Indian Association for the Cultivation of Science

(Deemed to be University under *de novo* Category)

Master's/Integrated Master's-PhD Program/ Integrated Bachelor's-Master's Program/PhD Course

Theory of Computation II: COM 5108

Lecture I

Instructor: Goutam Biswas *Autumn Semester 2023*

1 Basic Complexity Classes

We consider *decidable* problems (Δ_1^0). It is known that we cannot compute every function or decide every language. But about the functions that are computable, or the languages that are decidable, one may ask whether the computation can be done within *reasonable time* with reasonable amount of space (memory). Computation time required to solve a problem increases with the increase in the size of the problem. For some problem the rate of increase is known to be *reasonable (polynomial)*, for some other problem the growth is known to be *steep (exponential)*. But there are large number of useful problems for which the computation time of known algorithms are *high*, but it is not known whether there can be a better algorithm. And there is a strong belief (no proof) that there cannot be any.

A Turing machine may be viewed as an algorithm. We try to characterize the time complexity of an algorithm in terms of the "*number of steps*" taken by a Turing Machine (TM) to solve a problem instance. The "*number of steps*" taken by a TM is treated as the time taken by it. For a given problem there may be different Turing machines (algorithms) with different running times. Even for a particular TM (algorithm) for a problem, there are *good instances* and *bad instances* of the input in terms of the number of steps. We are often interested about the asymptotic estimate of the worst case scenario or the average case scenario.

Problems are classified in terms of the nature of growth of the number of computation steps of an algorithm with the increase in the size of input. Turing machine is our basic model of computing. The *instruction set (basic operations)* of a Turing machine is primitive compared to the instruction set of a modern CPU. But any problem solvable on a modern computer in a certain time can in principle be solved on a multi-tape Turing machine with only a polynomial slowdown. Similarly a multi-tape Turing machine can also be simulated on a single-tape Turing machine with a quadratic slowdown. Also the simulation of Turing machines of similar types can have linear speedup.

1.1 Turing Machines

We already have talked about different Turing machine models. In the *start configuration*, the input tape has $\triangleright x \sqcup^\omega$, where $x \in \Sigma^*$ is the actual input to the machine. The Σ is often $\{0,1\}$. In a k-tape machine we may assume that the input tape is *read-only*. Other $(k-1)$ tapes/strings are empty, $\triangleright \sqcup^\omega$, where $\sqcup^\omega$ stands for the infinite number potentially available blank cells. Henceforth we shall not specify them explicitly. All tape-heads are below the corresponding left-end marker. Each head can move to *left* ($\leftarrow$) or to *right* ($\rightarrow$) or may remain *stationary* ($-$). But a head cannot move left of the left-end marker.
The computation is a sequence of *configurations* consisting of *state*, *head positions*, and contents of k tapes.

The machine does not move after entering a *halt state*. For some proof we assume that the machine cleans all its work-tapes, and positions the input tape head to some appropriate place before entering an accept halt state. A machine may never enter a *halt state* and run forever. Often our machines are decider or computing total functions. Such machines halts on every input.

We have also defined a *nondeterministic* Turing machine. The difference in such a machine is in their transition function. There may be more than one possible transitions from a combination of state and current input symbol, $(q, \sigma_1, \gamma_2, \cdots, \gamma_{k-1})$. A nondeterministic machine accepts an input $x \in \Sigma^*$ if it starts from a start configuration, and there is a *computation path* (more than one computation paths are possible due to nondeterminism) that reaches an *accept-halt* configuration.

Let the maximum degree of nondeterminism in the state transition function of a nondeterministic Turing machine M be m. Then there exists an equivalent nondeterministic Turing machine with the degree of nondeterminism at most two. Such a machine may be viewed of having two transition functions δ_0 and δ_1, that are nondeterministically chosen at every combination of state and input.

