CSC2401F Introduction To Computational Complexity
Lecture 19 / Thursday, November 15, 2001
Instructor: Allan Borodin
Scribe: Peter Kanareitsev

1 Review

1.1 Two important inclusions

In the last lecture, we said that

$$NSPACE(s(n)) \subseteq DEPTH(O(s(n)^2))$$

with a notion of uniform complexity on the right-hand side providing for the family of circuits to be generated by a DTM. Note that there is a big-O expression on the right-hand side because there is no size or depth compression result for the boolean circuit model of computation.

We have also seen that

$$uniform - DEPTH(D(n)) \subseteq DSPACE(D(n))$$

These connections between TM and circuit complexity are evidence of Goldschlager's *parallel computation thesis*, which states that space and parallel time are equivalent up to polynomial factors, given any reasonable notions of space and parallel time. (Only the notions that satisy this statement can be considered reasonable.) The parallel computation thesis also equates efficient parallel computation with polylogarthmic time = polylogarithmic depth. A somewhat more restrictive version of this thesis will follow when we introduce the NC^k classes.

1.2 Savitch's theorem revisited

Note that Savitch's Theorem

$$NSPACE(s(n)) \subseteq DSPACE(s(n)^2) \quad \text{for space-constructible } s(n) \geq \log n$$

is a corollary of the two inclusions above if $DEPTH$ is made uniform.

2 Proof of the inclusions

2.1 Proof that $NSPACE(s(n)) \subseteq DEPTH(O(s(n)^2))$ in the special case $s(n) = \log n$

Consider, for simplicity's sake, $STCON$, which is a prototypical language in $NL = NSPACE(\log n)$ and let us verify that it can be decided by circuits of depth $O((\log n)^2)$.

Recall that $STCON$ of graphs G with a fixed number of vertices n can be expressed in terms of the *path* predicate as

$$
\begin{aligned}
STCON(u,v) = &\quad \{(u,v) \mid path(u,v,n-1)\} \quad \text{where} \\
path(u,v,l) = &\quad \exists \text{ path } u \to v \text{ of length } \le l \\
= &\quad \exists\, w \text{ such that } (path(u,w,\lfloor \tfrac{l}{2} \rfloor) \wedge path(w,v,\lceil \tfrac{l}{2} \rceil)
\end{aligned}
$$

We can represent G as an adjacency matrix (or any other desired encoding over binary alphabet) and build a circuit which on input (G,u,v) computes the *path* predicate recursively.

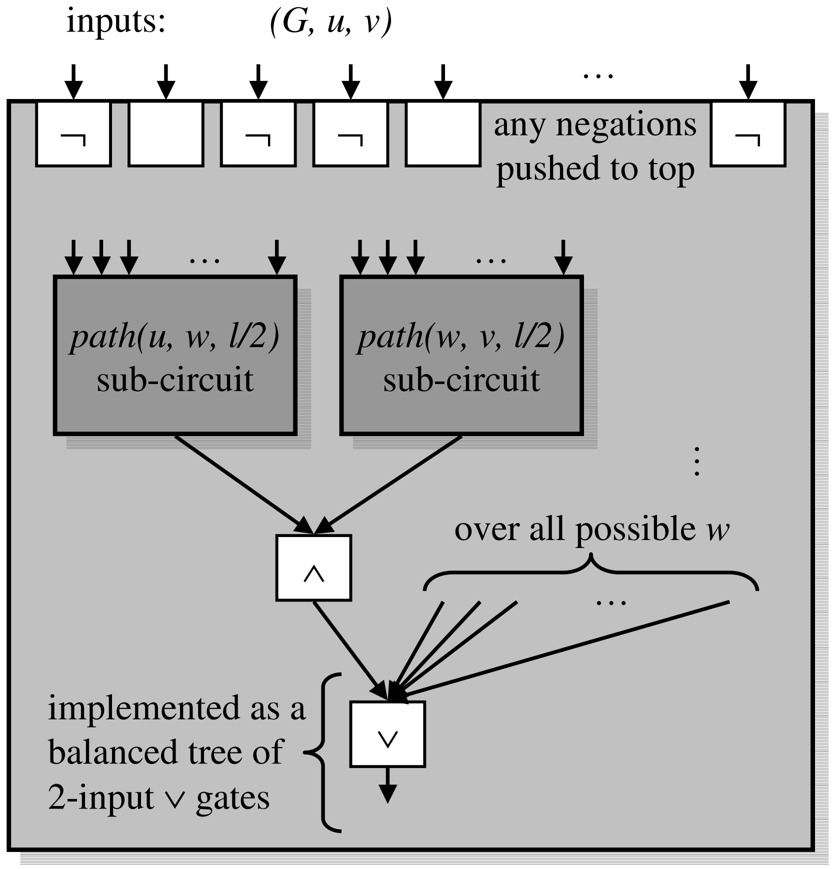

Figure 1: Circuit for $path(u,v,l)$ (accepts $STCON$ when $l=n-1$).

