3 \leq h \leq \ell + 2$).
6. Begin a simulation of M on input ω — using tape 2 and tapes #$\ell+3, \ell+4, \ldots, k+\ell+1$ as M's tapes (so that tape 2 is M's input tape) by setting the current state of M to be the start state, q_0.
7. If tape #h now stores a binary representation of 0 then ***reject***. Otherwise subtract one from the value represented by this counter.
8. Suppose that M is currently in state q and that symbols $\sigma_1, \sigma_2, \ldots \sigma_k$ are visible on tapes 2 and $\ell + 3, \ell + 4, \ldots, k + \ell + 1$ (so that these would be symbols visible on M's tapes). Let $m = |\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)|$.

   Reject if the symbol visible on the first tape is not $\tau_i \in \Sigma_C$ for some integer i such that $0 \leq i \leq m - 1$. Otherwise move ***right*** on the first tape — and update $\mathcal{M}$'s current state and the contents and locations of the tape heads of tapes 2 and $\ell+3, \ell+4, \ldots, k+\ell+1$ by carrying out the transition χ_i corresponding to q, $\sigma_1, \sigma_2, \ldots \sigma_k$, and i — ***accepting*** if χ_i moves to state q_{accept} and a blank is visible on the first tape, ***rejecting*** if χ_i moves to state q_{reject}, and returning to step 7, otherwise.

Figure 1: Verification Algorithm for the Language L

    * and, finally, since the input is rejected at step 6 if the counter's value reaches zero,

  it can be argued that $\mu = \omega\#\nu$ is only accepted if ω and ν are as described above and, furthermore, $|\nu| \leq c_1 f(|\omega|) + c_0$.

  Indeed, an examination of step 8 confirms that, for strings μ, ω and ν as above, the string $\mu = \omega\#\nu$ is accepted *if and only if* ν specifies a path, down from the root in the computation tree for $\mathcal{M}$ and ω down to leaf, that leads to an accepting configuration. Since this computation tree has depth *at most* $c_1 f(|\omega|) + c_0$, such a path exists *if and only if* $\omega \in L$.

– It remains only to confirm that a Turing machine implementing the algorithm in Figure 1 halts when executed on any input string $\mu \in \Sigma^\star$ and, furthermore, that if the input string μ has the form $\omega\#\nu$, where $\omega \in \Sigma^\star$ and $\nu \in \Sigma_C^\star$, then the number of steps used by this Turing machine is in $O(f(|\omega|))$.

    * All strings $\mu \in \widehat{\Sigma}^\star$ that do not begin with a sequence of symbols in Σ, followed by #, are ***rejected*** during the first two steps of the algorithm shown in Figure 1 after $O(|\mu|)$ moves have been made. It therefore suffices to consider strings $\mu = \omega\#\nu$ where $\omega \in \Sigma^\star$ and $\nu \in \widehat{\Sigma}^\star$, when completing the proof.
    * The first four steps of the algorithm shown in Figure 1 can be carried out at the cost of sweeping right over the prefix $\omega \in \Sigma^\star$ of the input string, sweeping back to the left over this, and carrying out at most a constant number of additional steps: $O(|\omega|)$ steps are used.
    * It is given that the function f is time-constructible, and (inferred) that M_f can be used to compute the binary representation of $f(n)$, where $n = |\omega|$, using $O(f(n))$ moves. Since M_f is a *fixed* Turing machine this can be considered to be a "component" of the Turing machine implementing the algorithm, so that this Turing machine can carry out each each move of M_f using a single move too. Thus the binary representation of $f(n)$ — a string with length in $O(\log f(n))$ — can be produced using $O(f(n))$ moves.

      Since c_1 and c_0 are positive integer *constants* the binary representation of $c_1 f(n) + c_0$ (another string with length in $O(f(n))$) can be obtained from the binary representation of $f(n)$ using $O(\log f(n))$ additional moves.

      It follows that step 5 of the algorithm in Figure 1 can be carried out using $O(f(n))$ moves.
    * It is not clear that step 6 requires a move, of its own, at all: It can probably be carried out just by choosing which state to move to, after the execution of step 5 ends.
    * The total number of moves, needed for all executions of step 7, is essentially the same as the total number of moves needed to decrement a binary counter, whose initial value is $c_1(f(n)) + c_0$, until the counter's value reaches zero. Since

c_1 and c_0 are positive integer *constants*, the total number of moves needed is in $O(f(n))$.

