Hierarchy Theorems

Anton Bankevich

June 24, 2006

Abstract

In the paper the hierarchy theorems for certain classes of languages are proved. Several cases where hierarchy does not occur are discussed.

Contents

1 Introduction

The main idea of hierarchy theorems is as follows. We observe a time or space bounded class, e.g. $DTime(f(n))$ and two functions $f(n)$ and $g(n)$. The Hierarchy Theorem states what assumptions should be taken for $f(n)$ and $g(n)$ so that we could claim that $DTime(f(n))$ is not equal to $DTime(g(n))$.

2 Basic Definitions

In this section we shall define the complexity classes and the types of functions we are going to deal with.

Definition 2.1 *$DTime(f(n))$ is a set of languages which can be decided by a DTM in $f(n)$ steps. Here $f(n)$ is a time constructible function.*

Definition 2.2 *$DSpace(f(n))$ is a set of languages which can be decided by DTM using $f(n)$ space. Here $f(n)$ is a space constructible function.*

Definition 2.3 *$NTime(f(n))$ is a set of languages which can be decided by NTM in $f(n)$ steps. Here $f(n)$ is a time constructible function.*

Definition 2.4 *For any bounded complexity class $CC(f(n))$ and Ω, a set of functions, $CC(\Omega) = \cup_{f(n)\in\Omega} CC(f(n))$*

These definitions are certainly well known, but they are presented here because in literature $CC(f(n))$ is often used with the same meaning as in our definitions of $CC(O(f(n)))$. We give these definitions to introduce our notation.

Definition 2.5 *$f : N \to N$ is a time constructible function iff there is a DTM which if given an input consisting of 1^n constructs $f(n)$ and which prints it on the output tape in $f(n)$ time at most.*

Definition 2.6 *$f : N \to N$ is a space constructible function iff there is a DTM which if given an input consisting of 1^n constructs $f(n)$ and which prints it on the output tape using $f(n)$ space at most.*

These definitions are also well known, but in some textbooks they are slightly different. For example, one can find definitions where DTM does not print the value of $f(n)$, but it must work for exactly $f(n)$ time in the case of time constructible functions or use exactly $f(n)$ space in the case of space constructible functions. But it is quite clear that these definitions are equivalent.

All the proofs in this paper are very technical so we do not give the complete proofs but the main idea will always be explained.

3 Hierarchy theorems for DTime, DSpace and NTime

3.1 DTime

Theorem 1 *If $f(n)$, $g(n)$ are time constructible functions $f(n)log(g(n)) = o(g(n))$ and $f(n) > n$ for sufficiently large n ,then $DTime(O(f(n))) \subsetneq DTime(O(g(n)))$.*

Proof Obviously $DTime(O(f(n))) \subset DTime(O(g(n)))$, so we will prove that $DTime(O(f(n))) \neq DTime(O(g(n)))$. Therefore we shall construct DTM D such that $L(D)$ (the language which is decided by D) belongs to $DTime(O(g(n)))$ and does not belong to $DTime(O(f(n)))$. First we give the enumeration of all DTMs in the fallowing way: we enumerate the language and the states of a machine with binary numbers. DTM is just a number of rules which state where we should move the head and what we should print on the tape, so all these rules can also be written as a string in the alphabet $\{0, 1\}$. In this way we enumerate all DTMs. We define M_k as a DTM which corresponds to k in our enumeration.

Then we construct D. D will work with the alphabet $\{0, 1, \$\}$ and it will operate on the string S in the following way:

- If $S \neq k\$1^l$ for certain k and l, then D accepts S (here and thereafter, k as part of the string means binary representation of k).
- If $S = k\$1^l$, then D simulates M_k on S for $h(n)$ steps. h(n) will be defined later.
  - If M_k doesn't exist, D accepts S.
  - If M_k halts within this time and accepts input, D rejects S.
  - If M_k halts within this time and rejects input, D accepts S.
  - If M_k doesn't halt within this time, D accepts S.

1) $D \in DTime(O(g(n)))$

It is easy to see that the checking that the input string is correct takes O(n) time. Therefore, we do not want the second part of the algorithm to work too long. The question is, what additional factor for the duration of work of M_k the simulation needs. The answer is that this factor is logarithmic. It means that the simulation works within $O(h(n)log(h(n))$ time period. This simulation construction is complicated, so it will not be given here. Here we need $h(n)$ to be such that $h(n)log(h(n)) = o(g(n))$.

