MTAT.07.004 — Complexity Theory	November 10th, 2011
Lecture 9	
Lecturer: Peeter Laud	*Scribe(s): Kristjan Krips*

1 The class NC

We will define a complexity class NC_j. The class is named after Nicholas John Pippenger.

Definition 1 A language L belongs to the class NC_j if it can be decided by a log-space uniform family of circuits with depth $O(\lambda n.log^j\, n)$. ♢

The definition says that the family of circuits has to be uniformly generated using logarithmic memory and therefore the circuit has to be of polynomial size. It is important that the circuit is of polylogarithmic depth, where depth is the longest path from input to output. As defined before, each node in the Boolean circuit has a maximum of two inputs.

The class NC is a union of classes NC_j, $\mathsf{NC} = \cup_{j\in\mathbb{N}}\mathsf{NC}_j$. The class is important because it is the model for efficient parallel computation. A problem is efficiently parallelizable if it can be computed with a polynomial number of processors in polylogarithmic time. Solving a polynomial time problem with parallelization should reduce the time complexity to polylogarithmic. Such algorithms are scalable, e.g. if the number of processors increases two times then the solving time reduces by a factor of 2.

Actually, the previous statement is not so clear compared to the statement that "P models efficient computation". When comparing problems in polynomial time and exponential time then it is intuitional to see the difference. However, if we compare a polylogarithmic problem to a small polynomial problem, e.g. $log^3 n$ to $\sqrt{n}$ then at first the logarithmic problem grows faster than the polynomial problem and the they will be about equal when $n = 10^9$. But this shows that in this case we would benefit from polylogarithmic complexity only if we would have more than 10^9 processors. Also, if we have large constants in the polylogarithmic problem then n would have to be large before polylogarithmic problem would be easier to solve than the problem in $\sqrt{n}$.

2 P-completeness

Definition 2 A problem in P is P-complete if every other problem in P is log-space reducible to it. ♢

The P-complete problems are the hardest to parallelize.

2.1 Reducibility in NC

Claim 1 *Graph reachability is in* NC_2. *Boolean matrix multiplication is in* NC_1 *and therefore transitive closure is in* NC_2.

Proof It is important to know that the transitive closure of a matrix is computable in class NC with a circuit of depth log^2. For multiplication we view a ring $(\{true, false\}, \vee, \&)$, where $\vee$ defines addition and & defines multiplication on the set $\{true, false\}$.

Now, if we have two square matrices then we use the standard formula for multiplication. For matrices A and B of size $n \times n$ the product elements $c_{j,k}$ can be found with the formula $c_{j,k} = \bigvee_{i=1}^{n} a_{j,i} \& b_{i,k}$. From the formula we see that we can parallelize computing the values of the product. Computing the disjunction $\bigvee_{i=1}^{n} a_{j,i} \& b_{i,k}$ in parallel takes $log\, n$ time. For parallelization n^3 processors are required as $\forall i, j, k$ we have to compute $a_{j,i} \,\&\, b_{i,k}$.

The transitive closure of a matrix A is $A^+ = A + A^2 + A^3 + \cdots$. More important is reflexive transitive closure if we want to know if two nodes are connected in a graph, $A^* = I + A + A^2 + A^3 + \cdots$. The reflexive transitive closure shows if two nodes are connected by a path of arbitrary length.
With Boolean matrices we can compute A^* so that first we will take matrix $I + A$, this will show if two nodes are connected with at most one step. Now we will take a square of it $(I + A)^2 = I \vee A \vee A \vee A^2 = I \vee A \vee A^2$ and this shows if two nodes are connected with at most two steps. The previous matrix can also be squared to get $(I + A)^4$, which shows if two nodes are connected in at most four steps. We can continue like that and finally we will get $(I + A)^{2^{log\, n}}$, which shows if two nodes are connected in at most n steps. Therefore, if multiplication can be done in logarithmic time then transitive closure can be computed in $\log^2$ time. Also, the corresponding circuit that does the computation has to be generated in logarithmic memory.