  - ⋆ Since M is a *fixed* nondeterministic Turing machine, its finite control can be "hard wired" into the finite control of the Turing machine implementing the algorithm, in such a way that each execution of the step at line 8 can be carried out using a single move, too. Since this step is executed at most $c_1 f(n) + c_0$ times, it follows that the total number of moves used for executions of this step is in $O(f(n))$ too.

  Since $f(n) \in \omega(n)$, $n \in O(f(n))$ and it follows by the above that the total number of moves used, for an execution of this algorithm on an input string $\mu = \omega\#\nu$, when $\omega \in \Sigma^\star$ and $\nu \in \Sigma_C^\star$, is in $O(f(|\omega|))$. It follows that $L \in \text{NTIME}_\text{V}(f)$.

  Since L was arbitrarily chosen from NTIME(f), it now follows that NTIME$(f) \subseteq$ NTIME$_\text{V}(f)$.

- Suppose, next, that $L \in \text{NTIME}_\text{V}(f)$.

  – Then there exists a verifier for f using time at most f. That is, there exists a "certificate alphabet" Σ_C and a *deterministic* Turing machine

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

  where $\widehat{\Sigma} = \Sigma \cup \Sigma_C \cup \{\#\}$, $\# \notin \Sigma \cup \Sigma_C$, such that the following properties are satisfied.

    - ⋆ M decides a language that is a subset of $\{\mu_I\#\mu_C \mid \mu_I \in \Sigma^\star \text{ and } \mu_C \in \Sigma_C^\star\}$, so that M only accepts a string $\mu \in \widehat{\Sigma}^\star$ if μ includes exactly one #.
    - ⋆ For every string $\omega \in \Sigma^\star$, $\omega \in L$ if and only if there exists at least one string $\nu \in \Sigma_C^\star$ (called a *certificate* for ω) such that M accepts the string $\omega\#\nu \in \widehat{\Sigma}^\star$.
    - ⋆ There exist a positive constant c_1 and a nonnegative constant c_0 such that, for every string $\mu \in \widehat{\Sigma}^\star$, if $\mu = \omega\#\nu$ for a string $\omega \in \Sigma^\star$ and a string $\nu \in \Sigma_C^\star$, then the number of steps that M takes when executed on μ is at most $c_1 f(|\omega|) + c_0$.

  – For every positive integer, i, the i^{th} symbol in the certificate is $|\omega| + i + 1$ cells to the right of the leftmost cell on the first tape — so M must make at least $|\omega| + i + 1$ moves before this symbol is seen.

  It follows that — since M makes at most $c_1 f(|\omega|) + c_0$ moves — if

  $$i > c_1 f(|\omega|) - |\omega| + c_0$$

  — then the i^{th} symbol in the certificate is not read, at all, before the execution of M on ω ends.

Thus, if $\nu \in \Sigma_C^\star$ is a certificate for a string $\omega \in L$, whose length is greater than $c_1 f(|\omega|) - |\omega| + c_0$, then the *prefix* $\widehat{\nu} \in \Sigma_C^\star$ with length $c_1 f(|\omega|) - |\omega| + c_0$ is also a certificate for ω.

It follows that, for every string $\omega \in \Sigma^\star$, $\omega \in L$ if and only if ω has a certificate *whose length is at most* $c_1 f(|\omega|) - |\omega| + c_0$.

– With that noted, consider a nondeterministic Turing machine $\widehat{M}$, with input alphabet Σ, and a tape alphabet Γ such that $\Sigma \cup \Sigma_C \cup \{\#\} \subseteq \Gamma$, implementing the nondeterministic algorithm in Figure 2 on page 8. One can see by an examination of the details of this algorithm that a string $\omega \in \Sigma^\star$ is accepted by M if and only if there is a certificate $\nu \in \Sigma_C^\star$ for ω with length at most $c_1 f(|\omega|) + c_0$. It therefore follows, by what has been noted above about the existence of “short” certificates, that the language of $\widehat{M}$ is L.