2) $D \notin DTime(O(f(n)))$

If $f(n) = o(h(n))$ then for all M_k which belong to DTime(f(n)) D will stop evaluation of M_k in time for l that is large enough. But when the simulation stops in time, D gives the answer opposite to the answer of M_k, therefore D is not equal to M_k for any M_k from $DTime(o(f(n)))$.

Now we find h(n) such that $f(n) \leq h(n)$ and $h(n)log(h(n)) = o(g(n))$. Assume $h(n) = \frac{g(n)}{log(g(n))}$. This will be the function which we want, because $f(n)log(g(n)) = o(g(n))$.h This completes the proof. □

Rem 1 *The method used in this proof is called diagonalisation: in this method we construct TM which, given some other TM and some word as input, simulates all operations of the given TM and calculates its own answer from the answer of the given TM. Then we just look at how our machine works if given itself as an input. With the help of this construction we are able to prove that certain classes do not coincide.*

The theorem without O also holds:

Theorem 2 *If $f(n)$, $g(n)$ are time constructible functions $f(n)log(g(n)) = o(g(n))$ and $f(n) > n$ for sufficiently large n, then $DTime(f(n)) \subsetneq DTime(g(n))$.*

Proof This theorem can be proved in the same way as the previous one. We only need to define $h(n)$ as $\varepsilon \frac{g(n)}{log(g(n))}$ for ε which is small enough. □

3.2 DSpace

Theorem 3 *If $f(n)$, $g(n)$ are space constructible functions $f(n) = o(g(n))$ and $f(n) > log(n)$ for sufficiently large n, then $DSpace(O(f(n))) \subsetneq DSpace(O(g(n)))$.*

Rem 2 *As we can see, here we do not need any additional logarithmic factor.*

Proof We use the same enumeration as in the proof of Theorem 1. Here we also have an obvious inclusion, so our aim is to prove that these classes are not equal. The construction of D is as follows(S is an input string here):

- If $S \neq k\$1^l$ for certain k and l, then D accepts S.
- If $S = k\$1^l$, then D simulates M_k on S for $h(n)2^{h(n)}$ steps while not more than $h(n)$ space is used.
  - If M_k does not exist, D accepts S.
  - If M_k halts after using not more than the allotted space and time and it accepts input, D rejects S.
  - If M_k halts after using not more than the allotted space and time and it rejects input, D accepts S.
  - If M_k does not halt in time or it uses extra space, D accepts S.

Here we have space and time bounds for the simulation. We need a time bound because if we do not have it one could find such an input S that D never stops. But our time bound is so large that if a certain TM operates during more than this time and does not use more than $h(n)$ space then it will never stop because it has to be in the same configuration at least twice.

1) $D \in DSpace(O(g(n)))$

This inclusion follows from the construction of D where on each step we do not use more than $h(n)$ steps. So we need $h(n)$ to be $O(g(n))$.

2) $D \notin DSpace(O(f(n)))$

Here we can say that for each TM from $DSpace(O(f(n)))$, D simulates it with no time or space problems in case $f(n) = o(h(n))$.

So here we can assume $h(n) = g(n)$. □

Theorem 4 *If $f(n)$ and $g(n)$ are space constructible functions $f(n) = o(g(n))$ and $f(n) > log(n)$ for sufficiently large n, then $DSpace(f(n)) \subsetneq DSpace(g(n))$.*

Proof Here we also need $h(n)$ just to have an additional constant $\varepsilon g(n)$, and this will yield the proof. □

3.3 NTime

Theorem 5 *If $f(n)$ and $g(n)$ are time constructible functions $f(n) = o(g(n))$ and $f(n) > n$ for sufficiently large n, then $NTime(O(f(n))) \subsetneq NTime(O(g(n)))$.*

Proof We shall prove this theorem for the case $f(n) = n$ and $g(n) = n^2$, because in this theorem there are even more technical details than in the previous ones.

First we define function $N(i) = 2^{2^{2^{2^{2^i}}}}$ and function $N^{-1}(i)$ such that $N(N^{-1}(i)) < i < N(N^{-1}(i) + 1))$.

