CS538, Spring 1998
Script for Lecture on 2/27/98
by Takahiro Murata

Characterization of NC^k

This theorem shows that the two "definitions" of NC^k, one with the resource bound on an alternating Turing machine and the other using a class of circuits, are in fact equivalent.

Theorem For $k \geq 1$:

$$NC^k = ASPACE(\log n)TIME(\log^k n)$$

where NC^k denotes the class of languages accepted by DLOGTIME-uniform circuit families of fan-in 2 $\wedge$, $\vee$, and $\neg$ gates with polynomial size and $O(\log^k n)$ depth.

Proof For $\subseteq$, let A be accepted by family C_n with the specifics as above. Note that, given a string $path \in \{R, L\}^*$ and the name g of a gate in C_n, assuming a $\neg$ gate takes its input from its left child and has a dummy gate as its right child, we can identify a unique gate, if any, which is "reachable" by "traversing" the circuit towards the input starting at g and following $path$. When h is the gate identified in such a way, henceforth we say h is reachable from g via $path$. Consider the following machine M:

on input x
  compute $|x|(= n)$; // time $O(\log |x|)$
  $\exists$ guess the name g of the output gate of C_n;
  $\forall$ do the following:
    (1) check that g is the output gate of C_n;
    (2) call $Evaluate(g, \epsilon, 0)$;

where $Evaluate(g, path, b)$ is given in Figure 1.

We can observe that the amount of space used by M is $O(\log n)$ since the names of the gate for g and h as well as i for the i-th bit of input x are all sized $\log n$. The amount of time spent by M should be bounded by the following:

$$\begin{aligned} \leq\ & \frac{\text{depth of the circuit}}{\log n}(\text{time to evaluate the "}|path| = \log n\text{" case}) \\ & + (\text{depth of the circuit})(\text{time to evaluate the other case } \textit{except} \text{ at the input level}) \\ & + (\text{time at the input level}) \end{aligned}$$

Further, note the following:

- the depth of the circuit is $O(\log^k n)$,
- the time to evaluate the "$|path| = \log n$" case is $O(\log n)$ mainly due to writing down the guess of a gate name,
- the time to evaluate the other case except at the input level is bounded by a constant,
- and, the time at the input level is also $O(\log n)$.

In the above, regarding the evaluation time at each level of recursive call to $Evaluate()$, we only considered each path that leads to another recursive call. Everywhere the computation path bottoms out without encountering 0, 1, 'input' or 'negated input', checking if a guessed

type or name of the gate is correct, the time is also bounded by $O(\log n)$, which leads to no extra amount of time required to bound the total computation. Combining these together, we conclude that the running time of M is $O(\log^k n)$.

> Remark 1: if we had no conditional branch on "$|path| < \log n$" case in the program of *Evaluate*, although the time would be bounded by the depth of the circuit, i.e., $O(\log^k n)$, the space would become too large because the $|path|$ may also be about the depth of the circuit.

Now for $\supseteq$, let A be accepted by an $ASPACE(\log n)TIME(\log^k n)$ machine. We can assume, as usual, without loss of generality that the ATM reads its input only at halting states. Let the gates be configurations of the ATM. Then, the size of our circuit is $n^{O(1)}$. Also, the depth of the circuit is proportional to the time bound of the ATM which is $O(\log^k n)$. In order to see DLOGTIME-uniformity, we observe that there is a deterministic machine to recognize the following language in linear time:

$$\{(n, g, h, p) : |p| \leq \log n \text{ and } p \in \{L, R\}^* \text{ and, } h \text{ is reachable from } g \text{ via path } p \text{ in } C_n\}$$

since all this machine has to do is to write down the configuration represented by gate g, follow path p, and check if the resultant configuration is the one represented by gate h. The other two criteria for the uniformity can be similarly argued. □

> Remark 2: NC^1 is important because:
>
> - Circuits in this class characterize the complexity of Boolean formula evaluation and Regular sets. That is, all regular sets are in NC^1; and some regular set is complete for NC^1. Similarly, Boolean formula evaluation is complete for NC^1.
> - It coincides with the class of the languages recognized by bounded-width branching programs with polynomial size as shown in the following lecture, which is fundamental to the algebraic theory of circuit complexity.

```
Evaluate(g, path, b)
  // path ∈ {R, L}*, and b ∈ {0, 1} to represent the parity of the number of ¬ gates
  // between the output gate and the gate reachable from g via path.
if |path| < log n then
  ∃ guess the type of the gate h reachable from g via path;
    // type ∈ {∧, ∨, ¬, 'input', 'negated input', 0, 1}
  ∀ do the following:
    (1) check that the type is correct;
    (2) if b = 0 then
          if type = ∨ then
            ∃ branch into two
              on one branch, call Evaluate(g, pathL, b)
              on the other branch, call Evaluate(g, pathR, b) ;
          if type = ∧ then
            ∀ branch into two
              on one branch, call Evaluate(g, pathL, b)
              on the other branch, call Evaluate(g, pathR, b) ;
          if type = ¬ then
            call Evaluate(g, pathL, 1 − b);
          if type = 'input' then
            ∃ guess the i such that h is x_i;
            ∀ check that i is correct & halt, and accept ⟺ x_i = 1;
          if type = 'negated input' then
            ∃ guess the i such that h is x̄_i;
            ∀ check that i is correct & halt, and accept ⟺ x_i = 0;
          if type = 0 then halt and reject;
          if type = 1 then halt and accept;
        if b = 1 then
          if type = ∨ then
            ∀ branch into two
              on one branch, call Evaluate(g, pathL, b)
              on the other branch, call Evaluate(g, pathR, b) ;
          if type = ∧ then
            ∃ branch into two
              on one branch, call Evaluate(g, pathL, b)
              on the other branch, call Evaluate(g, pathR, b) ;
          if type = ¬ then
            call Evaluate(g, pathL, 1 − b);
          if type = 'input' then
            ∃ guess the i such that h is x_i;
            ∀ check that i is correct & halt, and accept ⟺ x_i = 0;
          if type = 'negated input' then
            ∃ guess the i such that h is x̄_i;
            ∀ check that i is correct & halt, and accept ⟺ x_i = 1;
          if type = 1 then halt and reject;
          if type = 0 then halt and accept;
if |path| = log n then
  ∃ guess the name of the gate h reachable from g via path;
  ∀ do the following:
    (1) call Evaluate(h, ε, b);
    (2) check that h is the right gate;
```

Figure 1: Program for *Evaluate*