

Lecture 2: Space Complexity

Lecturer: Kristoffer Arnsfelt Hansen

Scribe: Rocio Santillan Rodriguez

1 Sublinear space bounds

If we consider the total of cells read by the TM, we will not get less than space $O(n)$ computation. However, there are interesting classes that use sublinear space if we ignore the input. Therefore, we consider a specific model for space bounded computation.

The model consists of a multitape Turing machine where the first tape is an input tape, with a read-only two-way head. The rest of the tapes are work tapes, with a read/write head.

We define $space_M(x)$ as the number of *work* cells used by the machine M .

Observe that if we have 0-space used, basically we get a two-way finite automaton.

Theorem 1 *Every language recognized by a two-way finite automaton is regular.*

Proof See the proof in Theorem 3.13 in Hopcroft-Ullman. □

Corollary 2 *Every language recognized by a constant space bounded machine is regular.*

Theorem 3 *Every language recognized by an $o(\log \log n)$ space-bounded machine is regular.*

Proof The idea of the proof is to use crossing sequences which not only store the states, but also the symbols and the positions of the head. Thus, a *storage configuration* represents the three elements: the state, the contents of the work tape and the position of the worktape head.

Assume that M is $S(n)$ space-bounded with one work tape. We know that

$$\# \text{ storage configurations} = |Q| \cdot |\Gamma|^{S(n)} \cdot S(n) \leq c^{S(n)} \text{ for a fixed constant } c.$$

Let $cs_i(x)$ be the sequence of storage configurations when moving between i and $i + 1$. Assume that $S(n)$ is not bounded by a constant.

Given k , let x be the minimum length input that makes M use k cells of workspace. Let us divide the input by half. There are two cases.

Case 1. There is a crossing sequence in the first half that uses k cells. The claim is that all crossing sequences in the second half must be distinct.

Suppose that it is not the case. Then, we can remove the cells in between this crossing sequences. The machine behaves in exactly the same way. Since we assumed that x is minimum, we get a contradiction.

Case 2. The end of k is the second half. Then all the crossing sequences in the first half are distinct. The proof is exactly the same as in Case 1. We get as a conclusion that x uses less than k cells, a fact that contradicts our assumption.

The conclusion of this part is that M gives $n/2$ different crossing sequences.

If M accepts input x , $|cs_i(x)| \leq c^{S(n)}$.

different crossing sequences when M accepts $x \leq (c^{S(n)})^{c^{S(n)}} \geq n/2$.

Applying log in both sides twice, we get

$$c^{S(n)S(n)\log(c)=\Omega(\log n)} \Leftrightarrow S(n)(\log c) + \log(S(n)) = \Omega(\log \log n). \quad \square$$

How to halt a space-bounded machine?

Let $S(n) \geq \log n$. Let M be a $S(n)$ bounded Turing machine. Let the configuration be the state Q , position of input head, workspace and position of work head.

How many configurations are there?

configurations = $|Q| \cdot n \cdot |\Gamma|^{S(n)} \cdot S(n) \leq c^{S(n)}$ for a fixed constant c , where n stands for the position of the input head, and $S(n)$ for the position of the work head.

The idea is to count up to $c^{S(n)}$ and then reject. For example, we can keep a counter in c -ary, and check if the counter has used more cells than the cells used by the working tape.

We define $S(n)$ as *space constructable* if there is a $O(S(n))$ space-bounded Turing machine that on input 1^n halts after using exactly $1^{S(n)}$ cells of the working tape.

2 Space Hierarchy Theorem

Let $DSPACE(S(n))$ be the class of languages computed by an $O(S(n))$ space bounded Turing machine.

Theorem 4 *Let $S_2(n) \geq \log n$ be space-constructable. If $S_1(n) = o(S_2(n))$, then*

$$DSPACE(S_1(n)) \subsetneq DSPACE(S_2(n)).$$

Proof Construct M , with $\Sigma = \{0, 1\}$ such that on input x :

1. Mark out $S_2(n)$ cells of worktape.
2. Run the following, using at most $S_2(n)$ cells, reject if violated.
 - (a) Let $x = 0^i w$. All inputs can be represented like this if we choose the right i (there may be several i for the same input, in that case we try all of them).
 - (b) If w is not an encoding of a Turing machine with a read-only input, halt and accept.
 - (c) Otherwise let M' be the machine encoded by w .
 - (d) Simulate M' on input x .
 - (e) If M' halts, then we accept iff M' rejects.
 - (f) If M' is looping, i.e., after $2^{S_2(n)}$ accept.

Suppose that M' is $S_1(n)$ space bounded and $L(M') = L(M)$. Run on input $x = 0^i w$, where w is the encoding of M' . By diagonalization, we know that $L(M') \neq L(M)$.

Since $S_1(n) = o(S_2(n))$, if M' loops then we know that there is some constant c s.t. if M' runs for $c^{S_1(n)}$ steps, and then it loops. Since $c^{S_1(n)} = o(2^{S_2(n)})$, we know that we can detect if M' loops (rejects) and then $L(M) \neq L(M')$.