■

Theorem 2 *If* $L \leq_m^L L'$ *and* $L' \in \mathsf{NC}$ *then* $L \in \mathsf{NC}$.

The theorem says that L is reducible to L' if there exists a function f running in logarithmic memory that maps L to L'. Reducibility in logarithmic memory is closed in the class NC. Also, if $L \leq_m^L L'$ and $L' \in \mathsf{NC}_i$ then $L \in \mathsf{NC}_{max(i,2)}$.

Proof We have the Boolean circuit for L' and we need a Boolean circuit for L. The Boolean circuit C_L for L could be composed of two circuits C_f and $C_{L'}$, the first one would be for computing f and the second circuit would accept L'. Therefore, we can construct C_L so that we apply C_f to the input and $C_{L'}$to the output of C_f. As f runs in logarithmic memory then $C_f \in NC_2$ and thus C_f is with depth at most log^2 of the input. Therefore, the depth of C_L is the depth of C_f plus the depth of $C_{L'}$. Thus, $L \in \mathsf{NC}$. ■

From the theorem we see that if a P-complete problem belongs to NC, then $\mathsf{P} = \mathsf{NC}$. Then all efficient problems can be parallelized and therefore P-complete problems are the most difficult to parallelize.

3 Circuit Value Problem (CVP)

3.1 CVP and variations

Now we will view some P-complete problems.

The first problem: Given deterministic TM M, input x and number of steps 1^n. Does M accept x in at most n steps?
Every problem in P can be reduced to this problem. The reduction f from a language $L \in \mathsf{P}$ is the following. Given x, $f(x)$ is $\langle M, x, 1^{p(|x|)}\rangle$, where M is a polynomial-time DTM that recognizes L and p is a polynomial function that bounds the running time of M. Parallelizing the solution to this problem is akin to finding a parallelization technique for arbitrary programs (represented by the machine M).
The second problem: Given a Boolean circuit $\{0,1\}^n \to \{0,1\}^m$, a n-bit string and $i \in \{1,\ldots,m\}$. What is the i-th output of the circuit on that string? This is called the Circuit Value Problem (CVP).
This problem is in a way analogous to SAT. SAT is NP-complete while this problem is P-complete. CVP is solvable in linear time by evaluating all its nodes one by one. The reduction from the previous problem is similar to the Cook-Levin reduction showing the NP-completeness of SAT.
The third problem: Same as previous, but circuit may contain only AND- and OR-nodes. This is called the Monotone CVP — MCVP.

Claim 3 *It is possible to write down an arbitrary monotone function*[1] *from* $\{0,1\}^n \to \{0,1\}$ *by using AND- and OR-nodes.*

Proof Let there be a monotone function $f : \{0,1\}^n \to \{0,1\}$. Now when we insert arguments $x_1,\ldots,x_n$ then we can write

$$f(x_1,\ldots,x_n) = x_1 \& f(1,x_2,\ldots,x_n) \vee \neg x_1 \& f(0,x_2,\ldots,x_n).$$

Now we have a function of $n-1$ variables using AND/OR nodes and we could use this step for induction if we could get rid of negation. Because we have a monotone function then $f(0,x_2,\ldots,x_n) \leq f(1,x_2,\ldots,x_n)$. If $x_1 = false$ then the first half of the disjunction is false and the second half is $f(0,x_2,\ldots,x_n)$. If $x_1 = true$ then the first half of the disjunction is $f(1,x_2,\ldots,x_n)$ and the second half of disjunction is always a subset of the the first half because $\neg x_1 = false$. Therefore, we can remove $\neg x_1$ from the function and the proof can be completed by induction. ■

Reduction. We will show how to construct the reduction f for $CVP \to MCVP$. If we have a gate in the non-monotone circuit then for each wire in that gate we have two wires in the monotone circuit and these wires always carry values that are negations of each other. If we would have a wire that carries a value v in the non-monotone circuit then in the monotone circuit there are two wires, where on the first wire is the value v and on the

[1] false $\leq$ true. The order on tuples of booleans is defined componentwise

second wire is the value $\neg v$.
So, if we have an AND gate in the non-monotone circuit then in the monotone circuit there are two gates with the same inputs, AND-gate and OR-gate. This comes from the De Morgan's laws. If we have an OR gate in the non-monotone circuit then its the other way around. Finally, if we have a NOT-gate then the wires are swapped.

