The classes L and NL

Huan Long

Shanghai Jiao Tong University

Acknowledgements

Part of the slides comes from a similar course given by Prof. Yijia Chen.

http://basics.sjtu.edu.cn/~chen/
http://basics.sjtu.edu.cn/~chen/teaching/TOC/

Textbook

Introduction to the theory of computation
Michael Sipser, MIT
Third edition, 2012

Outline

The classes L and NL

NL-Completeness

NL=coNL

The classes L and NL

Until now, we have considered only time and space complexity bounds that are at least linear — that is, bounds where $f(n)$ is at least n.

Now we examine smaller *sublinear* space bounds.

Not enough space to store the input. To consider this situation meaningfully, we need to modify the computational model.

Machine model for NL

A Turing machine with two tapes:

1. a read-only input tape;
2. a read/write work tape.

On the input tape, the input head can detect symbols, but not change them. The input head must remain on the portion of the tape containing the input. (Like a CD-ROM to a PC)

The work tape may be read and written in the usual way. (Like the main memory to a PC)

Only the cells scanned on the work tape contribute to the space complexity of this type of Turing machine.

For sublinear space bounds, we use only the two-tape model.

Definition

L is the class of languages that are decidable on logarithmic space on a deterministic Turing machine. In other words,

$$\text{L=SPACE}(\log n).$$

NL is the class of languages that are decidable in logarithmic space on a nondeterministic Turing machine. In other words,

$$\text{NL=NSPACE}(\log n).$$

$A = \{0^k 1^k \mid k \geq 0\} \in L$

Obviously, $A \in$ SPACE(n). To make it sublinear,

The machine counts the number of 0s and separately, the number of 1s in *binary* on the work tape.

The only space required is that used to record the two counters. Hence the algorithm runs in $\mathcal{O}(\log n)$.

PATH $\in$ NL

PATH
$=\{\langle G, s, t\rangle \mid G$ is a directed graph that has a directed path from s to $t\}$

The NTM: starting at node s, nondeterministically guessing the nodes of a path from s to t. In detail

- The machine only records the position of the current node at at each step on the work tape, *not the entire path* (which would exceed the logarithmic space requirement),
- The machine nondeterministically selects the next node from among those pointed by the current node,
- Repeat this action until it reaches node t and *accepts*.
  Or, until it has gone on for m steps and *rejects*, where m is the number of nodes in the graph.

It is not clear whether PATH$\in$ L holds.

Space vs. Time

The result that any $f(n)$ space bounded Turing machine also runs in time $2^{\mathcal{O}(f(n))}$ is no longer true for very small space bounds.

For example, a TM that use $\mathcal{O}(1)$ space may run for n steps.

Space vs. Time

To obtain a bound on the running time that applies for every space bound $f(n)$, we give the following definition.

Definition

If M is a Turing machine that has a separate read-only input tape and w is an input, a **configuration of M on w** is a setting of the state, the work tape, and the positions of the two tape heads. The input w is not a part of the configuration of M on w.

If M runs in $f(n)$ space and w is an input of length n, the number of configurations of M on w is $n2^{\mathcal{O}(f(n))}$.

Now, when $f(n) \geq \log n$, we still have that the time complexity of a machine is at most exponential in its space complexity. Savitch's theorem can be also extended to the sublinear space case provided that $f(n) \geq \log n$.

NL-Completeness

$$\mathsf{L} \stackrel{?}{=} \mathsf{NL}$$

Highly unlikely!

$\log$ space reduction

We define an **NL-complete** language: the one who is in NL and to which any other language in NL is reducible.

We don't use polynomial time reducibility, because

- all problems in NL except $\emptyset$ and Σ^* are polynomial time reducible to one another.

i.e., polynomial time reducibility is too strong to differentiate problems in NL from one another.

Instead, we use a new type of reducibility called $\log$ *space reducibility*.

$\log$ space reduction

Definition

A $\log$ *space transducer* is a TM with a read-only input tape, a write-only output tape, and a read/write work tape. The head on the output tape cannot move leftward, so it cannot read what it has written. The work tape may contain $\mathcal{O}(\log n)$ symbols.

A $\log$ space transducer M computes a function $f : \Sigma^* \to \Sigma^*$, where $f(w)$ is the string remaining on the output tape after M halts when it is started with w on its input tape. We call f a $\log$ *space computable function*.

Language A is $\log$ *space reducible* to language B, written $A \leq_L B$, if A is mapping reducible to B by means of a $\log$ space computable function f.

NL-complete

Definition

A language B is **NL-complete** if

1. $B \in$NL, and
2. every $A \in$ NL is $\log$ space reducible to B.

NL-complete

Theorem

If $A \leq_L B$ and $B \in L$, then $A \in L$.

Proof.

$f(w)$ may be too large to fit within the $\log$ space bound!

Suppose the $\log$ space reduction function is f, and B is decided by a TM M_B ∈L. We build a TM M_A for A:

- M_A computes individual symbols of $f(w)$ as required by M_B,
- M_A keeps track of where M_B's input head would be on $f(w)$,
- Every time M_B moves, M_A restarts the computation of f on w from the beginning and ignores all the output except for the desired location of $f(w)$.

Only a single symbol of $f(w)$ needs to be stored at any point, in effect trading time for space.

Corollary

If any NL-complete language is in L, then L=NL.

NL-complete problem

Theorem

PATH is NL-complete.

