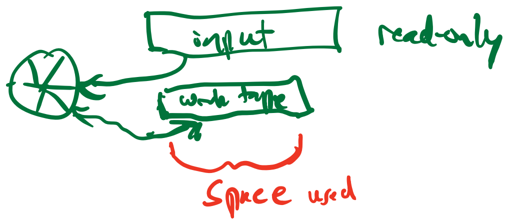
Lecture 21

CSE 431
Intro to Theory of Computation

So far: Space Complexity

Space-bounded TM model



$\text{SPACE}(S(n)) = \{A : A$ is decided by a TM as above using space $O(S(n))$.$\}$

$\text{NSPACE}(S(n)) = \{A : A$ is decided by an NTM as above using space $O(S(n))\}$.

e.g. Regular Languages $= \text{SPACE}(1)$

Note: In practice, space complexity is often as important as time complexity

Thm (a) $\left.\begin{array}{l}\text{TIME}(T(n))\\ \text{NTIME}(T(n))\end{array}\right\} \subseteq \text{SPACE}(T(n))$

(b) $S(n) \geq \log_2 n \Rightarrow \text{SPACE}(S(n)) \subseteq \text{TIME}(2^{O(S(n))})$

(c) $S(n) \geq \log_2 n \Rightarrow \text{TIME}(2^{O(S(n))})$

Proof (a) $\text{TIME}(T(n)) \subseteq \text{SPACE}(T(n))$ since a TM using time $T(n)$ only can touch $T(n)$ cells

$\text{NTIME}(T(n)) \subseteq \text{SPACE}(T(n))$ can try all paths in tree using only $O(T(n))$ space.

(b) Like algorithm for A_{LBA}.
\# of configurations on an input of length n before a repeat is –

$$\le n \cdot 2^{d S(n)}$$

position of input head

\# states × work tape contents × work-tape head position

Since $S(n) \ge \log_2 n$ this is $2^{O(S(n))}$.
Run space bounded TM using timer and reject if it runs too long.
Total space $O(S(n))$ simulation + $O(S(n))$ bits for timer
$2^{O(S(n))}$ time & $O(S(n))$ space simultaneously ⊠

(c) Obvious algorithm would be to try the "tree of paths" of length $2^{O(S(n))}$
Each path could only run this long but there would be too many leaves to check: exponential in $2^{O(S(n))}$!

Observation: Configurations on these other paths could be repeats of ones on the first path
Still only $2^{O(S(n))}$ total different configs

For each space-bounded TM M and input x, we define a directed graph on configurations $G_{M,x}$

Def $G_{M,x}$: each vertex is a configuration of M on input x (i.e. content of 1^{st} tape is x)

$2^{O(s(n))}$ vertices for $s(n) \geq \log_2 n$

start configuration: $C_0 = (q_0 x, \underset{\uparrow}{\sqcup})$

tape 1 tape 2

edge $C \to D$ iff $C \vdash_M D$ "yields in one step"

out-degree $\leq b$

$C \to D_1, D_2, \ldots, D_b$

for b some constant depending on δ function of M

without loss of generality, there is a unique accepting configuration

$$C_{accept} = (q_{acc} x, \underset{\uparrow}{\sqcup})$$

(simply have M clean up everything before accepting).

Note: • M accepts $x \iff \exists$ a path from C_0 to C_{accept} in $G_{M,x}$ (of length $2^{O(s(n))}$)

• M deterministic $\Rightarrow G_{M,x}$ has outdegree 1

Now: to prove (C) : Build $G_{M,x}$ and apply graph search for a path from C_0 to C_{accept} Time $(2^{O(s(n))})^2$ which is $2^{O(s(n))}$ ∎

Note: The algorithm for (c) uses $2^{O(S(n))}$ space along with $2^{O(S(n))}$ time.

Can we do better for simulating NSPACE(S(n)) machines?