Next, we ask what is the circuit depth required to check a path of length l. Reading the circuit from top to bottom, we see that it is roughly

(depth required to check a path of length $\frac{l}{2}$) $+ 1 + \log n$

The $O((\log n)^2)$ bound is clear from this recurrence relation when we put $l = n-1$. ■

This argument can be generalized to other $NSPACE$ classes. Note also that had we allowed unbounded-fanin $\vee$-gates, we would obtain $DEPTH(O(\log n))$ for $STCON$ and $DEPTH(O(s(n)))$ in general.

2.2 Sketch of proof that $DEPTH(D(n)) \subseteq DSPACE(D(n))$

To start with, whatever notion of uniform $DEPTH$ we adopt, there must be DTM's able to generate the circuit family, or answer local questions about the circuit family, in $LOGSPACE$. To recognize the language accepted by a given circuit family, we would build a DTM that (on input x) does a depth-first search starting with the bottom (output gate) of the circuit (with $|x|$ inputs) by asking local questions about this circuit.

The procedure looks for an accepting subtree recursively at each gate. At an $\vee$-gate, it needs only one accepting parent gate, while at an $\wedge$-gate it needs all parent gates to accept.

A typical depth-first search would use a stack of gates, resulting in using $D(n)^2$ space. The trick that can be used to avoid this is to stack only the relative addresses of gates. ■

3 NC hierarchy

3.1 Definition of NC

Consider a family of circuit complexity classes

$$\begin{aligned} NC^k = \{L \mid\ & \exists\, c_1, c_2, l, \text{ and a circuit family } C_n \text{ computing } L_n \text{ such that} \\ & depth(C_n) \leq c_1 (\log n)^k \ \wedge\ size(C_n) \leq c_2 n^l\} \end{aligned}$$

NC stands for "Nick's class", named this way by Cook, after Nick Pippenger who was visiting U. of T. at the time he suggested these classes. Pippenger later tried to popularize "Steve's class" SC, however it never caught on. Pippenger's purpose in defining the NC family was to formulate parallelly-efficient computing. The parallel computation thesis allows computations that require (say) n^4 processors, which is not very practical. But as a mathematical class, there is some inhenernt stability in these definitions. A very restrictive parallel computation thesis might equate efficient parallel computation with say quasi-linear size and logarithmic depth.

Note that for NC^1, the restriction on $size$ in the above definition is redundant since the fanin of each gate is bounded by 2.

Also, let us define $NC = \bigcup_{k \geq 0} NC^k$

3.2 Hierarchy

With correct notions of uniformity, we have the following hierarchy:

$$NC^1 \subseteq DLOG \subseteq NLOG \subseteq NC^2 \subseteq NC^3 \subseteq NC \subseteq P \subseteq NP \subseteq \Sigma_2 \subseteq \Sigma_3 \subseteq PSPACE$$

None of the above inclusions has been proven to be proper, although, of course, we know that (say) $NC \neq PSPACE$.

A canonical example of a language $L \in NC^2 - NLOG$ (where the $\notin NLOG$ part is conjectured) is matrix determinant. As a function, the determinant (say of an integer matrix) can be computed in NC^2.

3.3 AC classes and some examples

Earlier in this lecture, we observed that allowing unbounded-fanin gates in a circuit would enable us to avoid squaring the depth, so that we could say $NSPACE(s(n)) \subseteq DEPTH(O(s(n)))$. This motivates the following definition (A stands for "alternating"):

$$AC^k = \{L \mid \ \exists\, c_1, c_2, l, \text{ and a circuit family } C_n, \text{ with unbounded fan-in OR and AND gates, computing } L_n, \text{ such} \\ depth(C_n) \leq c_1(\log n)^k \ \wedge \ size(C_n) \leq c_2 n^l\}$$

Here size should be defined as the number of edges in the DAG defining the circuit. These new classes fit into the above hierarchy as follows:

$$AC^0 \subseteq NC^1 \subseteq DLOG \subseteq NLOG \subseteq AC^1 \subseteq NC^2 \subseteq \ldots$$

Simple boolean functions like disjunction (of any number of variables) are AC^0 functions. The parity (XOR) function is *provably* in $NC^1 - AC^0$.

