Computability and Complexity

Lecture 9, Tuesday August 29, 2023

Ari Feiglin

9.1 Spatial Complexity

Definition 9.1:

We define a Turing machine with three tapes,

(1) The input tape, which is read-only. We can move forward and backward on this tape.

(2) The output tape, which is write-only. For comfort, we assume we only move forward on this tape. But this is equivalent to also being able to move backward.

(3) And the working tape, which is read-write. We can move bidirectionally on this tape.

Given a space function $S\colon \mathbb{N} \longrightarrow \mathbb{N}$, we define the class $\mathsf{DSPACE}(S(n))$ to be the set of all decision problems A which can be decided by a Turing machine M (with three tapes as above) such that for every x, $M(x)$ utilizes at most $S(|x|)$ cells on its work tape.

Similarly for a time function $t\colon \mathbb{N} \longrightarrow \mathbb{N}$ we define the class $\mathsf{DTIME}(t(n))$ to be the set of all decision problems A which can be decided by a Turing machine M whose runtime is bound by $t(n)$.

Notice that

$$\mathbf{P} = \bigcup_{p(n)\text{ polynomial}} \mathsf{DTIME}(p(n))$$

And furthermore, $\mathsf{DTIME}(t(n)) \subseteq \mathsf{DSPACE}(t(n))$ since if our runtime is bound by $t(n)$, we can only perform at most $t(n)$ writes and therefore can only utilize at most $t(n)$ cells in the working tape.

Proposition 9.2:

For every spatial function S,

$$\mathsf{DSPACE}(S(n)) \subseteq \mathsf{DTIME}\Big(n \cdot 2^{O(S(n))}\Big)$$

Notice then if $S(n) \geq \log(n)$, then

$$\mathsf{DSPACE}(S(n)) \subseteq \mathsf{DTIME}\Big(2^{O(S(n))}\Big)$$

as then $n \leq 2^{S(n)}$ so

$$n \cdot 2^{c\cdot S(n)} \leq 2^{(c+1)S(n)}$$

This means that $n \cdot 2^{O(S(n))} = 2^{O(S(n))}$.

Proof:

Recall the definition of the *configuration* of a Turing machine after t steps. It is a tuple (or a string) which contains the content of the tapes, the placement of the heads, and the current state. In the case of our input-output-work Turing machine, our configuration needs only

(1) The placement of the heads on the input and work tapes. We don't need the placement of the head on the output tape, as we cannot read from it.

(2) The contents of the work tape. We don't need the content of the input tape, as it is constant. And we don't need the content of the output tape, as we cannot read from it.

(3) The internal state of the Turing machine.

If at any point we get the same configuration twice as different times, then since the next steps are determined entirely by the configuration, we would enter a never-ending loop. So each configuration must be different.

Now let us consider the number of possible configurations. There are n possible placements on the input tape, and $S(n)$ possible placements on the work tape. There are $2^{S(n)}$ possible strings which can be the content of the work tape, and the number of internal states is some constant c. Thus the total number of possible configurations is

$$n \cdot S(n) \cdot 2^{S(n)} \cdot c$$

and this is in $n \cdot 2^{O(S(n))}$ (a better bound would be $O(n \cdot 2^{S(n)})$), and since every configuration is unique, there can therefore be at most $n \cdot 2^{O(S(n))}$ steps, meaning the decision problem is in $\mathsf{DTIME}\left(n \cdot 2^{O(S(n))}\right)$ as required. ■

Definition 9.3:

Let us define the class of decision problems

$$\mathbf{L} = \bigcup_{c=1}^{\infty} \mathsf{DSPACE}(c \cdot \log(n))$$

the class of decision problems which use logarithmic space.

Since

$$\mathsf{DSPACE}(c\log(n)) \subseteq \mathsf{DTIME}\left(2^{O(\log(n))}\right) = \mathsf{DTIME}\left(n^{O(1)}\right) = \mathbf{P}$$

we have that

$$\mathbf{L} \subseteq \mathbf{P}$$

Whether or not this is a proper inclusion is an open problem. It is hypothesized that it is proper, but this has not been proven.

Theorem 9.4 (The Space Hierarchy Theorem):

For every two space functions $S_1, S_2 \colon \mathbb{N} \longrightarrow \mathbb{N}$ such that $S_1(n) \geq \log(n)$ and $S_2(n) \in \omega(S_1(n))$ (meaning $S_1(n) \in o(S_2(n))$). Then

$$\mathsf{DSPACE}\big(O\big(S_1(n)\big)\big) \subsetneq \mathsf{DSPACE}\big(O\big(S_2(n)\big)\big)$$

In other words, if $S_2(n)$ is asymptotically larger than $S_1(n)$, there exist decision problems which can be solved with $S_2(n)$ space, but not with $S_1(n)$ space.

