MTAT.07.004 — Complexity Theory November 4th, 2011

Lecture 8

Lecturer: Peeter Laud *Scribe(s): Kristjan Krips*

Circuit complexity Non-uniform model of computation

1 Boolean circuits

A boolean circuit is a directed acyclic graph(DAG), where each internal node v contains a Boolean operation op(v) of small fixed arity. There are three types of nodes in the boolean circuit: input nodes, internal nodes and output nodes. Only the internal node have Boolean operations. E.g. the operation might be a conjunction, a disjunction, negation or a constant if the node always outputs the same value. The arity of the Boolean operation determines for the corresponding node how many incoming edges the node has. If the operation is not symmetric then it is important to distinguish different orderings of the incoming values. Therefore, the edges of the nodes are ordered. The input to the boolean circuit comes from the input nodes and the output of the circuits comes from the output nodes. Each output node has only a single output edge. But there might be several input and output nodes and therefore they have to be ordered so that it would be possible to distinguish them. An example of a boolean circuit is shown on the Figure 1.

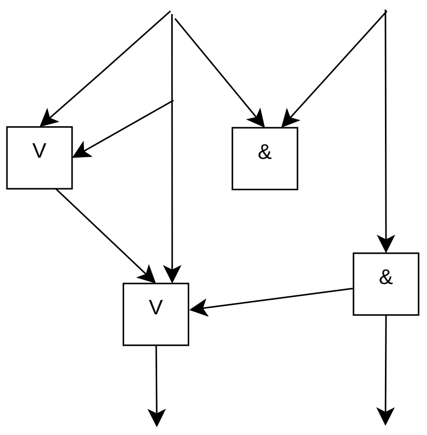

Figure 1: A boolean circuit with four Boolean operations.

Definition 1 A Boolean circuit C with n inputs and m outputs defines a function

$$[\![C]\!] : \{0,1\}^n \to \{0,l\}^m .$$

◊

This means that the Boolean circuit takes n-bit input and outputs m-bit result. Mostly we consider the cases with $m = 1$, i.e. for recognizing a language. Also, from this definition we see that each Boolean circuit can be used only for a fixed length input. Additionally, as there are no cycles in the Boolean circuits and the input is limited to a finite size then the amount of work that is done in the boolean circuit is also finite. In order to model cycles we would have to increase the size of the circuit.

1.1 The complexity classes $\mathsf{SIZE}(T)$

We want to define a family of circuits in order to speak about circuits with variable length input.

Definition 2 A family of circuits is $\{C_n\}_{n\in\mathbb{N}}$, where C_n is a circuit with n inputs and 1 output. The family is of size $T : \mathbb{N} \to \mathbb{N}$ if $\exists c \forall n : |C_n| \leq c \cdot T(n)$. $\diamondsuit$

Therefore, we have to define a boolean circuit for each input length. The size of the boolean circuit is the number of nodes it contains, the number of edges does not differ much from the number of nodes as there are not many input edges for each node. $\mathsf{SIZE}(T)$ limits the size of the boolean circuit. If a language belongs to $\mathsf{SIZE}(T)$ if it can be recognized by a family of boolean circuits that is size-limited by $O(T)$.

Now we will define a class of languages that can be recognized by boolean circuits with polynomial size. We denote this class with $\mathsf{P/poly}$.

$$\mathsf{P/poly} = \cup_{c\in\mathbb{N}} \mathsf{SIZE}(\lambda n.n^c).$$

We call the circuit model of computation non-uniform. The notion of non-uniformity comes from the fact that for inputs of different lengths the computations can be different. We notice that the total size of every family of circuits is infinite, because the description of the family would be infinite.

1.2 Effects of non-uniformity

Every language is recognizable by a family of boolean circuits. For that $O(2^n \cdot n)$ nodes are needed.