Thm [Savitch] $S(n) \geq \log_2 n \Rightarrow$
$$NSPACE(S(n)) \subseteq SPACE(S^2(n))$$

Proof idea Use search for path from C_0 to C_{accept} in $G_{M,x}$ but don't write down the whole graph.

We know that if there is such a path then it has at most
$$T \leq 2^{dS(n)}$$ steps for some d.

Let's pretend we know T:

$$\overbrace{C_0 \xrightarrow{\leq T/2} C_{mid} \xrightarrow{\leq T/2} C_T}^{\leq T}$$

Then some C_{mid} as above exists.

Define function $CANYIELD_t(C,D) = \begin{cases} \text{true} & \text{if } C \vdash^*_M D \text{ using } \leq t \text{ steps} \\ \text{false} & \text{otherwise} \end{cases}$

configurations not writing x since it is the same for all nodes in $G_{M,x}$

Observe that we have the following recursive properties

$\text{CANYIELD}_0(C,D)$ iff $C=D$

$\text{CANYIELD}_1(C,D)$ iff $C \to D$, i.e.: $C \vdash_M D$ (or $C=D$)

Algorithm: can check using δ function of M

$\text{CANYIELD}_t(C,D)$ iff $\exists\, C_{mid}$ (config on input x) s.t. $\text{CANYIELD}_{t/2}(C, C_{mid}) \wedge \text{CANYIELD}_{t/2}(C_{mid}, D)$

Algorithm: Try all possible C_{mid} and make recursive calls

space for first call reused for second call

Goal: Compute $\text{CANYIELD}_T(C_0, C_{accept})$

Space used for recursive algorithm

# of levels : $\log_2 T$ which is $O(S(n))$

each level of call stack: C, D, T so $O(S(n))$

C, D: $O(S(n))$; T: # of bits $O(S(n))$

Other space used at each call level
$O(S(n))$ for C_{mid}

total $O(S^2(n))$

To do this we assumed that we knew T
But we don't actually need that

We modify the above to try all possible $T = 2^{d \cdot i}$

using $S = 1, 2, \ldots$ memory cells

Run above CANYIELD alg with above

Keep track of whether TM actually ever tries a rightward move when on last cell of work tape

If $\text{CANYIELD}_T(C_0, C_{accept})$ is true then accept

if no path found but a rightward move tried, then increase S, T

if no path found and no rightward move then reject

Total: $O(S^2(n))$ space & correct

Note: time used is worse than $2^{O(S(n))}$!

It is $2^{O(S^2(n))}$ but we only focus on space $\square$

Recall: $PSPACE = \bigcup_k SPACE(n^k)$

$NPSPACE = \bigcup_k NSPACE(n^k)$

Cor $NPSPACE = PSPACE$

Pf $NSPACE(n^k) \subseteq SPACE(n^{2k})$ $\square$

Now $P \subseteq NP \subseteq PSPACE \subseteq EXP$

$P \neq EXP$ (proof later) but all other containments conjectured to be $\subsetneq$ (open)

Is $P \stackrel{?}{=} PSPACE$? ($P \neq PSPACE$ implied by $P \neq NP$ and potentially easier to prove) open

Def^n B is PSPACE-hard iff $\forall A \in PSPACE$, $A \leq_m^P B$.

Def^n B is PSPACE-complete iff
- $B \in PSPACE$
- B is PSPACE-hard

Let φ be a Boolean formula in vars $x_1, \ldots, x_n$

$\langle \varphi \rangle \in SAT \iff \exists x_1 \ldots \exists x_n \, \varphi(x_1, \ldots, x_n)$ is true

$\langle \varphi \rangle \in TAUT \iff \underbrace{\forall x_1 \ldots \forall x_n \, \varphi(x_1, \ldots, x_n) \text{ is true}}_{\text{fully quantified Boolean formula}}$

Def^n $TQBF = \{\langle \Phi \rangle : \Phi$ is a fully quantified Boolean formula that is true$\}$

quantifiers may alternate

Next time we prove:

Thm TQBF is PSPACE-complete.