198:538 Lecture Given on January 26th, 1998

Lecturer: Professor Eric Allender
Scribe: Sunny Daniels

February 7, 2002

1 Notation

We will begin by defining some basic notation that we will use throughout this course.

1.1 Turing Machines

We assume that the reader is familiar with the widely-used model of computation known as the *Turing Machine*. At this stage in the course, we will assume that all Turing Machines are deterministic, and consist of:

1. A read-only *input tape*.
2. A (single) *finite-state control unit*.
3. A finite number of *work tapes*.

In some cases, we will designate one of the Turing Machine's work tapes as the Turing Machine's *output tape*. Each tape will have only one head. All of the heads will be under the direct control of the finite-state control unit, as shown in figure 1. Since there are only countably infinitely many Turing Machines, we can fix an enumeration of all of the Turing Machines:

$$M_1, M_2, M_3, \ldots \tag{1}$$

At the moment, this will be the notation for the enumeration of all of the *deterministic* Turing Machines. Later in the course, when we consider other types of Turing Machine (e.g. nondeterministic, alternating, probabilistic), we will use the same notation for an enumeration of all Turing Machines of one of these other types. We will assume that the type of Turing Machines that we are enumerating will always be clear from the context.

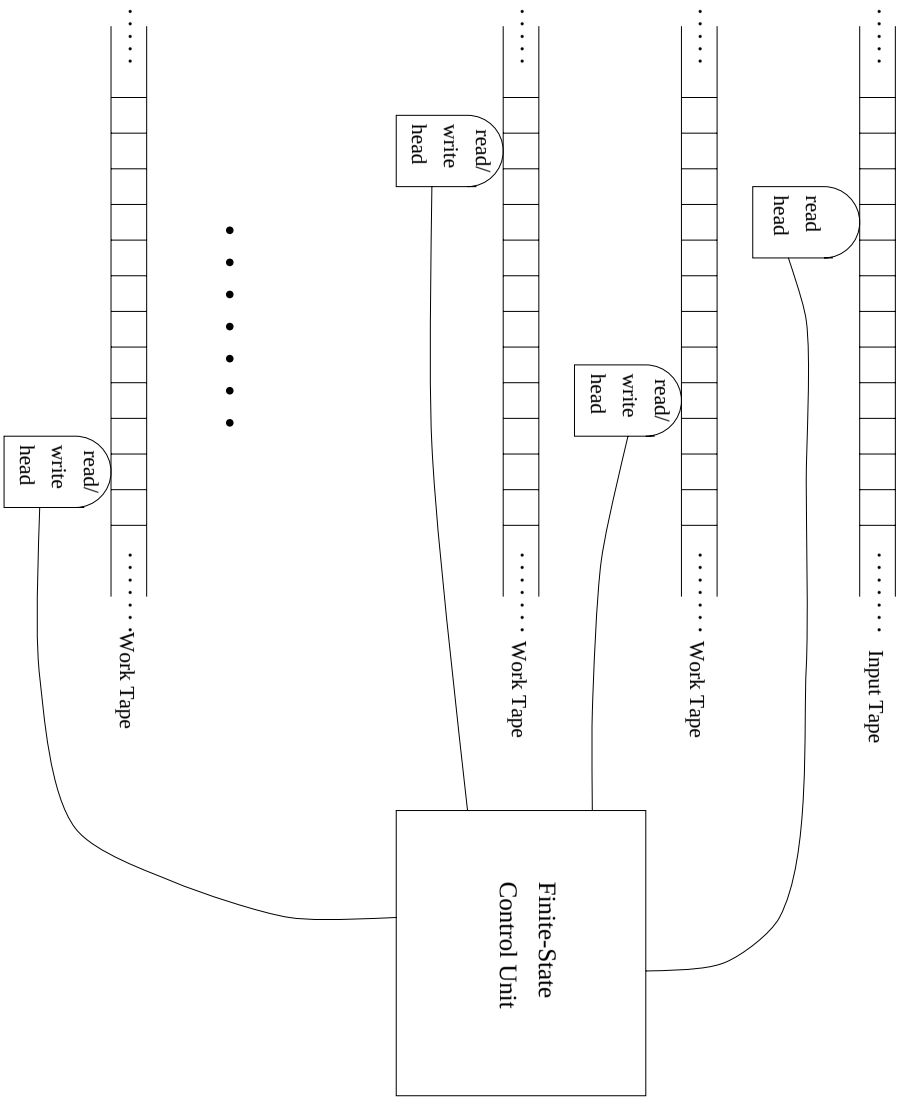

Figure 1: A Turing Machine

1.2 Measuring Space used by a Turing Machine

Take any Deterministic Turing machine M_i. Take any input string x for M_i. Define the *space used by M_i on x* to be the total number of work-tape cells, over all of M_i's work tapes, that M_i writes to during its computation if it is started with input x. We will denote the space used by M_i on x by $s_i(x)$. If, when M_i is started on input x, M_i runs forever *and* uses infinitely many work-tape squares, then we define $s_i(x)$ to equal infinity.

1.3 Measuring Time used by a Turing Machine

Take any Deterministic Turing machine M_i. Again, take any input string x for M_i. Define the *time used by M_i on x* to be the number of computation steps that M_i performs, if it is started with input x, before it halts. We will denote the time used by M_i on x by $t_i(x)$. If M_i never halts when it is started on input x, then we define $M_i(x)$ to equal infinity.

1.4 Space-Constructible Functions

Take any function $s : \mathbb{N} \rightarrow \mathbb{N}$. We say that $s(n)$ is *space-constructible* if and only if there is some Turing Machine M_i for which:

1. M_i halts on any input string x.
2. For any input string x to M_i, $s(x) = s_i(|x|)$.

2 Some elementary theorems about Turing Machines

2.1 Running Time Upper Bound

Theorem 1 *Let M_i have:*

1. *k tape symbols (both work-tape symbols and input-tape symbols).*
2. *c states in its finite-state control unit.*
3. *τ tapes, including the input tape.*