Claim 1 *$L \in \mathsf{SIZE}(\lambda n.n \cdot 2^n)$ for any language $L \subseteq \{0,1\}^*$.*

Proof Let there be a language L, then we must construct a circuit C_n that recognizes $L \cap \{0,1\}^n$. So we will have to construct such C_n. As L is finite, it is sufficient to look through/write down all possible words in L in order to show that C_n recognizes it. So we need n inputs, let them be $b_1, b_2, \cdots, b_n$ and all possible words of the language $L \cap \{0,1\}^n = \{w_1, w_2, \cdots, w_k\}$. Now we will have to check if $b_1, b_2, \cdots, b_n$ can give w_i if $i \in \{1, k\}$. Each b_i, $i \in \{1, k\}$ has the value 0 or 1. The value could be negated and therefore we have to add an operation between each b_i and w_i. If there is no negation, then the operation will be an identity function. To check if the input to the w_i-s gives w_i we will use the conjunction operation and therefore there will be conjunction nodes with n input edges. However, we can replace these nodes if we expand the n-input conjunction into a boolean circuit of conjunctions with each node having two inputs, doing this will increase the size of the initial boolean circuit by $n-1$ nodes for each w_i. Finally, a disjunction of w_i-s has to be computed and as the disjunction has many input edges we will have to expand the disjunction, in total the expanded disjunction takes $2^n - 1$ nodes. The completed boolean circuit C_n is depicted on Figure 2.

Now we can find out the size of the circuit C_n. Expansion of the disjunction takes $2^n - 1$ nodes. Expansion of the conjunction node takes $2n-1$ nodes and there can be 2^n different w_i-s. The input nodes and the output node take $n+1$ nodes. So, the total size of C_n is

$$|C_n| \leq 2^n - 1 + 2^n(2n-1) + n + 1 = 2n \cdot 2^n - 1.$$

■

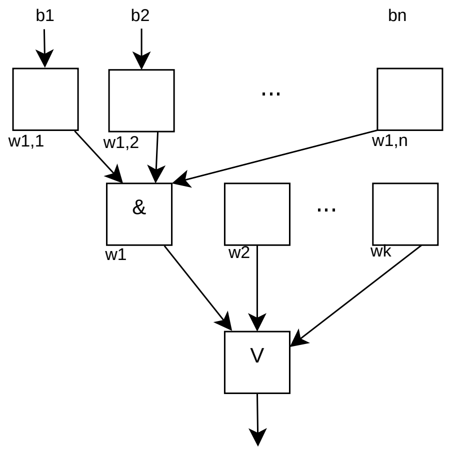

Figure 2: How to build C_n in order to recognize a language.

Claim 2 $L = \{x \in \{0,1\}^* | TM_{|x|} \textit{ stops on input } 1^{|x|}\} \in \mathsf{SIZE}(const)$.

In this claim, a language L is defined where a word belongs to L iff all words of the same length belong to L. Hence this language can be recognized by the following family of circuits:

$$C_n = \begin{cases} \mathsf{true}, & \text{if } \exists w \in \{0,1\}^n : w \in L \\ \mathsf{false}, & \text{if } \forall w \in \{0,1\}^n : w \notin L \end{cases}.$$

We also notice that if we can recognize whether a word belongs to the language L, then we can also solve the halting problem. Hence a language with high uniform complexity (defined through Turing machines) can have a small circuit complexity.

1.3 Uniformly generated circuits

Definition 3 A family of circuits $\{C_n\}_{n\in\mathbb{N}}$ in uniform if there exists a Turing machine M, such that $M(1^n)$ outputs C_n. ♢

The family is poly-time uniform if the TM M works in polynomial time.
The family is log-space uniform if the TM M works in logarithmic space (in n). When we require the generation in logarithmic space, we implicitly still require that the generation takes polynomial time (otherwise the generation cannot stop at all). I.e the size of C_n is still polynomial in n.

1.3.1 Uniformly generated circuits and class P

We consider a boolean circuit to be efficiently computable if it can be computed in P. We consider a family of circuits efficiently computable if it is uniform.

Theorem 3 *A language L is accepted by a poly-time / log-space uniform family of circuits iff $L \in P$.*

If we have a family of boolean circuits generated in polynomial time, then the languages that are accepted by this family are exactly these that are accepted by algorithms running in polynomial time.