The fourth problem: Same as previous, but the fan-in and fan-out of each internal node is 2, inputs go to and outputs come from OR-gates, and AND-s and OR-s alternate on each path from input gate to output gate. Besides that the fan-out of input nodes is also 2. This is called AM2CVP.

3.2 MCVP $\leq_m^L$ AM2CVP

We will transform MCVP to AM2CVP and in the process we will have to add input nodes to the new circuit. There are many steps in this transformation and they have do be done sequentially. Doing these steps has to be in logarithmic memory and therefore each of these steps has to be done in logarithmic memory.

1. step. The inputs have to go to OR-nodes, therefore every initial input is doubled. I.e. two new input nodes with the value of the initial input are connected to the OR-gate. This can be done in logarithmic memory.

2. step. If there is an AND node then we will append an OR node to it with the OR node having the input from the AND node. This is actually an identity transformation.

3. step. If there is a node with more than two outputs, i.e. more than two nodes are reading the output, then the node will be expanded by adding OR-nodes so that each OR-node will have one input and two outputs. Therefore, a tree of OR-nodes is created.

4. step. If there are nodes $AND \to AND$ then an OR-node is added so that we would have $AND \to OR \to AND$. The same is done for OR-nodes, i.e. when there are nodes $OR \to OR$ then an AND-node is added so that we would have $OR \to AND \to OR$.

5. step. If there is an OR-node with single input then we will add an extra input node. The other input is an input node with value zero.

6. step. If there is an AND-node with single input then we will add an input from OR-node, that gets input from two new input nodes. One of these new nodes has to have a value of one.

7. step. We will have to get two outputs for all nodes. At the moment all input nodes have only one output. Now we will make two copies of the circuit, except the input nodes and after this each input nodes gets two outputs, one for each of the copies. After this step all other nodes have one or two outputs and if a node has one output then it has one output in both copies. Therefore we will add a new output node. In case of an AND-node with single input we will add an OR-node, which gets inputs from both copies of

the node. In the case of an OR-node we will have to add an AND-node first and after that two new output OR-nodes, which get the second input from a new input node with a value zero. Solving the single output problem for AND-,OR-nodes is shown on the Figure 1.

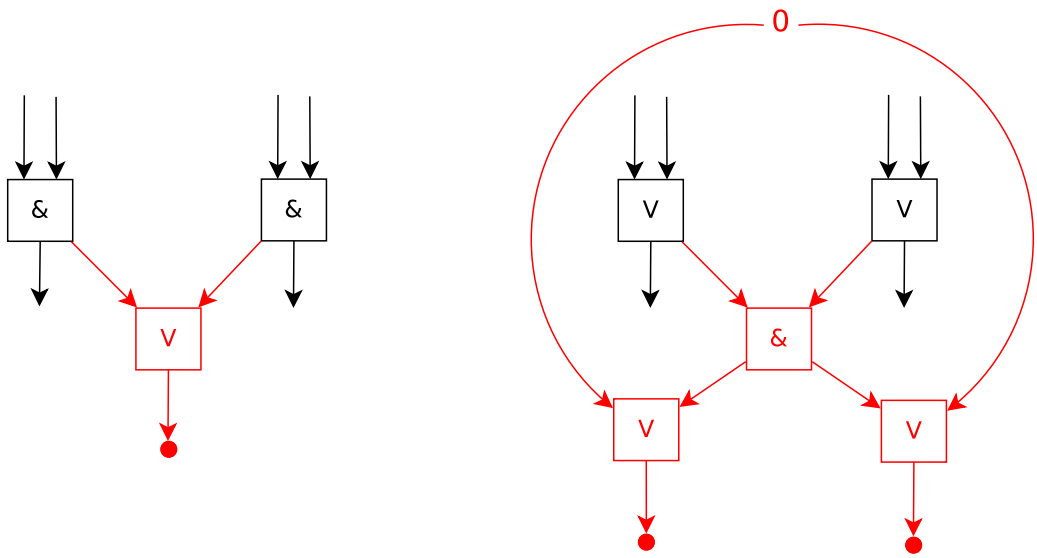

Figure 1: Adding the second output to AND-nodes and OR-nodes.