NC^1 was at some point considered to be a good candidate for the class of "practically computable" functions.

Next term, an interesting course will be taught by Steve Cook, which will explore the relationship between complexity classes and proof systems.

We now present a single model of computation that will produce all the classes mentioned in the above hierarchy in an easy and natural way.

4 Alternating Turing Machines

4.1 Definition

It is possible to develop many of the essential concepts of complexity theory (i.e. all of the complexity classes mentioned above) witihin one uniform model of computation, namnely Alternating Turing Machines (ATM's). Martin Tompa presented a complexity course using ATM's as the underlying model. There is now a link to his lecture notes on the course website.

We will define an ATM by generalizing a NTM. Consider a one-tape NTM and ignore its output tape. It can be pictured as

where finite control is captured by state transition *relation*

$$\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma \times \underset{\substack{\uparrow \\ \text{(worktape)}}}{\{L, R, N\}} \times \underset{\substack{\uparrow \\ \text{(input tape)}}}{\{L, R, N\}}$$

where for each non-halting state, there are some (finitely many) moves, each of which can be done next. We call such states *existential states*. To define ATM's we also allow *universal states*, wherein all of the finitely many configurations prescribed by the transition relation must be verified. With ATM's we have a notion of *accepting computation subtree*, which is finite-branching (branching only in universal states). This is analogous to acceptance by a $\vee$-$\wedge$ circuit.

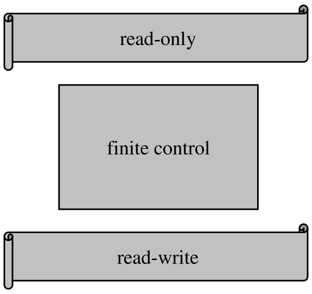

Figure 2: Simplified picture of a NTM.

We can also define the notions of time and space complexity for ATM's. For example, time is the longest path in an accepting computation subtree:

$$\begin{aligned} time_M(x) = &\ \min_{\text{accepting computation subtrees}} (\text{max path length of subtree on input } x) \\ time_M(n) = &\ \max_{|x|=n} time_M(x) \end{aligned}$$

Similarly,

$$\begin{aligned} space_M(x) = &\ \min_{\text{accepting computation subtrees}} (\text{max space of any configuration in subtree}) \\ space_M(n) = &\ \max_{|x|=n} space_M(x) \end{aligned}$$

We can also count the number of alternations between existential and universal states:

$$alt_M(x) = \ \min_{\text{accepting computation subtrees}} (\text{number of alternations along any path in subtree})$$

These definitions give rise to complexity classes $ATIME(t(n))$, $ASPACE(s(n))$, as well as the simultaneous resource bounded classes such as the time and space classes $ATIMESPACE(t(n), s(n))$.

4.2

The similarity between ATM's and $\vee$-$\wedge$ circuits suggests that alternating time is equivalent to parallel time, however the definition of ATM's given so far does not allow examination of input bits in parallel, as is possible with circuits, so any nontrivial computation on an ATM must take at least time n. We would like to remove this restriction and we do so by endowing an ATM with an index tape, which allows $\log n$-time access to the input bits (writing to index tape effectively performs a seek to the address written, on the input tape).

With this modification, we can say that $ATIME(t(n)) \sim$ parallel time $t(n)$ (for $t(n) \geq \log n$). In particular, $ATIME(t(n)) =$ "uniform" NC^1, which is yet another characterization of NC^1.

It is clear how ATM's give rise to AC and NC classes. In the next lecture we will see how they give rise to the Σ classes of the polynomial hierarchy.

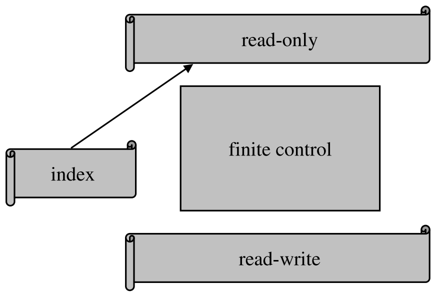

Figure 3: ATM with an index tape.