We can enumerate all NTMs in the same way as DTMs. As above, we only need to construct NTM D such that $L(D) \in NTime(O(g(n)))$ and $D \notin NTime(O(f(n)))$. Here is the construction of D:

- If $S \neq 1^n$ for any n, then D accepts input.
- If $S = 1^n$ then D computes $i = N^{-1}(n)$
- If M_i does not exist, D accepts S.
- If $n \neq N(i+1)$, D simulates M_i on the string 1^{n+1} with the help of nondeterminism for $n^{1.5}$ steps. (It means that when M_i makes some nondeterministic choice, D makes the same choice and their computational trees coincide).
  - If Mi halts in time, D outputs the answer of M_i.
  - If M_i does not halt in time, D accepts S.
- If $n = N(i+1)$, D simulates M_i on the string $1^{N(i)+1}$ for $n^{1.5}$ steps, checking all the brunches.(Here we just look through all brunches of the computational tree).
  - If M_i halts in time, D outputs an answer opposite to M_i .
  - If M_i does not halt in time, D accepts S.

1)$D \in NTime(O(g(n)))$

We construct D in such a way that every step of our algorithm works within $O(g(n))$ time, so the entire algorithm operates within $O(g(n))$ time.

2) $D \notin NTime(O(f(n)))$

Let D be equal to M_k and let D belong to $NTime(O(f(n)))$. In this case D given M_k as an input works correctly (it means that it stops in time on every brunch of computation). Let $M(i)$ be the answer of D to 1^i for any Turing Machine M and number i. By construction of D for all $N(k)+1 \leq i < N(k+1) D(i) = M_k(i+1)$ and $D(N(k+1) \neq M_k(N(k)+1)$. But we assumed that $D = M_k$, so $D(N(k)+1) = D(N(k)+2) = \ldots = D(N(k+1)-1) = D(N(k+1)) \neq D(N(k)+1)$?!?. This proves the theorem. □

4 Cases where hierarchy does not occur

In this part we shall discuss cases where the function is not time or space constructible or it does not satisfy some other conditions from the hierarchy theorems.

4.1 GAP Theorem

Theorem 6 *There exists such a function $f : N \to N$ that: f can be calculated using a certain DTM (without time or space bounds). $DTime(f(n)) = DTime(2^{f(n)})$ and $f(n) > n$ for n that is large enough.*

Rem 3 *Here we can use any other constructible function instead of $2^{f(n)}$.*

Proof We use the enumeration of all DTMs, which has already been defined. We shall define a property $P(i,k)$ in the following way. $P(i,k)$ takes place if and only if each machine among M_1, M_2 $\ldots$, M_i on each input of length i either halts after fewer than k steps or it halts after more than 2^k steps or it does not halt at all. $P(i,k)$ property can be checked by the following algorithm:

Simulate all M_j for $j = 1 \ldots i$ on all inputs of length i (the number of these inputs for each machine will be finite, since the alphabets of our machines are finite) for 2^k steps. If all evaluations stopped in less than k steps or did not stop until the time was over, then $P(i,k)$ takes place; otherwise, it does not.

Let $k_j(i)$ be a number of sequences such that: $k_1(i) = 2i$ and $k_{j+1}(i) = 2^{k_j} + 1$. And let $N(i) = \sum \varsigma_k^i k = 1i$.

Let us consider a DTM which, if given the number n, will follow the algorithm below.

For all $j = 1 \dots N(i)$ it will decide $P(i, k_j(i))$. And if $P(i, k_j(i))$ takes place, our DTM will output it.

Since $P(i, k_j(i))$ will take place for at least one j, our algorithm will always output something and we define $f(i)$ as a result of this algorithm. Now we shall show why this $f(n)$ is a function we need.

For all i $P(i, f(i))$ occures. It means that each DTM M_i halts between $f(i)$ and $2^{f(i)}$ time only for a finite number of cases, but we can construct a DTM which will check these cases apart from all the others. Therefore, since this constant does not influence the asymptotic, we can claim that no language belongs to $DTime(2^{f(n)}) \setminus DTime(f(n))$, thus this completes the proof. □

4.2 Space theorems

In the space hierarchy theorem we have a bound for $f(n)$ that $f(n) \geq log(n)$. And now we shall prove that if we do not have this condition, the hierarchy does not occur. It means that for a function which is small enough, $f(n)$ $DSpace(O(f(n))) = DSpace(O(1))$.

Theorem 7 *1)* $DSpace(O(log(log(n)))) \subsetneq DSpace(O(1))$
2) For all $\varepsilon > 0$ $DSpace(O(log(log(n))^{1-\varepsilon})) = DSpace(O(1))$.

Proof q) Let us define language L as

$$\{0\dots00\$0\dots001\$0\dots010\$\dots\$1\dots11|\ \textit{where between two consequent \$ there are exactly k digits}\}$$

In this case the length of our string is $k2^k + 2^k - 1$. We are going to prove that L belongs to $DSpace(O(log(log(n))))$ and does not belong to $DSpace(O(1))$. We need the following lemma.