Proof:

Since $S_1(n) \in o(S_2(n))$, obviously there is inclusion. We will show that there exists a problem solvable in $O(S_2(n))$ space but not in $O(S_1(n))$ space. We define

$$A = \left\{ (M, \omega) \;\middle|\; \begin{array}{l} M \text{ is an input-output-work Turing machine which accepts } \omega \text{ within } 2^{S_2(n)} \text{ steps and} \\ \text{also uses at most } S_2(n) \text{ cells in its work tape.} \end{array} \right\}$$

(Importantly, $n = |(M, \omega)|$.)

Firstly, we must show that $A \in \mathsf{DSPACE}(O(S_2(n)))$. We define a Turing machine B which accepts (M, ω) and runs M on its work tape for at most $2^{S_2(n)}$ steps. If M accepts ω within $2^{S_2(n)}$ steps, it returns one, and otherwise zero. It must allocate room on its work tape for M's work, which utilizes at most $S_2(n)$ cells, and for counting the number of steps it has used it must utilize another $\log(2^{S_2(n)}) = S_2(n)$ cells (it stores it in binary, which is why it is logarithmic). So all in all, B decides A and utilizes at most $2S_2(n)$ cells on its work tape, meaning $A \in \mathsf{DSPACE}(O(S_2(n)))$ as required.

Now suppose, for the sake of a contradiction, that $A \in \mathsf{DSPACE}(O(S_1(n)))$, so there exists Turing machine M_A which decides A and has spatial complexity bound in $O(S_1(n))$. So suppose M_A has spatial complexity bound by $cS_1(n)$ for $c > 0$. Since $S_1(n) \geq \log(n)$,

$$\mathsf{DSPACE}\, O(S_1(n)) \subseteq \mathsf{DTIME}\left(2^{O(S_1)}\right)$$

and so there exists a c such that A has time complexity bound by $2^{c' S_1(n)}$ for $c' > 0$. Let us define $C = \max\{c, c'\}$. Since $S_2(n) \in \omega(S_1(n))$, there exists an n_0 such that for every $n \geq n_0$,

$$S_2(n) \geq C \cdot S_1(n)$$

So for every input of size $n \geq n_0$, M_A utilizes at most $S_2(n)$ space and runs in at most $2^{S_2(n)}$ time.

Suppose we enumerate all the Turing machines $\{M_i\}_{i=1}^{\infty}$. We define a Turing machine D which gets as input a Turing machine M_i and returns the inverse of $M_A(M_i, M_i)$. Further let us assume that the length of D is at least n_0, we can do this by adding redundant code. D is a Turing machine, and so suppose $M_k = D$, then what does $D(D)$ return?

(1) If $D(D)$ returns one, $M_A(D,D) = 0$ so $D \notin A$. This means that D doesn't accept D within $2^{S_2(n)}$ steps and utilizing $S_2(n)$ space. But since $|D| \geq n_0$, $M_A(D,D)$ utilizes at most $S_2(n)$ space and runs in at most $2^{S_2(n)}$ time. But all $D(D)$ does is run $M_A(D,D)$ so it also utilizes $S_2(n)$ space and runs in $2^{S_2(n)}$ time, which means that D rejects D, which contradicts $D(D) = 1$.

(2) If $D(D) = 0$ then $M_A(D,D) = 1$, which means that $(D,D) \in A$, but this means D accepts D and so $D(D) = 1$ in contradiction.

So D cannot exist, but it is well-defined from M_A, which means M_A cannot exist so $A \notin \mathsf{DSPACE}(O(S_1(n)))$. Therefore we have shown

$$\mathsf{DSPACE}\big(O\big(S_1(n)\big)\big) \subsetneq \mathsf{DSPACE}\big(O\big(S_2(n)\big)\big) \qquad \blacksquare$$

Note:

This is a diagonal argument, similar to Cantor's diagonal argument proving $\aleph_0 < \mathfrak{c}$ and the diagonal argument showing that A_{TM} is undecidable.

Definition 9.5:

Given a space function $S\colon \mathbb{N} \longrightarrow \mathbb{N}$, we define the class $\mathsf{NSPACE}(S(n))$ to be the set of all decision problems A such that there exists a non-deterministic Turing machine M where for every x, the maximum number of cells M uses while running on M (meaning the maximum is taken over all the possible paths taken by M while running x) is bound by $S(|x|)$.

Similarly we define

$$\mathbf{NL} = \bigcup_{c=1}^{\infty} \mathsf{NSPACE}(c\log(n))$$

Definition 9.6:

A **log-space reduction** from one decision problem A_1 to another A_2 is a function f which can be computed in logarithmic space such that

$$x \in A_1 \iff f(x) \in A_2$$

Definition 9.7:

We say that a decision problem A is **NL-hard** if for every $A' \in \mathbf{NL}$ there exists a log-space reduction from A' to A. If an $\mathbf{NL}$-hard decision problem is also in $\mathbf{NL}$, it is called $\mathbf{NL}$-complete.

