18.404 Recitation 10

Nov 13, 2020

Today's Topics

- Review: Space Hierarchy
- Review: Time Hierarchy
- Prove: STRONGLY-CONNECTED is NL-Complete
- Rewording: NL=coNL
- $A \leq_L B$ and $B \in L$ implies $A \in L$

Review: Space Hierarchy

Review: $f(n) \in o(g(n))$ *means that*: $f(n) / g(n) \to 0$

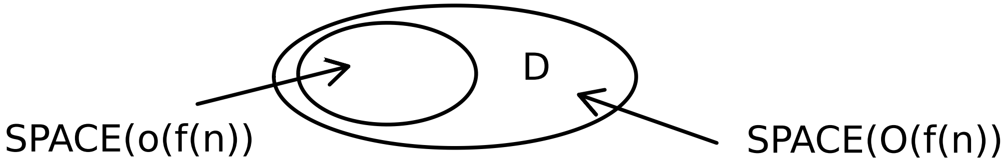
Goal: $\mathrm{SPACE}(o(f(n))) \subsetneq \mathrm{SPACE}(O(f(n)))$

Idea: Show that a language A exists that is decidable in $O(f(n))$ space but not in $o(f(n))$ space. This is done using a diagonalization.

Review: Space Hierarchy (cont.)

Come up with a diagonalization TM D such that:

1. D runs in O(f(n)) space
2. D ensures that its language is distinct from all L(M) where TM M runs in o(f(n)) space



D = “On input w

1. Mark off f(n) tape cells where n = |w|. If use more tape, *reject*
2. If w does not contain a TM description for M, *reject*
3. Simulate M on w (on the rest of w)
    a. *Accept* if simulation rejects
    b. *Reject* if simulation accepts”

Review: Space Hierarchy (cont.)

D = "On input w

1. Mark off f(n) tape cells where n = |w|. If use more tape, *reject*
2. If w does not contain a TM description for M, *reject*
3. Simulate M on w
   a. *Accept* if simulation rejects
   b. *Reject* if simulation accepts"

Issues:

1. What if M loops?
   a. Stop M if it runs for $2^{f(n)}$ steps. Need to include a counter, adds f(n) space is OK
2. How to compute f?
   a. Need to assume f is space-constructible. ie) can compute f in O(f(n)) space. Most functions such as log(n), n, n^2, 2^n are space-constructible

Note: not space-constructible is log(log(n))

Review: Time Hierarchy

Goal: TIME(o(f(n) / log(f(n)))) $\subsetneq$ TIME(O(f(n))

Same Idea: Show that a language A exists that is decidable in O(f(n)) time but not in o(f(n) / log(f(n))) time. This is done using a diagonalization.

Review: Time Hierarchy (cont.)

Come up with a diagonalization TM D such that:

1. D runs in O(f(n)) time
2. D ensures that its language is distinct from all L(M) where TM M runs in o(f(n) / log(f(n))) time

D = “On input w

1. Compute f(n)
2. If w ≠ <M>10* for some TM M, *reject*
3. Simulate M on v for f(n) / log(f(n)) steps
    a. *Accept* if M rejects
    b. *Reject* if M accepts or **has not halted yet**”

Review: Time Hierarchy (cont.)

Log factor comes from "Simulate M on w for f(n) / log(f(n)) steps"

In order to keep track of the counter (of size log(f(n))), need to carry it around with the head as added baggage.

a b c c b a a

TIME(o(f(n))) != TIME(O(f(n) * log(f(n)))

The act of moving this counter around costs O(log(f(n)) time per step.

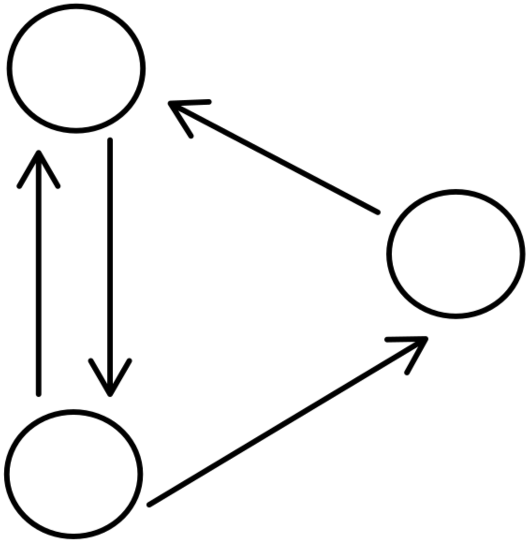
Therefore, a TM can only simulate a TM that is O(log(f(n)) smaller than it.

Prove: STRONGLY-CONNECTED is NL-Complete

Definition: STRONGLY-CONNECTED

A <u>directed</u> graph where a path exists between every pairing of nodes

Example:



STRONGLY-CONNECTED is NL-Complete (cont.)

1. STRONGLY-CONNECTED $\in$ NL
2. PATH $\leq_L$ STRONGLY-CONNECTED

Proving STRONGLY-CONNECTED $\in$ NL is easier via:

NOT-STRONGLY-CONNECTED $\in$ NL

meaning

STRONGLY-CONNECTED $\in$ coNL = NL

STRONGLY-CONNECTED is NL-Complete (cont.)