  It therefore remains only to show that the computation tree for $\widehat{M}$ and ω is finite, for all $\omega \in \Sigma^\star$ (so that $\widehat{M}$ decides L) and, furthermore, that this computation tree has depth in $O(f(|\omega|))$, to complete the proof.

  It now suffices to note (as in the analysis of the verification algorithm in Figure 1, above), that the initial steps of the algorithm in Figure 2 simply use a constant number of sweeps over the input string, so that they can be carried out using $O(|\omega|)$ moves on input ω. The third step involves the application of a Turing machine, M_f (assumed, here, to have ℓ tapes) to compute the binary representation of $f(|\omega|)$ and perform additional arithmetic, using $O(f|\omega|)$ moves. The total cost of all executions of step 5 is dominated by the number of moves to repeatedly decrement the value of a binary counter, with initial value $c_1 f(|\omega|) + c_0$, until the value reaches zero — which is in $O(f|\omega|)$ as well. Steps 6 is executed at most $c_1 f(|\omega|) + c_0$ times and each execution requires at most a constant number of moves, so that the total number of moves for all of its executions is also in $O(f(|\omega|)$. Step 7 requires a sweep over the string on the second tape, which has length in $O(f|\omega|)$. Finally, the number of moves of M included in the simulation at line 8 is at most $c_1 f(|\omega|) + c_0$ and, since M is a fixed Turing machine, this can be included as a component of $\widehat{M}$ so that the simulation of each move of $\mathcal{M}$ requires only a single move of $\widehat{M}$. It is not hard to show that the cost of other steps is in $O(f(|\omega|)$ too, so that the total number of moves used by $\widehat{M}$ is in $O(f(|\omega|))$, as required.

Since L was arbitrarily chosen from NTIME$_{\mathsf{V}}(f)$, it follows that NTIME$_{\mathsf{V}}(f) \subseteq$ NTIME(f).
$\square$

On input $\omega \in \Sigma^\star$ . . .

1. While the symbol visible on the first tape is not blank, copy this symbol onto the second tape — except that the first of these symbols should be "marked" (with a symbol $\dot{\sigma}$ used instead of a symbol $\sigma \in \Gamma$) so that the leftmost cell on the second tape can be located, later on — and write 1 onto the third tape. Move ***right*** on the first, second, and third tapes without moving on other tapes or changing the contents of any tape except the second and third.
2. While the symbol visible on the second tape is not a "marked" copy of a symbol in Γ, move ***left*** on both the second and third tapes without moving on other tape or changing the contents of any tape.
3. Carry out the execution of the deterministic Turing machine M_f, using tapes $3, 4, \dots, \ell+2$ as $\mathcal{M}_f$'s tapes. If M_f's output tape stores the binary representation of an integer b, after this, replace this with the binary representation of $c_1 b + c_0$.
4. While the symbol visible on the second tape is not blank, move ***right*** on this tape without moving on other tapes of changing the contents of any tape. When a blank is seen on the second tape write # on this tape, moving right — again, without moving on, or changing the contents of, other tapes.
5. If the value of the binary counter on the M_f's output tape is zero, then go to step 7. Otherwise decrease the value of this binary counter by one.
6. Nondeterministically ***guess*** either to go to step 7 or to ***guess*** a symbol $\tau \in \Sigma_C$, writing τ onto the second tape and moving ***right*** (again, without moving on or changing the contents of other tapes) and returning to step 5.
7. While the symbol visible on the second tape is not a "marked" copy of a symbol in Γ move ***left*** on this tape without moving on or changing the contents of other tapes. When a "marked" symbol $\dot{\sigma}$ is visible on the second tape, replace it with the corresponding "unmarked" symbol $\sigma \in \Gamma$ on the second tape, without moving on any tape or changing the contents of any other tape.
8. Simulate the execution of M using the second tape as M's first (input) tape and using tapes $\ell+3, \ell+4, \dots, k+\ell+1$ as M's other tapes — ***accepting*** if M accepts, and ***rejecting*** if M rejects.

Figure 2: A Nondeterministic Algorithm to Decide L

Proof of Claim #2

Claim 2. $\mathsf{TIME}(f) \subseteq \mathsf{NTIME}(f)$ *for every function* $f : \mathbb{N} \to \mathbb{N}$.

Proof. As described in the lecture notes, it is sufficient to "interpret" a deterministic Turing machine as a nondeterministic one that never guesses.