3.3 NANDCVP / NORCVP

NANDCVP is like CVP but all gates are NAND-s. NAND is an universal operation as all other operations can be computed using NAND-s. NORCVP is the same but all gates are NOR-s. As before, we want all inputs and internal nodes to have two outputs.

Theorem 4 $AM2CVP \leq_m^L NANDCVP$.

Proof The idea is to complement all inputs and turn all gates to NAND-s. ■

3.4 Depth-first-search (DFS)

Given a directed graph G, where the outputs of each node are ordered and nodes themselves are also ordered (think of the graph represented as an adjacency list). Also, two vertices u and v are given. Depth-first traversal of the graph gives an ordering of the nodes. Is u visited before v in the depth-first traversal of G? The next theorem shows that it is difficult to parallelize this problem.

In NORCVP all nodes contain the operation NOR and have two inputs and two outputs. We assume that when we are given NORCVP as a bitstring then the nodes are topologically sorted, i.e. for each node its inputs are before the node.

Theorem 5 $NORCVP \leq_m^L DFS$.

Proof Let C be a circuit consisting only of NOR-gates with fan-in and fan-out 2. Assume its nodes are topologically sorted, all our reductions provide or preserve this. We are given the circuit C and we will have to transform it into a graph. Let i be a node with inputs i_1, i_2 that are smaller than i and outputs j_1, j_2 that are larger than i. A fragment of the graph is displayed on the Figure 2. ■

