L = SPACE(log n) Any language decidable by a determininstic Turing machine using O(log n) workspace

NL = NSPACE(log n) Any language decidable by a nondeterministic Turing machine using O(log n) worspace

We use a 2-tape Turing machine. A read only input tape. The head of the input tape cannot leave the input cells. A work tape that is read/write but only O(log n) cells.

{akbk} is in L, {w wR} is in L

Savich’s Theorem. NL is a subset of SPACE(log2 n). It is not known if L = NL.

The proof was we create configurations of the Turing machine, and we ask CANYIELD(c1, c2, t). CANYIELD(ci,c2t) = CANYIELD(c1,x,t/2) and CANYIELD(x,c2, t/2).

We end up with a call stack size equal to the size of a configuration.

The size of a configuration was the size of the tape + 2.

What about for L? The trick is we hard-code the input w into CANYIELD. So we say we have **configurations of M on w.** The configuation is the contents of the work tape, the location head on the work tape, the state of M, and the location of the head on the input tape, represented in binary.

Reductions in logspace. A <=L B.

The reduction uses a 3-tape TM.

We have a read only input tape. The input head cannot leave the cells containing the input.

We have a O(log n) size work tape that is read/write.

We have an output tape that is write only. We can’t inspect the output tape to see what has been written.

Theorem: If A <=L B and B is in L then A is in L.

Recall: If A <=P B and B is in P then A is in P. This had a pretty trivial. We can solve an input to B in polynomial time. Take input to A, make an input to B (in polynomial time, and the input to B is polynomial size). When we solve B, the runtime is a polynomial of a polynomial of n.

For logspace, it is a little more tricky. Suppose we can decide B in logspace. We are given an x input to A. We need to convert it to f(x), the input to B. But f(x) can be much larger than log size. We can’t store it when we run B.

Solution: Use f(x) as an “oracle”. We are going to query f for the kth bit of f(x). Given x (input to A), we run the solver for B. Whenever B needs its next input value, we pause B, run f

on input x until we get that bit of f(x) that B needs, then run B again. How much space do we need? Log space for B, we need to store what bit/cell B’s solver needs (log space), we need to simulate f on x (log space) until we get that cell/bit.

A problem B is complete for NL if

a) B is in NL

b) for every language A in NL, A <=L B.

PATH = {<G, a, b> | G is a directed graph and there exists a path from a to b in G}

PATH is NL-complete.

a) Show that PATH is in NL.

Place a on the worktape. (represented in binary, log n bits)

Guess the next vertex to visit in the path to b. Check with the input that there is an edge from a to this vertex. Erase a, guess the next vertex of the path and check that that edge exists. Repeat. The machine stores only 2 vertices at a time on the work tape. Accept if b is ever guessed and there is a valid edge to b.

Keep a counter on the worktape that counts the number of vertices in our path. If the counter ever exceeds the number of vertices in G, reject.

b) Show that for every A in NL, A <=L PATH. We are given A, we have machine M that decides A, we need to, in log space, create <G, a, b>. Each vertex will be a configuration of the nondeterministic Turing Machine that decides A. (A configuration is the contents of the worktape, the location of the input head, the work head, and state.) Each edge will indicate that it is possible to go from one configuration to another in one step of the machine for A. a be the initial configuration, b be the only accepting configuration.

First, enumerate all possible configurations for A in lexicographical order to the ouput tape (the vertices of G). Enumerate all pairs of possible configurations (x,y), then simulate A on x for one step and see if y is one of the possible results of one step of A. If it is, add (x,y) to the output tape. (the edges of G).

L is a subset of P. The workspace is only O(log n) cells. If d is the number of symbols of the tape, the machine can only run for d^(O(log n)) steps without out looping. This is a polynomial in n. So the machine runs in polynomial time.

PATH is in P. Since the logspace reduction is polynomial time, we have that NL is a subset of P.

co-NP = the set of languages L where L is in NP. For example, 3-SAT (the set of 3-SAT formulas with no solutions plus all strings that are not 3-SAT formulas) is in co-NP.

co-NL = the set of languages L where L is in NL.

We do not know of NP = coNP, and we suspect not.

But, we have a theorem: co-NL = L. We will prove it by showing PATH is in NL.