The rest of the proof is the same as in the time hierarchy theorem, i.e., using diagonalization we know that $L(M) \neq L(M')$. □

3 Non-determinism

Replace $\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R\}$ by a relation $\delta : Q \times \Gamma^k \rightarrow 2^{Q \times \Gamma^k \times \{L, R\}}$.

At every step, the machine has a number of options, which should be constant. It accepts iff at least one of the possible computations accepts.

$\text{NTIME}(T(n))$ is the class of languages accepted by a multitape non-deterministic Turing machine in time $T(n)$.

$\text{NSPACE}(S(n))$ is the class of languages accepted by a non-deterministic Turing machine in space $S(n)$.

By definition:

$\text{DTIME}(T(n)) \subseteq \text{NTIME}(T(n))$, and

$\text{DSPACE}(T(n)) \subseteq \text{NSPACE}(T(n))$.

3.1 The reachability method.

Theorem 5 For $T(n) \geq n$, $S(n) \geq \log n$,

- a) $\text{NTIME}(T(n)) \subseteq \text{DSPACE}(T(n))$,
- b) $\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))})$.

Proof a) We can see the computation of a non-deterministic Turing machine as a tree of depth $T(n)$, or a *configuration graph*.

- Every configuration is encoded by a string of length $O(T(n))$.
- Deterministically, do depth first search in the computation tree, and accept if an accepting configuration is reached.
- Each path, i.e., choice in the tree, can be described by a string of length $O(T(n))$.
- Only keep the configuration of the node of the tree which is being visited.
- Erase each configuration after generating the next one. To follow another path, start from the root and recompute the choices made. Keeping track of a path is $O(T(n))$.
- Space usage: $O(T(n))$.
- The tree is assumed to be finite, i.e., the machine is assumed to terminate.
- b) We use the configuration graph G :
 - Nodes in G are all the configurations.
 - There are directed edges between configurations iff the machine can move between them in one step.
 - Assume that $S(n)$ is space - constructible. Then, G can be generated in time $2^{O(S(n))}$.
 - We need to check if there is a path from the initial configuration to an accepting configuration doing graph traversing (for ex. using depth first search). This takes linear time in the size of G .
 - Total time usage: $2^{O(S(n))}$.

To get rid of the space-constructibility assumption, we run the algorithm for $S = 0, 1, 2, \dots$ until it works, i.e., until the machine actually halts. The time required for this algorithm is:

$$\sum_{i=0}^{S(n)} c^i = \frac{c^{S(n)+1} - 1}{c - 1} = 2^{O(S(n))}.$$

□

Now we can construct a hierarchy of classes:

$$\begin{aligned} L &= \text{DSPACE}(\log n) \stackrel{S}{\subseteq} \text{NL} = \text{NSPACE}(\log n) \stackrel{b}{\subseteq} P = \text{DTIME}(n^{O(1)}) \stackrel{S}{\subseteq} \\ \text{NP} &= \text{NTIME}(n^{O(1)}) \stackrel{a}{\subseteq} \text{PSPACE} = \text{DSPACE}(n^{O(1)}) \stackrel{b}{\subseteq} \text{EXP} = \text{DTIME}(2^{n^{O(1)}}) \stackrel{S}{\subseteq} \\ \text{NEXP} &= \text{NTIME}(2^{n^{O(1)}}) \stackrel{a}{\subseteq} \text{EXPSPACE} = \text{DSPACE}(2^{n^{O(1)}}). \end{aligned}$$

S - By syntax,

a - By Theorem 5, a),

b - By Theorem 5, b).

Theorem 6 (*Savitch's Theorem.*) *Let $S(n) \geq \log n$. Then*

$$\text{NSPACE}(S(n)) \subseteq \text{DSPACE}(S(n)^2).$$

Proof We want to find a path in the configuration graph of M from the initial configuration to an accepting configuration.

- Assume that $S(n)$ is space constructible.

- Encode the configuration by strings of length $O(S(n))$.

- Define $\text{reach}(u, v, i) = \text{Yes}$ iff there is a path from u to v in at most 2^i steps.

- We want to compute $\text{reach}(s, t, c \cdot S(n))$:

$\text{reach}(u, v, i) = \text{Yes}$ iff $\exists w$ s.t. $\text{reach}(u, w, i-1) = \text{Yes}$ and $\text{reach}(w, v, i-1) = \text{Yes}$.

This implementation of reach by a recursive algorithm has the idea of reusing space. We get $O(S(n))$ levels of recursion. In each, we need to store 3 configurations (u , v and w) and the depth i . This takes space $O(S(n))$.

The space used is then: $S(n)$ levels of recursion $\times S(n)$ space = $O(S(n)^2)$.

Again, we can get rid of the assumption of $S(n)$ space-constructibility, by trying all $S = 0, 1, \dots$ until the machine halts. \square