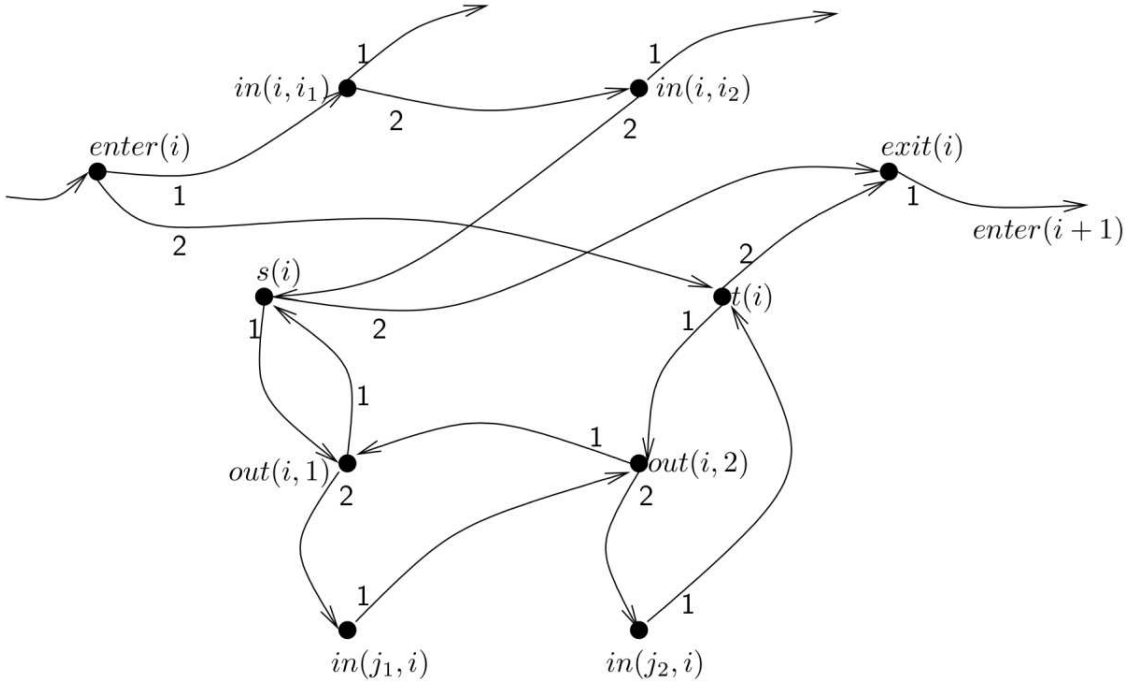

Figure 2: A fragment of the DFS graph.

Traversing the graph starts from the enter of the first node, goes to the exit of the first node, then goes to the enter of the second node and after that to the exit of the second node and so on. Now we give some lemmas for traversing the graph.

Lemma 6 *At the moment traversal is about to enter enter(i):*

- $s(i), t(i), out(i,1), out(i,2), exit(i), in(j_1,i), in(j_2,i)$ *are untraversed.*
- $in(i,i_k)$ *is traversed iff the node* i_k *evaluates true in the circuit.*
- *Nodes pointed to by the first outedges of* $in(i,i_k)$ *are traversed.*

Lemma 7 *If node i evaluates to true then* $s(i)$ *is traversed before* $t(i)$*. If node i evaluates to false then* $t(i)$ *is traversed before* $s(i)$*.*

Initialization

- There are nodes $in(k, i)$ for each input node i and each node k that receives input from i.
- All true inputs are chained together and traversed first.
- I.e. when we reach $enter(k)$, the node $in(k, i)$ has been traversed iff input i is true.
- Finally, we ask whether $s(output)$ is traversed before or after $t(output)$.

3.5 Alternating reachability (AGAP)

Given a directed graph G where each node is labeled with $\forall$ or $\exists$, and two nodes u, v. Does $apath(u, v)$ hold, where

- $apath(x, x)$ holds for all nodes x.
- If x is labeled with $\exists$, then $apath(x, y)$ if exists z, such that $x \to z$ and $apath(z, y)$.
- If x is labeled with $\forall$, then $apath(x, y)$ if for all z, where $x \to z$, we have $apath(z, y)$.

This is similar to alternating Turing machines. Besides that, actually the acronym AGAP stands for alternating graph accessibility problem.

Theorem 8 $MCVP \leq_m^L AGAP$.

Proof We are given circuit C, inputs x and output z. We want to know if the monotone circuit evaluates to true. We will introduce two constant nodes 0 and 1 and use these for the input instead of x. Now, let AND-nodes be labeled with $\forall$ and OR-nodes with $\exists$. Then all edges should be reversed in order to move from the end to the beginning, i.e. start moving from the output node z of the initial graph. We are interested if the input 1 reaches output or not. We know that the OR-node is passed if one of its inputs is one and passing the AND-node requires that both of its inputs are ones. That is why we label AND-nodes with $\forall$ and OR-nodes with $\exists$. ■

4 HORNSAT,UNIT,GEN,CFPARSE

4.1 HORNSAT

Recall that in HORNSAT there are variables and positive and negative literals. A Horn clause is $l_1 \vee l_2 \vee \cdots \vee l_n$, where at most one literal is positive. A Horn formula is a conjunction of Horn clauses. HORNSAT is the set of all satisfiable Horn formulas. HORNSAT is P-complete.

Theorem 9 $AGAP \leq_m^L HORNSAT$.

Proof We will try to construct function f that reduces AGAP to HORNSAT. We are given graph G and we need to write it down using Horn formulas. Besides graph G we are given nodes u, v and we will apply f to them, i.e. $f(G, u, v)$. We will introduce variables $x_i \equiv apath(i, v)$ that correspond to apath from i to v. Now we will try to write down the graph using Horn formulas.

