CAS705 Fall 2005 **Assignment 2** **Grzegorz Herman**

1. Let C_i be the i-th circuit in the given family. I assume that it uses only three types of gates: OR, AND and NEG. We modify the circuit according to the following procedure:

   (a) traverse the circuit from its output towards inputs, pushing any NEG gates "through" AND and OR gates using de'Morgan laws

   (b) traverse the circuit from its inputs towards the output - if the output of any AND (OR) gate is connected to the input of another AND (OR) gate, delete the connection and connect all of its inputs directly to the latter gate

   (c) delete gates with 0 fan-out (possibly created in this or previous step)

   (d) place all variables (with possible negations) at level 0

   (e) for $k = 1, 2, \ldots$ (until all original gates have been placed) do the following:

   - set TYPE to be OR if $2 \mid k$ and AND otherwise
   - take all TYPE gates which all inputs have been already placed and put them at level k, connecting them to appriopriate input gates or their "copies" (see below) at level $k-1$
   - for each gate at level $k-1$ which still has "unused" outputs, create a new, fan-in 1 TYPE gate at level k to copy its value

   None of the steps of the presented procedure change the semantics of the whole circuit. Thus the resulting circuit calculates the same function as the original one. Steps (a) and (e) guarantee that the circuit contains only AND and OR gates (except for the variable level) and that they are placed in alternating layers with connections between consecutive layers only.
   Let us denote by d the number of layers in the resulting circuit. Then the original circuit's output gate G must have been placed at level $d-1$. If all of G's inputs were placed at levels $d-3$ or lower, they would be put at levels $d-4$ or lower (because all that "survived" steps (b) and (c) have different type than G and thus must have been placed at level of the opposite parity), and then G could have been put at level $d-3$. Using the above argument inductively, we can conclude that a path of length at least $d-1$ must have existed in the original circuit. Therefore the height of the new circuit does not exceed the depth of the original by more than 1, and is $O(log^i(n))$.
   Let us now consider the size of the resulting circuit. The only step that introduces new gates is step (e). For each of the original gates, at most one new gate is created at any particular level. Thus the number of the gates in the resulting circuit is at most $O(n^t) * O(log^i(n)) = O(n^{t+1})$ and is still a polynomial of n.
   Therefore the new circuit satisfies all of the required conditions.

2. If the AND gate is connected (through the OR gates) to no more than e variables, then there are at most 3^e different (in terms of input literals) OR gates (for each variable x the OR gate either uses x, its negation, or none of them). Deleting "duplicate" input

connections we can lower the fan-in of the AND gate to be at most 3^e. The fan-in of the OR gates is, of course, bouded by e. Thus, using distributive laws to exchange the AND and OR levels, we will create a single OR gate connected to no more than e^{3^e} AND gates. The number of gates in the whole circuit increases at most e^{3^e} (which is a constant!) times, and thus stays polynomially bounded.
If we switch to "semantic" dependance, then we know that setting any values for all but e variables would result in the same output from our AND gate. Thus, we create a single copy of each OR gate considered, replacing all its inputs (except those from our e variables) by 0 or 1 (choosing arbitrarily, but consistently). Then our AND gate computes the same function as the original, and it depends "syntactically" on no more than e variables. For each AND gate in the circuit we created a copy of each OR gate it uses. If the size of the circuit was bounded by $p(n)$, then there were no more than $p(n)$ AND gates, and each of them "used" no more than $p(n)$ OR gates. Thus the new circuit has at most $p^2(n)$ gates, so it still remains polynomially bounded.

3. First, let us note that the i-th carry bit (C_i) can be easily expressed in terms of the input bits:

$$C_i = \bigvee_{0 \le j < i} \left((a_j \wedge b_j) \wedge \bigwedge_{j<k<i} (a_k \vee b_k) \right)$$

Let us now consider the circuit with inputs $a_0, \ldots, a_{n-1}$, $b_0, \ldots, b_{n-1}$ and the following levels:

- $O_i := a_i \vee b_i$ for $0 \le i < n$
  $O_n := 0$
  $A_i := a_i \wedge b_i$ for $0 \le i < n$
  $A_n := 0$
- $N_i := \neg A_i$ for $0 \le i \le n$
  $D_{i,j} := A_j \wedge O_{j+1} \wedge \ldots \wedge O_{i-1}$ for $0 \le j < i \le n$
- $C_0 := 0$
  $C_i := D_{i,0} \vee \ldots \vee D_{i,i-1}$ for $0 < i \le n$
- $P_i := O_i \wedge N_i$ for $0 \le i \le n$
  $Q_i := A_i \wedge C_i$ for $0 \le i \le n$
- $R_i := P_i \vee Q_i$ for $0 \le i \le n$

It is not hard to see that C_i calculates the i-th carry, and thus R_i calculates the i-th bit of the sum. The circuit has exactly 5 levels (regardless of n) and thus the whole family of circuits is in AC^0.

Now let us consider the problem of repeated addition (in which the n-th circuit computes the sum of n numbers, each having n bits). First, let us consider the following circuit (called S_n) with inputs $a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1}, c_0, \ldots, c_{n-1}$:

- $x_i := a_i$ xor b_i xor c_i for $0 \leq i < n$ (expressed in CNF)
  $x_n := 0$
- $y_0 := 0$
  $y_{i+1} := (a_i \wedge b_i) \vee (a_i \wedge c_i) \vee (b_i \wedge c_i)$ for $0 \leq i < n$

It is easy to see, that $a + b + c = x + y$ (x and y have $n + 1$ bits each). Therefore we can reduce the problem of adding 3 n-bit integers to adding 2 $(n + 1)$-bit integers in constant depth. Cascading the appriopriate circuits in a tree-like way we can reduce the original problem of adding n n-bit numbers to adding two numbers with no more than $O(n+\log(n)) = O(n)$ bits each. Then we can use the AC^0 circuit for addition and transform it into an NC^1 circuit by replacing all gates with fan-in k with trees of depth $O(\log(k))$. As the largets fan-in was equal to the number of the bits in the inputs, the total depth of our repeated addition circuit will still be $O(\log(n))$.

4. First of all, let us notice that if f_n is a symmetric Boolean function (on n variables), then the value of f depends only on the number of inputs that are set to 1. Let us now construct the circuit that calculates f. First, we treat all inputs as 1-bit numbers and construct a circuit that adds them, thus counting the number of 1's. As we know from the previous question, this can be done using an NC^1 circuit. The output of this circuit is a $\log(n)$ bit number. Thus if write the dependency between the number of 1's in the input and the result of f in the conjunctive normal form, we will need not more than $O(2^{\log(n)}) = O(n)$ gates to do this. The depth of our circuit will increase only by 2, so the result will still be an NC^1 circuit.