There are Turing machines that computes function $F : \mathbb{N}_0 \to \mathbb{N}_0$. What is actually computed is the numeral of $F(n)$, from the numeral of n as input. If we choose binary numeral, then we have the corresponding function $f : \{0,1\}^* \to \{0,1\}^*$, such that whenever x is a binary numeral of n, then $f(x)$ is the binary numeral of $F(n)$. The computation of F is equivalent to the computation of f. $[\mathbb{N}_0 = \{0, 1, 2, \cdots\}]$

The description of a Turing machine can be encoded as a string over $\{0,1\}$. As such every $x \in \{0,1\}^*$ is not a *valid* encoding of any TM. But we shall assume that a *trivial* TM is encoded by those invalid strings. So every $x \in \{0,1\}^*$ encodes a TM.

A TM M can be represented by infinitely many strings. This it seems simplifies some of the proofs.

There is a *universal TM* $\mathcal{U}$ such that for every $\alpha, x \in \{0,1\}^*$, $\mathcal{U}(\alpha, x) = M_\alpha(x)$, where α is an encoding of TM M_α. Also if M_α halts on x in t steps, then $\mathcal{U}$ halts on (α, x) in $Ct \log t$ steps, where c is a constant depends on M_α.

1.2 Time and Space Bounded Classes

We are interested about Turing machines that halts on all input. In such a machine the computation time or space can be expressed as a function of the size of input. A complexity class is a collection of problems that can be solved

within the time (or space) bounded by some function of input size.

Definition of complexity class depends on *machine model* e.g. k-tape Turing machine; mode of operation and acceptance e.g. deterministic or nondeterministic; type of resource e.g. time, space etc.

Definition 1. If a k-string deterministic Turing machine M halts on every input $x \in \Sigma^*$, and takes at most $f(n)$ steps for all but finitely many inputs of length n, where $T : \mathbb{N}_0 \to \mathbb{N}_0$, then M is called $f(n)$-*time bounded*.

A nondeterministic Turing machine is also time bounded in a similar way. But in this case computations on all possible paths must come to a halt state within $f(n)$ steps.

If a TM M is a *decider*, then $L(M)$ is decided in $T(|x|)$ time, where $x \in \Sigma^*$ is an input. As a computer of a function, M starts with x on the input tape, and when it halts, it has $f(x)$ on the output tape. We say that M computes $f : \{0,1\}^* \to \{0,1\}^*$ in $T(|x|)$ time.

A decision problem of a language $L \subseteq \Sigma^*$ may be viewed as computation of the characteristic function $\mu_L : \{0,1\}^* \to \{0,1\}$, so that $\mu_L(x) = 1$ if and only if $x \in L$.

Definition 2. If a k-tape deterministic Turing machine M halts on every input $x \in \Sigma^*$, and uses at most $f(n)$ cells on the $(k-1)$ work-tapes, for all but finitely many inputs of length n, then M is called $f(n)$-*space bounded*. The function S is from $\mathbb{N}_0$ to $\mathbb{N}_0$.

A nondeterministic Turing machine is also space bounded in a similar way. But in this case computations on each of its possible paths should not use more than $f(n)$ cells of work-tapes. If M is a decider, then $L(M)$ is decided in $f(n)$ space.

It is necessary that the space and time binding functions should be *nondecreasing* and *constructible*.

Definition 3. A function $f : \mathbb{N}_0 \to \mathbb{N}_0$ is said to be *time constructible* if f is *nondecreasing* and there is k-tape Turing machine $M_f = (Q, \Sigma, \delta, s)$, that for any $n \in \mathbb{N}_0$ and any input x of length n, does the following computation in time $t = O(n + f(n))$.

$$(s, \rhd, x, \rhd, \varepsilon, \cdots, \rhd, \varepsilon) \to^t_{M_f} (h, \rhd, x, \rhd, \sqcup^{j_2}, \cdots, \rhd, \sqcup^{j_{k-1}}, \rhd, 1^{f(|x|)}),$$

where $j_i = O(f(n)), i = 2, 3, \cdots, k-1$.

