Hierarchy Theorems (a.e. versions)

Norbert Lis, based on lecture by Prof. Eric Allender

January 28, 1998

This lecture introduces a.e. hierarchy theorems, which say that in a given space/time class bounded by space/time constructible function f, there are languages which require space/time of f to compute almost all of their members.

1 Space

Hierarchy Theorem (a.e. version for space): *If $s(n) = \omega(1)$ is space-constructible, then there exists language $A \in DSPACE(s(n))$, such that*

$$\forall_{i \in N} \;\; L(M_i) = A \;\; \Longrightarrow \;\; s_i(x) = \Omega(s(|x|)). \tag{1}$$

Proof.

The proof will exhibit the algorithm for language A, and procede to show that A has properties stated in the theorem. Consider the following algorithm that takes as input a machine M and an input x:

1. On input (M, x):
2. compute $s(|x|)$
3. for $i \leftarrow 1$ to $s(|x|)$
4. in space no more than $s(|x|)$ search for a y such that
5. $|y| \leq s(|x|) \;\wedge\; y \leq x$, such that
6. $y \in L(M) \iff y \notin L(M_i)$
7. if no y is found, then
8. check if $M_i(x)$ can be simulated in $\leq s(|x|)$ space
9. if so, then accept $x \iff x \notin L(M_i)$
10. endfor
11. if control reaches here, then accept x

By the recursion theorem (which is stated and proved at the end of these notes) there is a machine M such that the output and space complexity of this algorithm on input (M, x) is the same as the output and space complexity of M on input x.

Let $A = L(M)$. Since each step in the algorithm is explicitely designed not to use more than $s(|x|)$ space on input x, A is clearly in $DSPACE(s(n))$.

To show (1), assume the opposite

$$\exists_{i \in N} \;\; L(M_i) = A \;\; \wedge \;\; \exists_x^{\infty} s_i(x) < \frac{s(|x|)}{i} \tag{2}$$

Let k be the least i for which (2) is true. Therefore, for all $j < k$, either $L(M_j) \neq L(M)$, or $L(M_j) = L(M)$ and $s_j(x) = \Omega(s(|x|))$. For each $j < k$, such that $L(M_j) \neq L(M)$, let y_j be the smallest input for which $M(x) \neq M_j(x)$, and let S be the set of these y_j's.

Consider some input α, which satisfies

$$s_k(\alpha) < \frac{s(|\alpha|)}{k} \;\; \wedge \;\; s(|\alpha|) > k \;\; \wedge \;\; s(|\alpha|) > \max_{y_j \in S}\{|y_j|\} \;\; \wedge \;\; \alpha > \max S \tag{3}$$

Notice what happens when M examines successive machines when run on α. Each machine M_j, $j < k$, either will not accept $L(M)$, or will accept $L(M)$ in space $\Omega(s(n))$.

If such M_j accepts $L(M)$, then y in line 4 will **never** be found, and control will reach line 8. If $M_j(\alpha)$ could be simulated in $\leq s(|\alpha|)$ space, line 9 would be executed, M would halt and α would have property $\alpha \in L(M)$ if and only if $\alpha \notin L(M_j)$, which would contradict the fact that $L(M) = L(M_j)$. Therefore, in line 8, $M_j(\alpha)$ cannot possibly be simulated in $\leq s(|\alpha|)$ space, and the execution will proceede to examine next machine M_{j+1}.

If such M_j does not accept $L(M)$, then y in line 4 **will** be found (because α was chosen to satisfy (3)), and the execution will proceede to examine next machine M_{j+1}.

Therefore, M will not halt if $i < k$, and eventually M_k will be considered. Since $L(M) = L(M_k)$, no y in line 4 will be found, control will reach line 8, where, since α was chosen to satisfy $ks_k(\alpha) \leq s(|\alpha|)$, $M_k(\alpha)$ can be simulated in $\leq s(|\alpha|)$ space. Therefore line 9 will be reached and M will halt, implying $\alpha \in L(M)$ if and only if $\alpha \notin L(M_k)$, which contradicts $L(M) = L(M_k)$.

Hence, (1) is true. □

2 Time

Hierarchy Theorem (a.e. version for time): *If $t(n) = \omega(1)$ is time-constructible, then there exists language $A \in DTIME(t(n))$, such that*

$$\forall_{i \in N} \;\; L(M_i) = A \;\; \Longrightarrow \;\; t_i(x)\log(t_i(x)) = \Omega(t(|x|)). \tag{4}$$

Proof.

The proof will exhibit the algorithm for language A, and procede to show that A has properties stated in the theorem. Consider the following algorithm that takes as input a machine M and an input x:

1. On input (M, x):
2. compute $t(|x|)$
3. for $i \leftarrow 1$ to $\log(|x|)$
4. in time no more than $\frac{t(|x|)}{2^i}$ search for a y, such that
5. $|y| \leq \log^*(|x|)$, such that

6. $y \in L(M) \iff y \notin L(M_i)$
7. if no y is found, then
8. check if $M_i(x)$ can be simulated in $\leq \frac{t(|x|)}{2^i}$ time
9. if so, then accept $x \iff x \notin L(M_i)$
10. endfor
11. if control reaches here, then accept x

By the recursion theorem (which is stated and proved at the end of these notes) there is a machine M such that the output and space complexity of this algorithm on input (M, x) is the same as the output and space complexity of M on input x.

