E0 224 Computational Complexity Theory **Fall 2014**

Lecture 9: October 03

Lecturer: Chandan Saha *Scribe: Datta Krupa R*

9.1 Abstract

1. Study space requirement of Turing machine
2. Study space complexity classes
3. Hierarchy between classes
4. Savitch's Theorem

9.2 Space complexity or space bounded computation

Definition 9.1 (class DSPACE(S(n)) (or SPACE(S(n))) *Let $S\colon \mathbb{N} \mapsto \mathbb{N}$ a language $L \subseteq \{0,1\}^*$ is in SPACE (S(n)), if there is a TM M that decides L using no more than $S(n)$ cells of the work tapes (excluding input tape)*

Definition 9.2 (class NSPACE) *Let $S\colon \mathbb{N} \mapsto \mathbb{N}$ a language $L \subseteq \{0,1\}^*$ is in NSPACE (S(n)), if there is a Non deterministic TM M that decides L using no more than $S(n)$ cells of the work tapes (excluding input tape)*

Here S(n) is space constructible function

Definition 9.3 (Space constructible function) *a function f is space-constructible if there exists a Turing machine M which, given a string 1^n consisting of n ones, outputs the binary (or unary) representation of $f(n)$, while using only $O(f(n))$ space*

Space constructible function examples $\log n, n, n^3, 2^n$

Not space constructible function examples $\log \log n, f(n) = C(\text{a constant function})$

Definition 9.4 (class PSPACE)

$$PSPACE = \bigcup_{c>=1} SPACE(n^c)$$

*Problems which are **solvable** by a deterministic turing machine using polynomial cells on work tape.*

Definition 9.5 (class NPSPACE)

$$NPSPACE = \bigcup_{c>=1} NSPACE(n^c)$$

Language decided by non deterministic turing machines using polynomial cells on work tape.

Examples of problems in PSPACE

3SAT problem

To show: To show: a TM which decides 3SAT in linear space.
If k is number of variables in 3SAT problem, generate all possible 2^k values and check if any of them satisfies. Set up a counter which counts from 0 to 2^k, this needs only O(k) space.

Circuit evaluation

$$cireval = \{(\phi, a) : \phi(a) = 1\}$$

Its a P complete problem, when ϕ is 3CNF then cireval can be solved in log space

Log space computation

$$L = SPACE(\log n)$$

$$NL = NSPACE(\log n)$$

Examples of problems in NL

$PATH = \{\langle G, s, t\rangle$; there is a path from node s to node t in directed graph G$\}$

To show : A NDTM with space $\log n$ which verifies PATH in log time
Have a pointer which starts at start node s, and TM guesses next node, verify that next node is reachable from current node, if not stop TM and output NO. if reachable move pointer to current node, continue above till we reach ending node t(stop and output YES), or adjacency condition fails. In either case TM halts and outputs an answer.

Hilbert's Nullstellensatz

$$f_1...f_n \in Q[x_1...x_n]$$

$f_i's$ are polynomials in variables $x_1...x_n$ with rational co-efficient.

Task: check if $f_1...f_n$ have common solution over complex numbers. This can be solved in PSPACE

9.3 Configuration graph of a TM – $G_{M,x}$

Definition 9.6 *Configuration graph of a TM – $G_{M,x}$ Configuration graph of a TM M, $G_{M,x}$ is a directed graph whose node corresponds to all possible configurations of M where input contains value x.*
Configuration is a snapshot of M during its execution.
Configuration graph depends both on input and machine M.

Configuration graph vertex contains

1. *Non-blank contents of working tapes*
2. *current state*
3. *current head position*

Configuration vertex of an $S(n)$ space bounded machine can be defined using $O(S(n))$ bits
Configuration graph contains a start vertex/node denoted by C_{start}, and a fixed unique accepting node, C_{accept}.
There is an edge from node C_i to C_j if M can go from config C_i to C_j by application of one transition function.