Then, if, for some input x, M_i runs for more than $ck^{\tau l}l^{\tau}$ steps without using more than l tape squares (i.e. without "touching" more than l tape squares with its heads, including input tape squares), then M_i does not halt on input x. Furthermore, in this case, $s_i(x) \leq l$.

Proof: Suppose that, for some input x, M_i uses no more than l space but runs for more than $ck^{\tau l}l^{\tau}$ steps. Clearly, each configuration of M_i will be completely specified by the following information:

1. The state of M_i's finite-state control unit. There are c possible values of this.
2. The contents of the portion of M_i's tapes that M_i has used. Each of the τ tapes has at most l used tape squares. Furthermore, each tape square contains one of k possible tape symbols. Therefore, there are at most $k^{\tau l}$ possibilities for the contents of these tapes.
3. The positions of M_i's heads. Since there are τ tapes, each of which has a head in one of at most l positions, there are at most $k(\tau^l)$ possibilities for the positions of the heads.

Hence, there are at most $ck^{\tau l}l^{\tau}$ different configurations that M_i will pass through, in its execution with the input x. But since M_i runs for more than $ck^{\tau l}l^{\tau}$ steps, the pigeonhole principle implies that M_i must pass through some configuration C more than once in its execution. But, since the execution sequence of M_i from any given configuration C is uniquely determined by C (because M_i is deterministic), this implies that the sequence of steps that takes M_i from the first occurrence of C to the second occurrence of C must be repeated after the second occurrence of C, and hence must lead to a third occurrence of C. But then, by the same reasoning, this sequence of execution steps must lead from this third occurrence of C to a fourth occurrence of C, and so on. Therefore, the execution sequence of M_i on input x must be infinite. Hence, M_i does not halt on input x, as required. Furthermore, since M_i's execution will not go through any configurations other than the configurations of M_i for its first $ck^{\tau l}l^{\tau}$ execution steps, this means that M_i will never use more than l tape squares. Hence $s_i(x) \leq l$.

2.2 The Gap Theorem

We will now prove another theorem, which will illustrate the importance of our notion of *space-constructibility*:

Theorem 2 *Let $r : \mathbb{N} \to \mathbb{N}$ be* any *recursive (i.e. computable) function for which, for all $n \in \mathbb{N}$, $r(n) \geq n$. Then there exists a recursive (i.e. computable) function $s : \mathbb{N} \to \mathbb{N}$ for which:*

$$\text{DSPACE}(s(n)) = \text{DSPACE}(r(s(n)))$$

Proof: We will start by making some preliminary definitions, and proving some preliminary results:

2.2.1 Space bounds for individual input lengths

We will first define a sequence of functions:

$$s_0 : \mathbb{N} \to \mathbb{N}$$
$$s_1 : \mathbb{N} \to \mathbb{N}$$
$$s_2 : \mathbb{N} \to \mathbb{N}$$
$$\ldots$$

by defining, for every $i, n \in \mathbb{N}$:

$$s_i(n) = \max_{x \in \Sigma^* : |x| = n} s_i(x) \tag{2}$$

where we assume, WLOG, that all of the Turing Machines in our enumeration are over the same input alphabet Σ^*.