Let $A = L(M)$. Time taken by each line of the algorithm is:

- computing $t(|x|)$ in line 2 takes $t(|x|)$ because t is time-constructible
- incrementing i takes constant time per increment, totalling $c_0 t(|x|)$
- time spent in the body of the loop is $c_i \frac{t(|x|)}{2^i}$ per i'th iteration, totalling

$$c_1 \frac{t(|x|)}{2^1} + c_2 \frac{t(|x|)}{2^2} + ... < \sum_{j=0}^{\infty} c_j \frac{t(|x|)}{2^j} = Ct(|x|)$$

Therefore, total time of execution of M is $Kt(|x|)$ for some constant K, and by the linear speed-up theorem, $L(M)$ is in $DTIME(t(n))$.

To show (4), assume the opposite

$$\exists_{i \in N} \;\; L(M_i) = A \;\; \wedge \;\; \exists_x^{\infty} t_i(x) \log(t_i(x)) < \frac{t(|x|)}{c_i 2^i}, \tag{5}$$

where c_i is some constant associated with simulating M_i on a universal Turing machine.

Let k be the least i for which (5) is true. Therefore, for all $j < k$, either $L(M_j) \neq L(M)$, or $L(M_j) = L(M)$ and $t_j(x) \log(t_j(x)) = \Omega(t(|x|))$. For each $j < k$, such that $L(M_j) \neq L(M)$, let y_j be the smallest input for which $M(x) \neq M_j(x)$, and let S be the set of these y_j's.

Consider some input α, which satisfies

$$t_k(\alpha) \log(t_k(\alpha) < \frac{t(|\alpha|)}{c_k 2^k} \;\wedge\; \log(|\alpha|) > k \;\wedge\; \log^*(|\alpha|) > \max_{y_j \in S}\{|y_j|\} \tag{6}$$

Notice what happens when M examines successive machines when run on α. Each machine M_j, $j < k$, either will not accept $L(M)$, or will accept $L(M)$ in time $\Omega(t(n))$.

If such M_j accepts $L(M)$, then y in line 4 will **never** be found, and control will reach line 8. If $M_j(\alpha)$ could be simulated in time $\leq \frac{t(|\alpha|)}{2^k}$, line 9 would be executed, M would halt and α would have property $\alpha \in L(M)$ if and only if $\alpha \notin L(M_j)$, which would contradict the fact that $L(M) = L(M_j)$. Therefore, in line 8, $M_j(\alpha)$ cannot possibly be simulated in time $\leq s(|\alpha|)$, and the execution will proceede to examine next machine M_{j+1}.

If such M_j does not accept $L(M)$, then y in line 4 **will** be found (because α was chosen to satisfy (6): notice that $\max_{y_j \in S}\{|y_j|\}$ is tiny comparing to the ammount of time allocated for lines 4,5,6), and the execution will proceede to examine next machine M_{j+1}.

Therefore, M will not halt if $i < k$, and eventually M_k will be considered. Since $L(M) = L(M_k)$, no y in line 4 will be found, control will reach line 8, where, since α was chosen to satisfy $c_k t_k(\alpha) \log(t_k(\alpha) < \frac{t(|\alpha|)}{2^k}$, $M_k(\alpha)$ can be simulated in time $\leq \frac{t(|\alpha|)}{2^k}$. Therefore line 9 will be reached and M will halt, implying $\alpha \in L(M)$ if and only if $\alpha \notin L(M_k)$, which contradicts $L(M) = L(M_k)$.

Hence, (4) is true. □

3 Recursion Theorem

The validity of recursive calls to itself by machine M in the proofs of the hierarchy theorems is assured by the following theorem.

Complexity-Theoretic Recursion Theorem: *Let $M_i(y, x)$ be a TM computing a 2-place function. Then,*

$$\exists_{e \in N}\ M_e(x) = M_i(e, x), \tag{7}$$

and the space and time complexity of $M_e(x)$ is the same as that of $M_i(e, x)$ (up to a constant factor).

Proof.
First observe the following facts:

1. There is a recursive function s, such that $\forall x \forall y \forall z\ M_{s(x,y)}(z) = M_x(y, z)$.
   Proof.
   Algorithm to compute s:
   On input x and y, return the index of the following algorithm:
   "On input z, call $M_x(y, z)$"
2. There is a recursive function c, such that $\forall i \forall j \forall x \forall y\ M_{c(i,j)}(x, y) = M_i(M_j(x), y)$.
   Proof.
   Algorithm to compute c:
   On input i and j, return the index of the following algorithm:
   "on input x and y:
   call $M_j(x)$
   call M_i on output of $M_j(x)$ and y"
3. There is a Turing machine M_α such that, if s is any recursive function, then $\forall y\ M_\alpha(y) = s(y, y)$.
   Proof.
   Algorithm M_α:
   On input y, duplicate y, and return $s(y, y)$

Now, note that
$$\begin{aligned} M_{s(c(j,\alpha),c(j,\alpha))}(x) &= M_{c(j,\alpha)}(c(j,\alpha), x) \\ &= M_j(M_\alpha(c(j,\alpha)), x) \\ &= M_j(s(c(j,\alpha), c(j,\alpha)), x) \end{aligned}$$

Letting $e = s(c(j,\alpha), c(j,\alpha))$ yields $M_e(x) = M_j(e, x)$ and completes the proof. □