Show: NOT-STRONGLY-CONNECTED ∈ NL

Ideas?

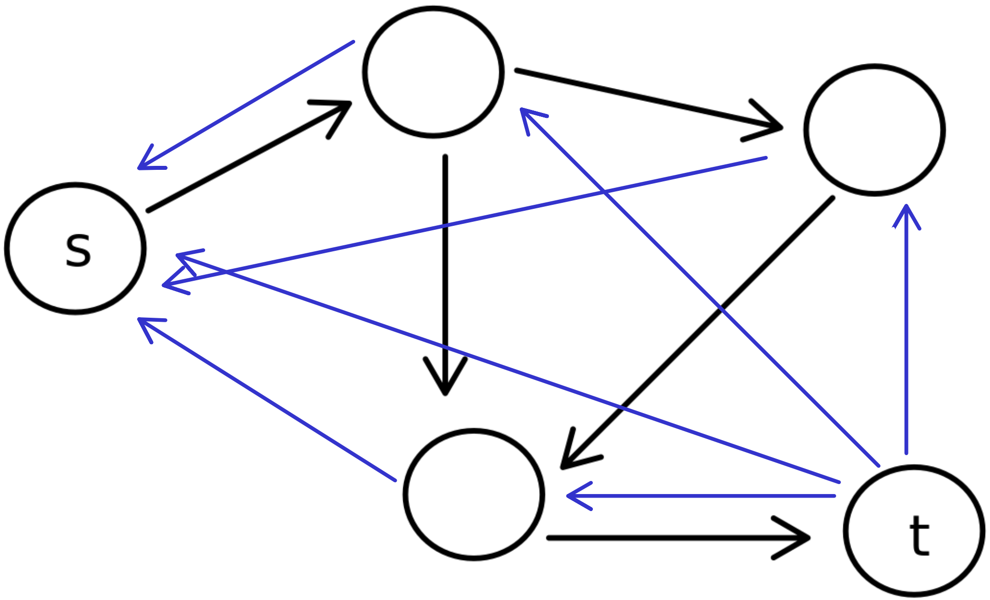
NOT-STRONGLY-CONNECTED = "On input G:
1. nondet. guess two vertices u, v
2. Return NOT-PATH(G, u, v)"

Since PATH in NL, NOT-PATH in coNL=NL. So can use that in proving NOT-STRONGLY-CONNECTED in NL.

STRONGLY-CONNECTED is NL-Complete (cont.)

Show: PATH $\leq_L$ STRONGLY-CONNECTED

PATH -> STRONGLY-CONNECTED
NOT-PATH -> NOT-STRONGLY-CONNECTED



Rewording: NL=coNL

Need to prove:

NOT-PATH ∈ NL

Since NOT-PATH is coNL-Complete (since PATH is NL-Complete)

Two parts:

1. An NL TM can calculate the number of nodes *c* reachable from s in k steps
2. With that knowledge try to guess all *c* nodes where k = m and if desired node t is not one of them, then NOT-PATH(G,s,t) accepts

Rewording: NL=coNL

Prove NOT-PATH ∈ NL

Need an NL algorithm that determines if <G,s,t> has no path from s to t

Two parts:

1. An NL TM can calculate the number of nodes *c* reachable from s in k steps
2. With that knowledge try to guess all *c* nodes where k = m and if desired node t is not one of them, then NOT-PATH(G,s,t) accepts

Rewording: NL=coNL

First: Assume an NL TM can calculate the number of nodes ***c*** reachable from s in k steps

Have an NL TM:

- Go through all m nodes in G, guessing if a node u is reachable from s within k steps
- If so, increment a *reachable* counter
- Also if the guessed u = t, then record a Flag that t was seen
- Once all m nodes have been iterated
  - Check to see if the *reachable* counter = c, *reject* if not
  - If t was seen via the Flag being set, *reject*
  - Otherwise accept

Rewording: NL=coNL

Show: An NL TM can calculate the number of nodes ***c*** reachable from s in k steps

Let c_i be the number of nodes reachable from s within i steps

We know that $c_0 = 1$, s itself reachable within 0 steps

Show strategy to compute c_{i+1} from c_i.

(This is a recursive induction proof)

Rewording: NL=coNL

Show: An NL TM can calculate the number of nodes ***c*** reachable from s in k steps

Let A_i be the nodes reachable from s within i steps, $A_0 = \{s\}$

A NL TM:

1. Go through all of the m nodes. Guess if v belongs to A_{i+1}
   a. Guesses if u belongs to A_i.
      i. Verifies if such path exists from s within i steps. If so, increment a *inner* counter
      ii. If (u,v) is an edge, set a Flag that v is in A_{i+1}
   b. If *inner counter* = c_i, means that A_i was correctly guessed
      i. If so and if Flag is set, increment *outer* counter
2. Return *outer* counter which is c_{i+1}

$A \leq_L B$ and $B \in L$ implies $A \in L$

Simple Lemma:

Same also holds for NL as well.