Proposition 9.8:

$\mathbf{L}$ is closed under log-space reductions.

Proof:

Let $A \in \mathbf{L}$ and suppose there exists a log-space reduction from A' to A, f. So there exists a Turing machine M_A which decides A in logarithmic space, and f can also be computed in logarithmic space. The issue is, $|f(x)|$ is polynomial in $|x|$ and so if we were to want to run $M_A(f(x))$ we would need to compute $f(x)$ which takes up polynomial space.

So instead, we will define a Turing machine $M(x)$ which simulates $M_A(f(x))$ without storing the entire value of $f(x)$. We do this by having M store the position of M_A's pointer to its input tape (there are $\log|f(x)|$ possibilities, if we store the value in binary, which is logarithmic in $|x|$), and at every step of M_A, we compute $f(x)$. But instead of saving the result, since the result is written sequentially (we can only move forward on the output tape), we compute the first character in $f(x)$, erase it, then compute the next, and so on until we reach the character pointed to by M_A's pointer.

The space needed is:

(1) Computing $f(x)$ takes $O(\log|x|)$ space.

(2) Storing M_A's pointer takes $\log|f(x)| \in O(\log|x|)$ space.

(3) Computing $M_A(f(x))$ takes $O(\log|f(x)|) \subseteq O(\log|x|)$ space.

(4) Plus we need a single cell for the current character of $f(x)$.

So all in all everything need $O(\log|x|)$ space, so M requires $O(\log|x|)$ space. Therefore A' can be decided by the logarithmic-space Turing machine M, meaning $A' \in \mathbf{L}$ as required. ■

This means that if there exists an **NL**-complete problem in **L**, **NL** = **L**. This is since if A is **NL**-complete and in **L**, then for every $A' \in \mathbf{NL}$ there exists a log-space reduction from A' to A which means that $A' \in \mathbf{L}$ since **L** is closed under log-space reductions. Thus $\mathbf{NL} \subseteq \mathbf{L}$, and $\mathbf{L} \subseteq \mathbf{NL}$ trivially.

So similar to how we study **NP**-complete problems to understand the relation between **P** and **NP**, we can study **NL**-complete problems to understand the relation between **L** and **NL**.

Proposition 9.9:

There exists an **NL**-complete problem.

Proof:

Let us define

$$\mathsf{st\text{-}conn} = \{(G, s, t) \mid G \text{ is a directed graph, } s \text{ and } t \text{ are nodes in } G \text{ which are connected}\}$$

We will define a non-deterministic algorithm which decides st-conn.

```
1. function $M(G = (V, E),\ s,\ t)$
2.     $current \leftarrow s$
3.     repeat $|V|$ times
4.         choose $next$ to be a neighbor of $current$.
5.         if $(next = t)$  return 1
6.         $current \leftarrow next$
7.     end repeat
8.     return 0
9. end function
```

This uses constant space, and it decides st-conn. Thus st-conn $\in \mathbf{NL}$ as required.

Now we must show that st-conn is **NL**-complete. Let $A \in \mathbf{NL}$, suppose it is decided by a non-deterministic log-space Turing machine M_A. Given an input x, we define the graph (G, s, t) where

(1) The vertices in G are all the possible configurations of M_A when running x.

(2) We define an edge from a configuration c_1 to c_2 if and only if it is possible to go from c_1 to c_2 in a single move.

(3) We define s to be the initial configuration.

(4) We define a new vertex t and add an edge from every configuration which is accepted by M_A to t.

Obviously $x \in A$ if and only if $(G, s, t) \in \mathsf{st\text{-}conn}$, so if we can show that computing $f(x) = (G, s, t)$ can be done in log-space, then this forms a log-space reduction.

At every point, we need only to store the current configuration and the next configuration. To store a single configuration we need to store:

(1) The placement of the pointer on the input tape, there are n positions so this takes $\log(n)$ space.

(2) The placement of the pointer of the working tape, and since M_A is log-space, this takes $O(\log(n))$ space.

(3) The content of the working tape, and since M_A is log-space, its length is bound by $O(\log(n))$ space.

(4) The internal state, which takes constant space.

So the space needed is $O(\log(n))$ so f is log-space, and therefore a log-space reduction. Therefore st-conn is **NL**-complete, as required. ■

But notice that st-conn is in **P** (recall from our algorithms course), and every log-space reduction is also a Karp reduction (since something which takes logarithmic space takes polynomial time), and so this means

Proposition 9.10:

$$\mathbf{NL} \subseteq \mathbf{P}$$