Note: There are variations of this definition e.g. (i) $f(n)$ is at least $O(n \log n)$ or (ii) the Turing machine maps 1^n to the binary representation of $f(n)$ in time $O(f(n))$ etc.

Definition 4. A function $f : \mathbb{N}_0 \to \mathbb{N}_0$, where $f(n) \geq O(\log n)$, is called *space constructible*, if there is a Turing machine that takes x of length $n \in \mathbb{N}_0$ as input and computes $f(n)$ using $O(f(n))$ work-tape cells.

The class of *constructible* or *proper* functions includes all reasonable functions such as $n, n \log n, n^k, 2^n$ etc. It can be proved that if functions f and g are constructible, then $g \circ f, f + g, f \times g$, 2^f and many more are constructible.

Let $T : \mathbb{N}_0 \to \mathbb{N}_0$ and $S : \mathbb{N}_0 \to \mathbb{N}_0$ be proper functions. We define the following complexity classes. Our model is a k tape Turing machine.

1. $DTIME(f(n)) = \{L :$ there is a $f(n)$ time bounded deterministic Turing machine (DTM) M so that $L = L(M)\}$.

2. $NTIME(f(n)) = \{L :$ there is a $f(n)$ time bounded nondeterministic Turing machine (NTM) M so that $L = L(M)\}$.

3. $DSPACE(f(n)) = \{L : \text{there is a } f(n) \text{ space bounded DTM } M \text{ so that } L = L(M)\}$.
4. $NSPACE(f(n)) = \{L : \text{there is a } f(n) \text{ space bounded NTM } M \text{ so that } L = L(M)\}$.
5. $LOGSPACE = DSPACE(\log n)$.
6. $NLOGSPACE = NSPACE(\log n)$.
7. $P = \cup_{i\geq 1} DTIME(n^i)$.
8. $NP = \cup_{i\geq 1} NTIME(n^i)$.
9. $PSPACE = \cup_{i\geq 1} DSPACE(n^i)$.
10. $NPSPACE = \cup_{i\geq 1} NSPACE(n^i)$.
11. $EXP(EXPTIME) = \cup_{i\geq 1} DTIME(2^{n^i})$.
12. $NEXP(NEXPTIME) = \cup_{i\geq 1} NTIME(2^{n^i})$.
13. $EXPSPACE = \cup_{i\geq 1} DSPACE(2^{n^i})$.
14. $NEXPSPACE = \cup_{i\geq 1} NSPACE(2^{n^i})$.

A variation of the definition is as follows:
$DTIME(f(n)) = \{L : \text{there is a } O(f(n)) \text{ time bounded deterministic Turing machine } M \text{ so that } L = L(M)\}$.
The linear speedup theorem is absorbed in the definition. It is often used it to simplify the argument.

Example 1. If a language L is decided in time $f(n) = 3.5n^2 + 10n\log n$, then according to our original definition $L \in DTIME(3.5n^2 + 10n\log n)$. But then we can have a constant c so that $cn^2 > 3.5n^2 + 10n\log n$ for all but finitely many inputs. So we can write $L \in DTIME(cn^2)$. But by the linear speedup theorem if there is a k-tape Turing machine ($k > 1$) that decides L in cn^2 time then there is another k-tape Turing machine, that decides L in n^2 time. So $L \in DTIME(n^2)$.

Proposition 1. Let L be decided by a k-tape Turing machine M in time $f(n)$. Then L can be decided by a single-tape Turing machine N in time $O(f(n)^2)$.

Proof: Let us take $k = 3$. The 3-tape Turing machine $M = (Q_m, \Sigma, \delta_m, s_m)$ and the single tape Turing machine $N = (Q_n, \Sigma_n, \delta_n, s_n)$. The initial configuration of M on input x is

$$(s_m, \rhd, x, \rhd, \varepsilon, \rhd, \varepsilon).$$

All heads are under '$\rhd$'. Let $\Sigma_m = \{\rhd, \sqcup, \cdots\}$. We take $\Sigma_n = \{\sigma, \underline{\sigma} : \ \sigma \in Q_m\} \cup \{\rhd', \lhd\}$. The new symbols $\underline{\sigma}$ indicate the head positions on different tapes of M on the tape of N. The symbol $\rhd'$ corresponds to $\rhd$ of M but can be crossed to left on N. The right end of a tape/string is indicated by $\lhd$.