Lemma 1 *For each $i, q, n \in \mathbb{N}$, the problem of determining whether or not $s_i(n) \leq q$ (where $s_i(n)$ is as defined in equation 2) is decidable.*

In order to show that this lemma is true, take any $i, q, n \in \mathbb{N}$. Then, we can define a Turing Machine $M_{(i,q,n)}$ which steps through each $x \in \Sigma^n$. For each $x \in \Sigma^n$, the machine $M_{(i,q,n)}$ simulates M_i's execution on the input x until one of the following three things happens:

1. M_i uses $> j$ space. In this case, we know that $s_i(x) > j$.
2. M_i's execution halts. In this case, we can look at M_i's tapes and count up the used tape squares to determine whether or not $s_i(x) \leq j$.
3. M_i runs for more than $ck^{\tau l}l^{\tau}$ execution steps without using more than j tape squares. Then, by theorem 1, $s_i(x) \leq j$.

The Turing Machine $M_{(i,q,n)}$ can hence determine whether or not $s_i(n) \leq j$, based on the results of the above simulations.

2.2.2 Another important property of $s_i(n)$:

Lemma 2 *Take any function $r : \mathbb{N} \to \mathbb{N}$ with $r(m) \geq m$ for all $m \in \mathbb{N}$. Then, for every $n \in \mathbb{N}$, there is some $j \in \mathbb{N}$ for which the following two sets are disjoint:*

$$\begin{aligned} R_j &= \{q : j < q \leq r(j)\} \\ S_n &= \{s_i(n) : i \in \mathbb{N}, i \leq n, s_i(n) < \infty\} \end{aligned}$$

Proof: Take any $n \in \mathbb{N}$. If $S_n = \emptyset$, then R_j and S_n are trivially disjoint, regardless of what value of j we pick. If, on the other hand, S_n is not empty, then let j be the maximum element of S_n (which S_n must have, since S_n is finite). Then any element of S_n must be $\leq j$, while any element of R_j must be strictly greater than j. Hence, S_n and R_j are also disjoint in this case.

2.2.3 Machine for computing the function s

In the previous subsection, we showed that for every $n \in \mathbb{N}$, there is a $j \in \mathbb{N}$ for which R_j and S_n are disjoint. We will now define a Turing Machine M_s to compute such a $j \in \mathbb{N}$ for any $n \in \mathbb{N}$. We will let M_s be the Turing Machine that executes the following program:

```
begin
  j := 1
  repeat
    i := 1
    while i ≤ n but not j < s_i(n) ≤ r(j) do
      i := i + 1
    if i ≤ n then
      j := s_i(n)
    else
      exit
  until exit
  output j
end
```

Now, clearly the "$i := 1$" line, and the following while loop have the effect of finding the minimum natural number $i \in \mathbb{N}$ for which $i \leq n$ and $s_i(n) \in R_j$, or setting $i = n + 1$ if there is no such natural number i. But $s_i(n) \in R_j$ implies that $j < s_i(n)$, and $i = n + 1$ implies that $i > n$, so the following **if** statement is certain to either increase j or terminate the program's execution. Since j increases with each iteration until the loop terminates, if the outer loop executes for sufficiently long then j will become greater than the maximum element of S_n (which S_n must have, since it is finite). Then inner loop will then fail to find any $i \in \mathbb{N}$ for which $i \leq n$ and $s_i(n) \in R_j$. Hence, the inner loop will set $i = n + 1$, and so the outer loop will terminate.

So the program M_s will certainly terminate for any input n. Now, for the program to terminate, we must have $i > n$ at the beginning of the **if** statement in the outer loop. This means that the preceding inner loop must have failed to find any natural number $i \in \mathbb{N}$ for which $i \leq n$ and $s_i(n) \in R_j$. The inner loop has therefore found that none of the elements of S_n are also elements of R_j. Hence, S_n and R_j are disjoint.