We have a Horn clause x_v. If i is an existence node $\exists$ and from i we can get to node j, i.e. $i \to j$ then $x_j \Rightarrow x_i$. If i is a for all node $\forall$ and $i \to j_1, \ldots, i \to j_k$ then $x_{j_1} \& \ldots \& x_{j_k} \Rightarrow x_i$. We want to know if there is a path from u to v. For that we should write down a new Horn clause $\neg x_u$. If there is a path from u to v then the conjunction of all previous clauses and $\neg x_u$??? is not satisfiable. ■

4.2 Unit resolution (UNIT)

The input to unit resolution is the same as the input to 3-CNF-SAT. 3-CNF-SAT means that there is a conjunction of disjuncts and in every disjunct there are at most three literals. The resolution is one method to check if 3-CNF-SAT is satisfiable. For that we do operations on the disjuncts and if we finally get an empty disjunct then 3-CNF-SAT is not satisfiable and otherwise it is. This is because we consider an empty disjunct to be not satisfiable.

If $C = l_1 \vee \cdots \vee l_m$ is a disjunct and $l = \neg l_j$, then the unit resolution of C an l gives

$$l_1 \vee \cdots \vee l_{j-1} \vee l_{j+1} \vee \cdots \vee l_m.$$

The name unit resolution comes from the fact that the second disjunct $l = \neg l_j$ is a single literal. Given a set of disjuncts of size ≥ 3, is it possible to derive the empty disjunct from the given set using unit resolution?

Theorem 10 $CVP \leq_m^L UNIT$.

Proof The gates $v_i \equiv v_j \wedge v_k, v_i \equiv v_j \vee v_k$ and $v_i \equiv \neg v_j$ can all be represented as conjunctions of disjuncts of size at most 3. True inputs are represented as unit clauses, i.e. every input is unit disjunct. If we would start to do the unit resolution on them then the computation is similar to the way how CVP is evaluated. ■

4.3 Generability (GEN)

Given a set X, subset $S \subseteq X$, element $x \in X$ and a binary operation $\bullet : X \times X \to X$ (given as a table). If there are n elements in X then the size of the table is n^2. Can x be generated from S using $\bullet$?

Theorem 11 $UNIT \leq_m^L GEN$.

Proof A set of disjuncts D is given, i.e. we are given 3-CNF instance. Let X be the set of all subdisjunctions of D (with 2,1 and 0 literals), plus an extra element $\perp$. The extra element is required because it might not be possible to apply unit resolution to all elements and in the case that it is not possible the result is $\perp$. We start by setting $S = D$. Let $\bullet$ be unit resolution and let x be the empty disjunct. We start with the given disjuncts S and try to get the empty disjunct. ■

4.4 Context-free parsing (CFPARSE)

The complexity of context-free parsing varies largely, depending on the fact whether there are productions where on the right side are empty strings or not. If such productions do not exist then context-free parsing is parallelizable and if such productions exist then it is not. Context-free grammar contains non-terminals, starting symbol S that is a non-terminal, terminals and productions that map a non-terminal to a string over terminal and non-terminals. We start from S and after a production we get to a string. We can choose a non-terminal from it, choose a production where the non-terminal is on the left side and replace the non-terminal in the string with the right side of the production. The grammar generates a language which consists of all words made of terminals which can be derived using the productions.

Given a context-free grammar $G = (N, T, P, s)$ and a word w. Does $w \in L(G)$?

Theorem 12 $GEN \leq_m^L CFPARSE$.

Proof We are given the instance of the generability $(X, S, \bullet, x)$. From this we put together a context-free grammar. Let $N = X$, $T = \{a\}$, $s = x$. The set of terminals is not important to us, it could be an empty set as all the given elements are non-terminals. The starting symbol is the one where we want to get to and the generating/productions corresponds to the opposite to the operation, i.e. if the operation is $s \bullet t = r$ then the production will be $r \to st$. All elements that belong to the initial set S can be deleted — i.e. we have the productions $r \to \varepsilon$ for $r \in S$.

We ask whether the empty word is generated by this grammar. So if starting from x we can derive a word consisting of only elements of S then we can delete all elements of S and so we get an empty word.

■