Let $C_m = (p_m, x_1, u_1, x_2, u_2, x_3, u_3)$ be a configuration of M, where

$$\begin{aligned} x_1 &= \triangleright\sigma_{11}\cdots\sigma_{1a}, \\ u_1 &= \gamma_{11}\cdots,\gamma_{1b}, \\ x_2 &= \triangleright\sigma_{21}\cdots\sigma_{2c}, \\ u_2 &= \gamma_{21}\cdots,\gamma_{2d}, \\ x_3 &= \triangleright\sigma_{31}\cdots\sigma_{3e}, \\ u_3 &= \gamma_{31}\cdots,\gamma_{3f}. \end{aligned}$$

The encoding of the configuration C_m on N is
$C_n = (p_n, \triangleright, x'_1u_1 \lhd x'_2u_2 \lhd x'_3u_3 \lhd \lhd)$, where

$$\begin{aligned} x'_1 &= \triangleright'\sigma_{11}\cdots\underline{\sigma_{1a}}, \\ x'_2 &= \triangleright'\sigma_{21}\cdots\underline{\sigma_{2c}}, \\ x'_3 &= \triangleright'\sigma_{31}\cdots\underline{\sigma_{3e}}. \end{aligned}$$

The initial configuration of N is $(s_n, \triangleright, x, \varepsilon)$, where $x = \sigma_1\cdots\sigma_k$. But N changes it to the encoded initial configuration of M as follows:

$$(s_m, \triangleright, \underline{\triangleright}x \lhd \underline{\triangleright} \lhd \underline{\triangleright} \lhd \lhd).$$

To simulate a move of M, the machine N scans the input and identifies the symbols under the three heads of M. This is a finite amount of information that depends on the machine M. In the second pass it makes three updates by shifting head positions of M and modifying the symbols under the head and comes back to the leftmost cell. This pass requires the state transition table of M within N (again a finite amount of information). Each of these two passes take $O(f(n))$ time.
If M at some point writes in the first 'blank cell' on the right of the i^{th} tape, then in N the contents of $(i+1)^{th}$ tape onwards are to be shifted by one cell. This will again take $O(f(n))$ time.
So each move of M is simulated in $O(f(n))$ time, and to simulate $f(n)$ moves on M requires $O(f(n)^2)$ time. QED.

If a language L is decided in $f(n)$ number of steps in any Turing machine M, then we can reduce the number of steps by any constant factor $\epsilon > 0$. We can design a Turing machine that decides L in $\epsilon f(n)$ time when $0 < \epsilon < 1$ and $f(n) > n$ ($\epsilon > 1$ is not of much interest). Following is a precise version of the theorem of *linear speedup*.

Proposition 2. If $L \in DTIME(f(n))$, then $L \in DTIME(\varepsilon f(n) + O(n))$, where $\varepsilon > 0$.

Proof: Let M be a k-tape DTM deciding L in $f(n)$ steps. If $k > 1$, then there is a k-tape DTM N that will decide L in $\varepsilon f(n) + O(n)$ steps. If $k = 1$, then N must be be 2-tape machine. All tapes are read/write enabled.

The main idea is, each tape symbol of N will encode more than one tape symbols of M, so that many moves of M can be simulated by a single move of N. The value of m depends on ε and M. Let $\Sigma_N = \{\triangleright, \sqcup\} \cup \Sigma_M \cup \Sigma_M^m$.
The machine N first encodes the input x by taking m consecutive elements at a time and writes it on the second tape. If x is not a multiple of m it can be padded with $\sqcup$s. This can be done in $O(|x|)$ steps and accounts for $O(n)$ of the theorem.