Lemma 4.1 *Let L be a language from* $DSpace(O(1))$*, then one can find n such that for each word x from L such that length of x is more than n x can be represented in the following way:* $x = abs$ *and for each integer number r* ab^rc *belongs to L too.*

This lemma is well known and we do not prove it here.

Now let L be from $DSpace(O(1))$. Then by the lemma we have a representation of our string $\{0\dots00\$0\dots001\$0\dots010\$\dots\$1\dots11$ as a concatenation of a b and c. So for large enough r we will have an increased length of our string, but the distance between two consequent signs \$ is still equal to k. But we know that the length of the string is equal to $k2^k + 2^k - 1$, so L can not be decided with $O(1)$ Space.

Now we need to construct a DTM which will decide L using only $O(log(log(n)))$ space. The algorithm is as follows:

- While we are not at the accepting or rejecting state, look through all integer i and for each i we shall observe the sequence of numbers, each number of which is formed of the last i digits before each \$ in our sequence.
- Check that this sequence is correct. It means that we shall check that the first number only consists of zeros; each number is equal to the previous plus one, and the last number only consists of units. It is easy to see that it can be done in $O(log(log(n)))$ space.
- If for some i our checking failed, we reject.
- If for some i the i^{th} digit from each \$ is \$ and for all numbers less than i our checking succeeds, we accept the input.

This algorithm decides our language with the help of $O(log(log(n)))$ space. It completes the proof of the first part of our theorem.

2) We shall now consider a DTM which decides a language L using not more than $O(log(log(n)))^{1-\varepsilon}$ space. Our aim is to prove that this language can also be decided with the help of $O(1)$ space.

It is well known that configuration of a Turing Machine is a set of a state of the machine, values of all its cells on the tapes and the positions of all heads. We define pseudo-configuration as a set of a state of the machine, values of all its cells on the tapes and the position of all heads except the one on the input tape and the value on the position of the heads on the input tape (we will call this head the main head). The performance of a Turing Machine can be characterized by the sequence of configurations which occur during the computation. But we shall consider the sequence of pseudo configurations.

Lemma 4.2 *If Turing Machine M decides a language L using $O(log(log(n))^{1-\varepsilon})$ space, then the number of sequences of pseudo configurations which can occur during the computation with a preset position of the main head on a string of length n is $o(n)$.*

Proof Let S be a number of states of our machine, and let C be the number of letters in the alphabet of our machine and $f(n) := O(log(log(n))^{1-\varepsilon})$. Then the number of pseudo configurations is $CSC^{f(n)}$. But if with a given position of all heads two pseudo-configurations occur twice, our machine will never stop, so the number of times when the main head is in the given position is less than the number of different pseudo-configurations. Therefore, the number of sequences is not greater than $(CSC^{f(n)})^{CSC^{f(n)}} = CS\exp(\ln(C)f(n)CSC^{f(n)}) = CS\exp(\exp(\ln\ln(C) + \ln(f(n)) + \ln(C)f(n))) = CS\exp(\exp(\ln\ln(C) + \ln(\ln(\ln(n))^{1-\varepsilon}) + \ln(C)\ln(\ln(n))^{1-\varepsilon}))) = O(\exp(\exp(\ln(\ln(n))^{1-\varepsilon/2})))) = o(n)$ □

Let N be such a number that for all $n > N$ the number of sequences is less than $n/2$. We want to show that M uses $O(log(log(N)))$ space.

Let S be a string of a length greater than N, such that it is the shortest input on which M uses more than $log(log(n))$ space. Then one can find three positions of the main head with the same sequence of pseudo-configurations. So our input can be represented in the following way: $S = \alpha a\beta a\gamma a\delta$. It can be readily seen that each pseudo-configuration which occurs during the computation of M on the input S should also occur during the computation of M on the input $S = \alpha a\beta a\delta$ or on the input $S = \alpha a\gamma a\delta$. So if M uses some amount of space when being run on S, then it uses the same amount of space on some shorter string but we assume that S is the shortest string on which M uses greater than $log(log(n))$ space, so we have the contradiction, and thus the theorem is proved. □

References

[1] C. Papadimitriou: Computational Complexity, Addison Wesley, 1994,

[2] E.A. Hirsch. Lecture notes. http://logic.pdmi.ras.ru/ hirsch/students/complexity1/

[3] Sanjeev Arora: Computational Complexity: A Modern Approach, Princeton University, 2006.