Proof We will rewrite the theorem in two statements and then show that these statements are equal.
1. $\exists$ a uniform $\{C_n\}_{n\in\mathbb{N}}$ generated in polynomial time by M_G that accepts language L.
2. $\exists$ DTM M that works in polynomial time and accepts language L.

First we will show 1. $\Rightarrow$ 2. and then 2. $\Rightarrow$ 1.

1. $\Rightarrow$ 2. In this case M generates C_n and then computes its output. First M will run M_G on the input n to generate C_n and then M will run C_n.

2. $\Rightarrow$ 1. We have M and we will have to describe how M_G functions. M_G gets the input and has to output C_n that works for any input with length n. $M_G(1^n)$ could take the description of M and generate a circuit of it, for that M_G would have to expand the cycles in M. This can also be explained with a universal Turing machine that interprets M. Such Turing machine would check what is the next step that M would make and then it would make that step. Then it could go to the next step of M and do the same. With such moves the universal Turing machine can expand the cycles of M. It is straightforward to translate such behaviour into a circuit. ■

Corollary 4 $\mathsf{P} \subseteq \mathsf{P/poly}$.

The corollary is obvious as $\mathsf{P/poly}$ can be a uniform or non-uniform family of circuits and P corresponds to a uniform family of circuits.

2 Turing machines that take advice

Definition 4 Let $f, g : \mathbb{N} \to \mathbb{N}$. A language L belongs to the class $\mathsf{DTIME}(f)/g$, if there exists a family of bit-strings $\{\alpha_n\}_{n\in\mathbb{N}}$, where $\alpha_n \in \{0,1\}^{g(n)}$ and a deterministic Turing machine M working in time $O(f)$ (considering only the first argument) so that for all $n \in \mathbb{N}$ and $x \in \{0,1\}^n : x \in L \leftrightarrow M(x, \alpha_n)$ accepts. $\diamondsuit$

In this definition α_n is called the *advice*.
E.g., if L is unary, i.e. it consists of bit strings of the form 1^n, then $L \in \mathsf{DTIME}(\mathit{const})/\mathit{const}$. In this case the advice tells for every value of n if 1^n belongs to the language. With this construction we assume that the input contains only ones because checking the input would mean that the advice cannot be a constant.

2.1 Advice vs. circuits

Theorem 5 $\mathsf{P/poly} = \bigcup_{c,d \in \mathbb{N}} \mathsf{DTIME}(n^c)/n^d$.

Proof We will start by writing down the statements for which we want to show the equality.
1. $L \in \mathsf{P/poly}$ means that $\exists \{C_n\}_{n\in\mathbb{N}} : |C_n| \leq \mathit{poly}(n)$ and C_n accepts $L \cap \{0,1\}^n$.
2. $\exists M, \{\alpha_n\}_{n\in\mathbb{N}}$: so that the runtime of M is $\leq \mathit{poly}(n)$ and $\alpha_n| \leq \mathit{poly}(n)$ and $M(\cdot, \alpha_n)$ accepts $L \cap \{0,1\}^n$.

1. $\Rightarrow$ 2. We will have to describe what is C_n given M, α_n and the runtime of M. We could transform M into a circuit. The circuit has n inputs corresponding to the first argument and $|\alpha_n|$ inputs corresponding to the second argument. It also has a single output. As M's runtime is polynomial, the circuit that corresponds to M is of polynomial size in n. Now, in order to complete the construction of C_n we would have to move the $|\alpha_n|$ inputs into the circuit. This is possible if these nodes are transformed into internal nodes with zero inputs and a constant output, the bit corresponding to α_n.

2. $\Rightarrow$ 1. A polynomial size C_n is given and we need to construct Turing machine M and α_n. We notice that C_n and α_n are related as both the family of C_n and family of α_n are infinite. Therefore, α_n encodes the graph C_n and M interprets Boolean circuits. The running time of M is linear in respect to the size of the Boolean circuit and the Boolean circuit was polynomial in respect to the input. In order to interpret the Boolean circuit in linear time in respect to the size of the circuit we will have to evaluate all the nodes. $\blacksquare$

3 Karp-Lipton theorem

Theorem 6 *If $SAT \in \mathsf{P/poly}$ then $PH = \sigma_2^p$.*

This theorem also says that if polynomial hierarchy does not collapse then $SAT \notin \mathsf{P/poly}$. The theorem binds the uniform family of circuits and the non-uniform family of circuits.