This point onward the second tape contains the input for N. Its length is $\left\lceil \frac{|x|}{m} \right\rceil$. The first tape may be used as a work tape by putting the symbol '$\triangleright$' at the end of the original input of M.

In the main part of simulation of M, the machine N simulates m steps of M in k or fewer steps, where k is a constant (typical value is 6). Let q be the state of M and $j_1, \cdots, j_k$ be the positions of heads on k-tapes of M. Each $j_i \leq m$ will be within the current *m-tuple* on each tape. This information, $(q, j_1, \cdots, j_k)$, will be stored as a part of the state of N.

The machine N makes one left, then two right and then another left move (total 4) on all its k-tapes to gather information about the current, previous and next *m-tuples* of M. The state transition table of M is available to N. So it can predict the next m-moves of M. Note that next m moves of M cannot take any one of k heads of M beyond the current, left or right m-tuples of the k heads of N. The required information in the state of N is $Q \times \{1, 2, \cdots, m\} \times \Sigma^{3km}$. The first two component (q, i), $q \in Q$ and $i \in \{1, 2, \cdots, m\}$ gives state and the next m move numbers of M. The last component gives the content of three consecutive m-tuples on k tapes $(((\Sigma^m)^3)^k)$.

It may be necessary to modify the current, left or right cell of N. This requires two more moves.
Total number of moves of N are $O(|x|) + 6\frac{T(|x|)}{m}$. The value of $\varepsilon = 6/m$ implies that $m = \lceil \frac{6}{\varepsilon} \rceil$. QED.

If $f(n) > O(n)$ in the previous theorem, then N is bounded by $\varepsilon f(n)$ time. The argument of the proof is also true for a nondeterministic Turing machines.

There are similar theorems for space bounded Turing machines. If $f(n) \geq \Omega(\log n)$, then $DSPACE(f(n)) \subset DSPACE(\varepsilon f(n))$.

Some of the inclusion results of complexity classes are as follows:
Any deterministic Turing machine by definition is also a nondeterministic Turing machine. If a language $L \in DTIME(f(n))$, then $L \in NTIME(f(n))$ i.e. $DTIME(f(n)) \subseteq NTIME(f(n))$. Similarly we have $DSPACE(f(n)) \subseteq NSPACE(f(n))$.

Proposition 3. Every NTM N has an equivalent DTM.
Proof: We first simulate N on a 3-tape DTM M.
The 1st-tape of M is read-only and contains the input: $\triangleright x$. Initially the 2nd and the 3rd tapes are empty ($\triangleright$). At any intermediate point the 2nd-tape contains the snapshot of the tape of N and the 3rd tape keeps track of M's position at the computation tree of N, by a sequence of nondeterministic choices.

If d is the maximum degree of nondeterminism, then from any configuration of N, the nondeterministic choices to next configurations can be labeled by elements of $\Gamma_n = \{1, 2, \cdots, d\}$. And any string of length i over Γ_n is a sequence of choices made by N starting from its *start configuration (root)* upto a configuration at the i^{th} level of the computation tree. As an example, if the string present on the 3rd-tape is 113, then the computation of N has chosen the *1st child of the root*, *1st child* of the 1st child of the root, and *2nd child* of the 1st child of the 1st child of the root[1]. The simulation procedure is as follows:

(i) Tape-1 contains the input ($\triangleright x$)m Tape-2 and Tape-3 are empty.

(ii) Copy the input from Tape-1 to Tape-2. Initialize Tape-3 to *null* (ε).

[1]Some strings over Γ_n may not correspond to any computation.

(iii) If no more symbol on Tape-3, `goto 4`.
Otherwise, use Tape-3 to simulate N on the sequence of nondeterministic choices stored in Tape-2. If the current state is 'Y' - *accept* the input.
If the current state is 'N' or if the number present on Tape-3 does not correspond to any nondeterministic choice, discard the computation and `goto 4`.