Suppose, in particular, that $f : \mathbb{N} \to \mathbb{N}$ and let $L \subseteq \Sigma^\star$ such that $L \in \mathsf{TIME}(f)$. Then there exists a deterministic Turing machine

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

such that M decides L. Furthermore, there exist a positive integer constant c_1 and a nonnegative integer constant c_0 such that M uses at most $c_1 f(|\omega|) + c_0$ moves when executed on input ω, for every string $\omega \in \Sigma^\star$. Suppose that M has k tapes.

Now let

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

be a ***nondeterministic*** Turing machine with the same number, k of tapes as M and with the same set Q of states, input alphabet Σ, tape alphabet Γ, start state q_0, accepting state q_{accept}, and rejecting state q_{reject}. Suppose as well that, for every state $q \in Q$ and all tape symbols $\sigma_1, \sigma_2, \dots, \sigma_k \in \Gamma$,

$$\widehat{\delta}(q, \sigma_1, \sigma_2, \dots, \sigma_k) = \begin{cases} \emptyset & \text{if } q = q_{\text{accept}} \text{ or } q = q_{\text{reject}}, \\ \{\delta(q, \sigma_1, \sigma_2, \dots, \sigma_k)\} & \text{otherwise.} \end{cases}$$

Since M and $\widehat{M}$ have the same number of tapes, set of states, and tape alphabet, they also have the same ***configurations***. With that noted, it is easy to prove the following by induction on t: For every integer t such that $t \geq 0$, and every configuration $\mathcal{C}$ of M (and of $\widehat{M}$),

- If M halts after $t - 1$ or fewer moves, when starting in configuration $\mathcal{C}$, then there is no configuration that is reachable from $\mathcal{C}$, using t or more moves of $\widehat{M}$, and
- otherwise, if $\widehat{\mathcal{C}}$ is the configuration that is reachable using t moves of M, when starting from configuration $\mathcal{C}$, then the *set* of moves reachable using t moves of $\widehat{M}$, when started from configuration is $\{\widehat{\mathcal{C}}\}$.

This can be used to show that $\widehat{M}$ accepts ω if and only if M accepts ω, for every string $\omega \in \Sigma^\star$ — and that the depth of the computation tree for $\widehat{M}$ and ω is at most $c_1 f(|\omega|) + c_0$ for every string $\omega \in \Sigma^\star$. Thus $\widehat{M}$ *decides* L, since M does and, furthermore, $L \in \mathsf{NTIME}(f)$.

Since L was an arbitrarily chosen language from $\mathsf{TIME}(f)$ it follows that $\mathsf{TIME}(f) \subseteq \mathsf{NTIME}(f)$, as claimed. □

Proof of Clam #3

Claim 3. *For every function $f : \mathbb{N} \to \mathbb{N}$ and for every language $L \subseteq \Sigma^\star$ such that $f \in$ NTIME(f), there exists an integer constant c (depending on L) such that $L \in$ TIME(c^f). Thus*

$$\mathsf{NTIME}(f) \subseteq \bigcup_{c \in \mathbb{N}} \mathsf{TIME}(c^f).$$

Proof. As described in the lecture notes, it suffices to describe and analyze a deterministic Turing machine that uses ***depth-first search*** in the computation tree for a nondeterministic Turing machine (that decides a given language), and for a given input string.

With that noted, let $L \subseteq \Sigma^\star$ such that $L \in$ NTIME(f). Then there exists a nondeterministic Turing machine

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

that decides L and, furthermore, there exist a positive integer constant c_1 and a nonnegative integer constant c_0 such that the depth of the computation tree for M and ω is at most $c_1 f(|\omega|) + c_0$, for every string $\omega \in \Sigma^\star$. Suppose that M has k tapes.

Let us assume that $q_0 \notin \{q_{\text{accept}}, q_{\text{reject}}\}$ — for either $L = \emptyset$ or $L = \Sigma^\star$ and L is, trivially decidable deterministically in constant time (so that there is nothing to prove) otherwise.

Since M is a *fixed* nondeterministic Turing machine, there exists a positive integer constant U such that

$$|\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)| \leq U$$

for every state $q \in Q$ and all tape symbols $\sigma_1, \sigma_2, \ldots, \sigma_k \in \Gamma$. It is now easily proved by induction on a non-negative integer d that, if the computation tree for M and a string $\omega \in \Sigma^\star$ has depth d, then the *size* of this computation tree is at most $d+1$ if $U = 1$, and at most $U^{d+1} - 1$ if $U \geq 2$.