Hence, the function $s : \mathbb{N} \to \mathbb{N}$ does indeed satisfy the required property that S_n and $R_{s(n)}$ are disjoint, for every $n \in \mathbb{N}$.

2.2.4 Conclusion of Proof

Firstly, note that we assume that $r(n) \geq n$ for all $n \in \mathbb{N}$. Therefore:

$$\text{DSPACE}(s(n)) \subseteq \text{DSPACE}(r(s(n)))$$

It hence remains only to show that $\text{DSPACE}(r(s(n))) \subseteq \text{DSPACE}(s(n))$. To show that this is true, take any Turing machine M_j with $L(M_j) \in \text{DSPACE}(r(s(n)))$. Then $s_j(n)$ is bounded above by $r(s(n))$ for all $n \in \mathbb{N}$. The definition of S_n implies that $s_j(n) \in S_n$, for all $n \in \mathbb{N}$ with $n \geq j$. So take any $n \in \mathbb{N}$ with $n \geq j$. Now, from the result in the previous subsection, we have that S_n and $R_{s(n)}$ are disjoint. Hence $s_j(n) \notin R_{s(n)}$. Since $s_j(n) \leq r(s(n))$, this implies that $s_j(n) \leq s(n)$.

Now, we have established that $s_j(n) \leq s(n)$ for all $n \in \mathbb{N}$ with $n \geq j$. The constant-time speed-up theorem therefore implies that $L(M_j) \in \text{DSPACE}(r(s(n)))$. This proves Theorem 2.

2.3 The Infinitely-Often Space Hierarchy Theorem

Theorem 3 *Let $s(n)$ be space-constructible. If $s`(n) = o(s(n))$ then*

$$\mathit{DSPACE}(s`(n)) \subsetneq \mathit{DSPACE}(s(n))$$

Proof: Since $s`(n) = o(s(n))$, the fact that $\text{DSPACE}(s`(n)) \subseteq \text{DSPACE}(s(n))$ follows from the linear-time speed-up theorem. So it remains only to show that $\text{DSPACE}(s`(n)) \neq \text{DSPACE}(s(n))$. We will do this below:

2.3.1 Proof that $\text{DSPACE}(s`(n)) \neq \text{DSPACE}(s(n))$:

To prove this, we will assume that it is false and derive a contradiction. So assume that, for some function $s` : \mathbb{N} \to \mathbb{N}$ with $s`(n) = o(s(n))$, we have:

$$\text{DSPACE}(s`(n)) = \text{DSPACE}(s(n)) \tag{3}$$

Now, define the Turing Machine M_k as follows:

1. Let x be the contents of the input tape.
2. Let j be the maximum non-negative integer for which 0^j is a prefix of x.
3. Use the fact that s is space-constructible to mark off $s(|x|)$ work tape squares.
4. *Simulate* M_j on input x using at most $s(|x|)$ work tape squares.
5. If M_j accepts x then reject else accept.

Now, since M_k is defined in such a way that it uses no more than $s(n)$ space, we know that $s_k = s$. Hence, $L(M_k) \in \text{DSPACE}(s(n))$, and so, by equation 3, $L(M_k) \in \text{DSPACE}(s`(n))$. So let $M_j$ be the Turing Machine for which both $L(M_k) = L(M_j)$ and $s_j = s`$. Now, since we have already established that $s_k = s$, we know that $s_j(n) = o(s_k(n))$. Therefore, there will be some $m \in \mathbb{N}$ for which $s_j(n) \leq s_k(n)$ for all $n \geq m$. Hence, for all sufficiently-long strings y, $|0^j 1y|$ will be greater than m, and hence it will be possible for M_k to simulate M_j on input $0^j 1y$ in space $s_k(|0^j 1y|)$. Hence, for any such input $0^j 1y$, M_j will accept $0^j 1y$ if and only if it rejects $0^j 1y$. This is clearly a contradiction! Therefore, we must conclude that theorem 3 is true.

2.3.2 Compare this to a.e. space hierarchy

Theorem 3 is referred to as the *infinitely-often space hierarchy theorem* in order to distinguish it from a conceptually-similar, but stronger, result that we will prove in the next lecture. This stronger result is known as the *almost-everywhere space hierarchy theorem.*