(iv) Replace the string on Tape-3 by the next string and `goto 2`.

We already know that a 3-Tape DTM can be simulated on a single-tape DTM. So there is a DTM equivalent to the NTM. QED.

Proposition 4. $NTIME(f(n)) \subseteq DTIME(2^{O(f(n))})$ i.e.

If L is decided by a NTM N (single tape) in time $f(n)$, then it is decided by a DTM M in time $2^{O(f(n))}$.

Proof:

Let the running time of N be $f(n)$. The maximum degree of nondeterminism of a state-input pair is $d > 1$. The depth of the computation tree of N on an input of length n is at most $f(n)$. The maximum possible number of leaf nodes (*halt* configurations) of the computation tree is $d^{f(n)}$ and that is the upper bound on the number of computation paths.

We have already seen the simulation of N on a 3-tape DTM M. The number of internal nodes are $O(d^{f(n)})$. The number of steps from the root (start configuration) to a leaf (halt configuration) is bounded by $O(f(n))$. So the running time of M is $O(f(n)d^{f(n)}) = 2^{O(f(n))}$.

Note: $\log(f(n)d^{f(n)}) = \log f(n) + f(n)\log d = O(f(n)) \Rightarrow f(n)d^{f(n)}) = 2^{O(f(n))}$.

Simulating the 3-tape DTM on a single-tape DTM gives the running time $(2^{O(f(n))})^2 = 2^{2O(f(n))} = 2^{O(f(n)}$. QED.

Proposition 5. $NTIME(f(n)) \subseteq DSPACE(f(n))$.

Proof: Let $L \in NTIME(f(n))$ i.e. there is a nondeterministic $f(n)$ time bounded Turing machine M so that $L = L(M)$. Let maximum degree of nondeterminism is d.

The deterministic Turing machine N will do a DFS (depth-first search) on the computation tree (nodes are configurations) of M. Every current node will be created on the fly, starting from the *start configuration* using a string over $\{1, 2, \cdots, d\}$ of length $f(n)$ corresponding different nondeterministic choices. The space required to store the current configuration may take at most $f(n)$ cells. The choice string of length $f(n)$ will also take $f(n)$ space. So N can be designed to be $f(n)$ space bounded. QED.

We know that $DTIME(f(n)) \subseteq NTIME(f(n)) \subseteq DSPACE(f(n)) \subseteq NSPACE(f(n))$. This implies that

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{NPSPACE}.$$

Proposition 6. $NSPACE(f(n)) \subseteq DTIME(2^{O(f(n))})$.

Proof: Let $L \in NSPACE(f(n))$. There is an $f(n)$ space bounded NTM M that decides L. We already know that a configuration of a k-tape NTM is $(p, x_1, u_1, \cdots, x_k, u_k)$. The head on the *read-only* input tape can have $n + 1$ positions, where $n = |x|$. The output tape will say *yes* or *no* so its content is little. There are $|Q|$ many possibilities of the state. The choices for the content of remaining $k - 2$ tapes is less than $|\Sigma|^{2(k-1)f(n)}$. The total number of configurations is at most $n \times c_1^{f(n)} = c^{\log n + f(n)}$, where c depends on M.

The computation of M forms a configuration tree (graph) of at most $c^{f(n)+\log n}$ nodes. The job of the deterministic machine machine is to check whether an *accepting configuration reachable* from the *start configuration*. The *reachability* problem in a graph can be solved in $O(|G(V)|^2)$ time (think of DFS). So the time taken by the DTM is $k(c^{f(n)+\log n})^2 = kc^{2(f(n)+\log n)} = 2^{O(f(n))}$.

We can generate the configuration graph in any manner. The time for the generation of each configuration is $c_1 f(n)$ (length is $f(n)$). So the DTM runs for $f(n)c^{f(n)} = 2^{O(f(n))}$ time. QED.