NOTE *To make C_{accept} unique erase all working tapes once we reach C_{accept}*

If M is deterministic every C_i has at most one outgoing edge
If M is non-deterministic every C_i has at most two outgoing edge

Observation *M accepts x if and only if there is a path from C_{start} to C_{accept} in $G_{M,x}$*

Theorem 9.7 (Theorem)

$$DTIME(S(n)) \subseteq SPACE(S(n)) \subseteq NSPACE(S(n)) \subseteq DTIME(2^{O(S(n))})$$

Proof:

$$DTIME(S(n)) \subseteq SPACE(S(n)) \subseteq NSPACE(S(n))$$

a *since a deterministic TM, run time is bounded by T(n) where as,a machine using S(n) space can run for $2^{\Omega(S(n))}$, as space can be reused.*

$$NSPACE(S(n)) \subseteq DTIME(2^{O(S(n))})$$

To show: problems solvable by non deterministic time machine using S(n) space can be solvable by deterministic time machine in $2^{O(S(n))}$ time. Possible number of nodes in configuration graph of $NSPACE(S(n))$ machine is $2^{O(S(n))}$, we can use deterministic STCON(st-connectivity) algorithm to find is there a path from C_{start} to C_{accept}.whose running time $2^{O(S(n))}$. ■

9.4 Savitch's Theorem

Theorem 9.8 (Savitch's Theorem) *Let $S(n)$ be a space constructible function and $S(n) \geq \log n$, then $NSPACE(S(n)) \subseteq DSPACE((S(n))^2)$*

Observation $\log n$ *implies machine should at least maintain a pointer to symbol being read from input tape*

To prove: *problems solvable by* $NSPACE(S(n))$ *machine can be solvable by a deterministic machine upper bounded by space* $O(S(n))^2)$
We show this by giving a path from C_{start} *to* C_{accept} *using* $O(S(n))^2)$ *cells on working tape.*

Proof: *Let* $L \in NSPACE(S(n))$
There is a NTM M that decides L using $O(S(n))$ *space, Number of configuration's is* $|G_{M,x}| \leq 2^{c*S(n)}$, *where c is constant depending on machine M but independent of input.*

Lets construct a deterministic TM that decides L.Given configuration graph if we can find a path C_{start} *to* C_{accept} *it would mean that* $L \in N$

Task: *On input x, we wish to check if there is a path from* C_{start} *to* C_{accept} *in* $G_{M,x}$

We define a recursive procedure Reach(C_1,C_2,*i) which, outputs is there a path of length i, between given vertices. It checks if there is a path of length i/2 between* C_1 *and intermediate vertex v and path of length i/2 between v and* C_2.

Is there an edge between to configuration's can be checked in $O(1)$ *time. We just need to see only few bits in adjacent configurations(as computation is local).*

```
1: procedure REACH(C1,C2,i)
2:     if i == 0 then return C1 == C2
3:     else if i == 1 then
4:         return true if there is an edge b/w C1 and C2
5:     else
6:         for all vertex v do
7:             if Reach(C1, v, i / 2) and Reach(v,C2, i / 2) then
8:                 return true
9:     return false
```

Let $N = 2^{O(S(n))}$ *be number of nodes*
i gets halved in each recursion call, i.e recursion depth will be $\log i = \log N$
In recursion stack space required is to store formal and local variables(C_1,C_2,*i,v) will be* $O(\log N)$.
total space requirement will be $O((\log N)^2)$.
i.e $O((\log(2^{O(S(n))})^2) = O((S(n))^2)$ ■

References

[AB09] *Sanjeev Arora and Boaz Barak, 2009.* Computational Complexity: A Modern Approach, *Cambridge University Press.*

[Savitch's Theorem] `http://en.wikipedia.org/wiki/Savitch's_theorem`

[Scribe Template] `http://www-inst.eecs.berkeley.edu/~cs294-8/sp03/Materials/scribe.tex`