Proof(1)

For any language A in NL, say NTM M decides A in $\mathcal{O}(\log n)$ space. Given an input w we construct $\langle G, s, t \rangle$ in $\log$ space, where G is a directed graph that

G contains a path from s to t iff M accepts w.

1. Nodes of G are the configurations of M on w;
2. For configurations c_1 and c_2 of M on w, the pair (c_1, c_2) is an edge of G if c_2 is one of the possible next configurations of M starting from c_1;
3. Node s is the start configuration of M on w;
4. M is modified to have a unique accepting configuration, which is node t.

Proof(2)

The above reduction operates in $\log$ space: there is a $\log$ space transducer T that outputs $\langle G, s, t\rangle$ on input w

1. List the nodes of G
   Each node is a configuration of M on w and can be represented in $c \log n$ space for some constant c.
   T sequentially goes through all possible strings of length $c \log n$ and tests whether each is a legal configuration of M on w, and output those that pass the test.
2. List the edges of G
   T tries all pairs (c_1, c_2) , tests whether each is a legal configuration of M on w. Those that do are added to the output tape.

Corollary

NL $\subseteq$ *P*.

Proof.

1. $A \in$ NL then $A \leq_L$ *PATH*;
2. As any TM uses space $f(n)$ runs in time $n2^{\mathcal{O}(f(n))}$, a log space reducer also runs in polynomial time;
3. 1. and 2. imply A is polynomial time reducible to *PATH*;
4. *PATH*$\in$P.

$\square$

NL=coNL

Theorem (Immerman-Szelepcsényi Theorem 1988,1987)

NL=coNL.

Proof

$\overline{\text{PATH}}$ =$\{\langle G, s, t\rangle$ | There is no path from s to t in $G\}$.

Suppose G has m nodes in all (represented by $[m]$). Let c be the number of nodes in G that are reachable from s. Consider the input $\langle G, s, t, c\rangle$ first.

The machine M works as following:

- Initialize $\theta = 0$, for every node $u \in [m]$:
  1. M nondeterministically guess if u is reachable from s.
     - 1.1 if $u = t$ and the guess is YES, reject.
     - 1.2 if $u \neq t$ and the guess is YES, verify the guess: Guessing a path of length at most m from s to u.
       - $i.$ If the verifying passes: $\theta + +$;
       - $ii.$ If the verifying fails: reject.
- If $\theta = c$, accept; otherwise, reject.

Proof (to get c)

$A_i \quad (0 \le i \le m)$ is defined as the collection of nodes that are at a distance of i or less from s.
Then $A_0 = \{s\}$, $A_i \subseteq A_{i+1}$.
Let $c_i = |A_i|$, then $c = c_m$.

Obviously $c_0 = 1$, we will calculate c_{i+1} from c_i.

- Initialize $c_{i+1} = 1$.
  For every node v, repeat: $c_i' = 0$,
  1. for every node u in G, guess whether $u \in A_i$
     1.1 If YES, verify the guess:
     Guessing the path of length at most i from s to u.
     - If the verifying passes, $c_i' + +$;
       - Test if $(u, v) \in G$:
         If YES, $c_{i+1} + +$ and return (try another v);
         otherwise return; (try another u)
     - otherwise return. (try another u)
  2. If $c_i' \neq c_i$, reject. (start another branch of 1.)

  (try another v)
- Output c_{i+1}.

Proof (the final algorithm)

Here is an algorithm for 'no *PATH*'. Let m be the number of nodes of G.

$M =$ "On input $\langle G, s, t\rangle$:

1. Let $c_0 = 1$. $[\![\, A_0 = \{s\}$ has 1 node $]\!]$
2. For $i = 0$ to $m - 1$: $[\![$ compute c_{i+1} from c_i $]\!]$
3. Let $c_{i+1} = 1$. $[\![\, c_{i+1}$ counts nodes in A_{i+1} $]\!]$
4. For each node $v \neq s$ in G: $[\![$ check if $v \in A_{i+1}$ $]\!]$
5.   Let $d = 0$. $[\![\, d$ re-counts A_i $]\!]$
6.   For each node u in G: $[\![$ check if $u \in A_i$ $]\!]$
7.   Nondeterministically either perform or skip these steps:
8.   Nondeterministically follow a path of length at most i from s and *reject* if it doesn't end at u.
9.   Increment d. $[\![$ verified that $u \in A_i$ $]\!]$
10.   If (u, v) is an edge of G, increment c_{i+1} and go to stage 5 with the next v. $[\![$ verified that $v \in A_{i+1}$ $]\!]$
11.   If $d \neq c_i$, then *reject*. $[\![$ check whether found all A_i $]\!]$
12. Let $d = 0$. $[\![\, c_m$ now known; d re-counts A_m $]\!]$
13. For each node u in G: $[\![$ check if $u \in A_m$ $]\!]$
14. Nondeterministically either perform or skip these steps:
15.   Nondeterministically follow a path of length at most m from s and *reject* if it doesn't end at u.
16.   If $u = t$, then *reject*. $[\![$ found path from s to t $]\!]$
17.   Increment d. $[\![$ verified that $u \in A_m$ $]\!]$
18. If $d \neq c_m$, then *reject*. $[\![$ check whether found all of A_m $]\!]$
Otherwise, *accept*."

‘

Complexity classes so far

$$\mathsf{L} \subseteq \mathsf{NL} = \mathsf{coNL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE}.$$