It is also possible to *order* the moves that are included in the value of the transition function, for any input. That is, for $q \in Q$ and $\sigma_1, \sigma_2, \ldots, \sigma_k \in \Gamma$, if

$$|\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)| = s$$

(so that s is an integer such that $0 \leq s \leq U$) then, if $s \geq 1$ then we may write $\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k)$ as

$$\delta(q, \sigma_1, \sigma_2, \ldots, \sigma_k) = \{\chi_0, \chi_1, \ldots, \chi_{s-1}\}$$

where $\chi_0, \chi_1, \ldots, \chi_{s-1} \in Q \times (\Gamma \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\})^k$ and, furthermore, χ_i can be determined from i, for each integer i such that $0 \leq i \leq s-1$.

With all this noted, consider a ***deterministic*** Turing machine

$$\widehat{M} = (\widehat{Q}, \Sigma, \widehat{\Gamma}, \widehat{\delta}, \widehat{q}_0, \widehat{q}_{\text{accept}}, \widehat{q}_{\text{reject}})$$

with $k+1$ tapes, that performs a depth-first search of the computation tree for M and the input string, $\omega \in \Sigma^\star$ — searching for an accepting configuration.

- $\widehat{M}$ has tape alphabet
$$Q \cup \Gamma \cup \{\dot{\sigma} \mid \sigma \in \Gamma\} \cup \{\texttt{(}, \texttt{)}, \texttt{,}, \texttt{L}, \texttt{R}, \texttt{S}\}.$$
- $\widehat{M}$'s $k+1^{\text{st}}$ tape will describe a ***path*** down from the root to some node in the computation tree. The representation of the path that appears on this tape will be given in more detail, below.
- The first k tapes of $\widehat{M}$ represent the configuration that $\mathcal{M}$ would be in, if it started with the initial configuration for the input string ω and then carried out the sequence of moves corresponding to the path shown on the $k+1^{\text{st}}$ tape.

  However, the contents of the ***leftmost*** cell on each of these tapes is "marked" in order to make the left end of the tape easy to detect: For $1 \le i \le k$, if the leftmost cell of the i^{th} tape of M would store a symbol $\sigma \in \Gamma$, then the leftmost cell of the i^{th} tape of $\widehat{M}$ stores the corresponding "dotted" symbol, $\dot{\sigma}$, instead.

The contents of the tapes of $\widehat{M}$ can be defined "inductively", in terms of the length, ℓ, of the path down the computation tree that is represented by $\widehat{M}$'s $k+1^{\text{st}}$ tape.

- If $\ell = 0$ then there are no non-blank symbols on $\widehat{M}$'s $k+1^{\text{st}}$ tape and the tape head for this tape is resting on the leftmost cell of the tape. $\widehat{M}$'s other tapes represent the initial configuration for M and the input string ω.

  In particular, let $n = |\omega|$.

  - If $n = 0$, so that $\omega = \lambda$, the empty string, then the leftmost cell of the first tape should store the "dotted" version of blank, $\dot{\sqcup}$, and every other cell of this tape should store a blank.
  - On the other hand, if $n \ge 1$ and
  $$\omega = \alpha_1\alpha_2 \dots \alpha_n$$
  then the leftmost cell of the first tape should store $\dot{\alpha}_1$, the i^{th} cell should store α_i for $2 \le i \le n$, and every other cell should store a blank.
  - The tape head for the first tape should rest on the leftmost cell of the tape, in either case.

  For $2 \le i \le k$, the leftmost cell of the i^{th} tape should store $\dot{\sqcup}$, every other cell should store a blank, and the tape head for this tape should rest on the leftmost cell.

  $\widehat{M}$'s finite control should be used to remember that M would be in its start state, q_0, at this point.

- Suppose, instead, that $\ell \geq 1$. Then the path $\mathcal{P}$ in the computation tree, represented using the $k+1$st tape, is obtained by starting with a path $\widehat{\mathcal{P}}$ with length $\ell-1$, and adding one more (downward) edge in the computation tree.
  - To begin, consider the contents and positions of tape heads that $\widehat{M}$'s tapes would have, if the (shorter) path $\widehat{\mathcal{P}}$ was represented, instead.