If $f(n)$ is a polynomial, then

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{NPSPACE} \subseteq \mathbf{EXP}.$$

It is not known whether $NP \subseteq P$. But the following theorem due to Walter J Savitch ([WJS], 1970) proves that $NPSPACE = PSPACE$.

Theorem 1. (*Savitch*) If $f(n) \geq \log n$ and space constructible, then $NSPACE(f(n)) = DSPACE(O(f(n)^2))$.

The job is to simulate an $f(n)$ space bounded NTM on an $(f(n))^2$ space bounded DTM. The earlier method of simulating every branch of computation will not work in this case. The reason is, with $f(n)$ space of configuration, $2^{O(f(n))}$ steps with nondeterministic choices are possible. The storage of sequence of choice of length $2^{O(f(n))}$ requires $2^{O(f(n))}$ space.

Savitch's approach was to test whether a configuration C_2 can be reached from a configuration C_1 in time t using $f(n)$ space. There is a recursive procedure which reuses space.

Proof: Let the NTM M decides the language L using $f(n)$ tape cells. We already know that the running time of an $f(n)$ space bounded machine is bounded by $c^{f(n)}$, where c is a constant.

Let C_1, C_2 be two configurations of M. We write $C_1 \vdash_M^{\leq t} C_2$, if C_2 is obtained from C_1 in t or fewer steps in M. The length of a configuration is bounded by $f(n)$.

The deterministic machine will implement the following *recursive procedure*: *yield*(C_1, C_2, t)

1. If $t = 1$, then test (i) if $C_1 = C_2$ or (ii) C_2 is obtained from C_1 in one step. If any one of these conditions is satisfied, return *true*; otherwise return *false*.

2. If $t > 1$, for each C_3, a configuration of M on x using space $f(n)$, perform the next three steps.

3. Run *yield*$(C_1, C_3, \lfloor t/2 \rfloor)$.

4. Run *yield*$(C_3, C_2, \lceil t/2 \rceil)$.

5. If both (3) and (4) *true*, then return *true*.

6. return *false*

We claim that the following deterministic machine N accepts L in $O(f(n)^2)$ space.

N: input x

1. Prepare the start configuration C_s and the accepting configuration C_a of M for input x.

2. Compute $2^{f(n)}$.

3. Run *yield*$(C_s, C_a, 2^{f(n)})$.

4. *accept* if and only if *yield()* returns *true*.

Each of the configurations uses $O(f(n))$ space. Every recursive call uses $O(f(n))$ space to stack the environment. The depth of the call is $\log_2 2^{\alpha f(n)} = O(f(n))$. So the total space requirement is $O(f(n)) \times O(f(n)) = O(f(n)^2)$. QED.

If $f(n)$ is a polynomial, then $(f(n))^2$ is also a polynomial. Therefore $NPSPACE = PSPACE$.

References

[MS] *Theory of Computation* by Michael Sipser, (3rd. ed.), Pub. Cengage Learning, 2007, ISBN 978-81-315-2529-6.

[CHP] Computational Complexity by *Christos H Papadimitriou*, Pub. Addision-Wesley, 1994, ISBN 0-201-53082-1.

[SABB] *Computational Complexity, A Modern Approach* by Sanjeev Arora & Boaz Barak, Pub. Cambridge University Press, 2009, ISBN 978-0-521-42426-4.

[DCK1] *Theory of Computation* by Dexter C Kozen, Pub. Springer, 2006, ISBN 978-81-8128-696-3.

[RMK] R M Karp, *Reducibility Among Combinatorial Problems*, in R E Miller and J W Thatcher, ed. *Complexity of Computer Communications*, pp 85-103, Plenum, 1972.

[NPMJF] Nicholas Pippenger, And Michael J Fischer, *Relation Among Complexity Measures*, in JACM 26, 2 (April 1979), pp 361-381.

[WJS] Walter J Savitch, *Relationships between nondeterministic and deterministic tape complexities*, Journal of Computer and System Sciences 4 (2), pp 177-192 1970.