    For $1 \leq i \leq k$, the symbol that would be visible on the ith tape, at this point, is a symbol
    $$\overline{\sigma}_i \in \Gamma \cup \{\dot{\sigma} \mid \sigma \in \Gamma\}.$$
    Thus $\overline{\sigma} \in \{\sigma_i, \dot{\sigma}_i\}$ for some (corresponding) symbol $\sigma_i \in \Gamma$.

    This configuration includes some state $q \in Q$ — which is remembered using $\widehat{M}$'s finite control. Furthermore, $q \notin \{q_{\text{accept}}, q_{\text{reject}}\}$, since it would not be possible to extend $\widehat{\mathcal{P}}$ by adding another edge, otherwise.
  - If q and $\sigma_1, \sigma_2, \dots, \sigma_k$ are as above then $\delta(q, \sigma_1, \sigma_2, \dots, \sigma_k)$ must be a ***nonempty*** subset of $Q \times (\Gamma \times \{\mathtt{L}, \mathtt{R}, \mathtt{S}\})^k$ — because it would not be possible to extend the path $\widehat{\mathcal{P}}$, in the computation tree, otherwise.

    Indeed, the path is extended by choosing one of the moves
    $$(r, ((\tau_1, d_1), (\tau_2, d_2), \dots, (\tau_k, d_k)))$$
    in the set $\delta(q, \sigma_1, \sigma_2, \dots, \sigma_k)$ and carrying this move out.
  - Thus, for $1 \leq i \leq k$, the ith tape would be updated, either by replacing σ_i with τ_i (if $\overline{\sigma}_i = \sigma_i$) or by replacing $\dot{\sigma}_i$ with $\dot{\tau}_i$ (if $\overline{\sigma}_i = \dot{\sigma}_i$). Let $\overline{\tau}_i$ be whichever of τ_i or $\dot{\tau}_i$ is written onto $\widehat{M}$'s tape when this update is carried out. The ith tape head would then move in direction d_i (staying where it is if $d_i = \mathtt{L}$ and the symbol visible was $\dot{\sigma}_i$, so that the tape head was already at the leftmost cell of the tape).
  - The representation of $\mathcal{P}$, stored on the $k+1$st tape, would be the concatenation of the representation of $\widehat{\mathcal{P}}$ and the string
    $$\mathtt{(}q\mathtt{,}\overline{\sigma}_1\mathtt{,}\overline{\sigma}_2\mathtt{,}\dots\mathtt{,}\overline{\sigma}_k\mathtt{,}r\mathtt{,(}\overline{\tau}_1\mathtt{,}d_1\mathtt{),(}\overline{\tau}_2\mathtt{,}d_2\mathtt{),}\dots\mathtt{,(}\overline{\tau}_k\mathtt{,}d_k\mathtt{))}$$
    (with length $8k+5$, in $\widehat{\Gamma}^\star$) — so that the string representing a path with length ℓ has length $8k\ell + 5\ell$ for every integer $\ell \geq 0$. Note that the above string identifies the tape heads that were resting at the leftmost cell, both before and after this update was carried out.

    $\widehat{M}$'s finite control would be used to remember that M would now be in state r.

    Each step in the "breadth first search" algorithm can be carried out using $O(k)$ moves of $\widehat{M}$. Details needed to establish this (and the correctness of the process) are left as an exercise.

The total number of steps used by the breadth first search algorithm is at most linear in the size of the computation tree. As noted above, this size is at most $c_1 f(|\omega|) + 2$ if $U = 1$ or at most $U^{c_1 f(|\omega|)+c_0+1} - 1$ if $U \geq 2$. In either case, if we set

$$c = \lceil \max(U, 2)^{c_1} \rceil$$

then it follows that

$$L \in \mathsf{TIME}(c^f) \subseteq \bigcup_{c \in \mathbb{N}} \mathsf{TIME}(c^f).$$

Since L was arbitrarily chosen from NTIME(f) it follows that

$$\mathsf{NTIME}(f) \subseteq \bigcup_{c \in \mathbb{N}} \mathsf{TIME}(c^f),$$

as claimed. □