Proof For the proof we will show that if $SAT \in P/poly$, then $\Pi_2^p \subseteq \sigma_2^p$. We will take a Π_2^p complete task and show that it is in σ_2^p. Let $\{C_n\}_{n\in\mathbb{N}}$ be the polynomial size circuit family recognizing SAT. Now we recall that

$$\Pi_2 SAT = \{\langle \vec{u}, \vec{v}, \varphi(\vec{u}, \vec{v})\rangle \,|\forall \vec{u} \exists \vec{v} : \varphi(\vec{u}, \vec{v}) = \mathit{true}\}.$$

We also recall that SAT is self-reducible, i.e. SAT is solvable if there is an oracle who solves any instance of SAT. If we have an oracle which solves SAT then it is easy to find a satisfying evaluation for SAT. If the whole formula can be evaluated to true then we can take the first variable and evaluate it as true and then check if the whole formula can still be evaluated to true. If fixing the first variable true made it impossible to evaluate the whole formula to true then the first variable has to be false. By continuing like this a satisfying evaluation can be found. This can be also be written as a Boolean circuit. Hence we deduce that the existence of $\{C_n\}_{n\in\mathbb{N}}$ implies the existence of polynomial size circuit family $\{C_n\}'_{n\in\mathbb{N}}$, such that

$$\varphi \in \{0,1\}^n \text{ is satisfiable } \Rightarrow \varphi(C'_n(\varphi)) = true.$$

If φ does not have a true evaluation then φ is false for every input.

If we are not worried about size and order, then a formula in the form $\forall u \exists v : \varphi(u,v)$ can be written as $\exists f \forall u : \varphi(u, f(u))$. This can be done because $\forall u \exists v$ is like a formula which for every u outputs a corresponding v. The drawback of this approach is that the description of f could take a large number of bits. However, the existence of a polynomial size $\{C'_n\}_{n\in\mathbb{N}}$ allows to express f in a polynomial number of bits.

The next formula can be interpreted so that for $\forall u \exists v$ so that $\varphi(u,v)$ and the formula evaluates to true iff $\forall u \exists v$ but existence of v is written as a function that gives us v. This function can be written so that it is C' and it outputs a true evaluation for a formula that has a true evaluation. The input formula for C' will be φ, where the value of u has been already fixed to be empty. Therefore, the only variable in φ will be v.

$$\langle \vec{u}, \vec{v}, \varphi(\vec{u}, \vec{v}) \rangle \in \Pi_2 SAT \Leftrightarrow \exists C' \in \{0,1\}^{poly(|\varphi|)} \forall \vec{u} : \varphi(\vec{u}, C'(\varphi(\vec{u}, \vec{\cdot}))) = true.$$

The problem in the right is in σ_2^p. This problem can be checked in polynomial time because C' is of polynomial size, interpreting the formula is done in polynomial time, partial evaluation of φ is in polynomial time and computing φ is also done in polynomial time.

In the proof we assumed that we have C_n but we used it only for showing that C'_n exists and we used the existence of C'_n to check if a true evaluation exists. ∎

4 Size hierarchy

Now we will speak about the size hierarchy of families of circuits.

Claim 7 *Most functions require large circuits. I.e. for a large enough n, all but exponentially small fraction of functions $\{0,1\}^n \to \{0,1\}$ requires a circuit of size at least $2^n/(10n)$.*

Proof This is because there are not so many small circuits as there are functions. There are 2^{2^n} functions with n-bit input and 1-bit output. As the output can be 0 or 1 we get two options for the output. The number of functions comes from the fact that we take the number of output elements to the power of the number of input elements. We would like to know how many Boolean circuits exist with the size s if the Boolean circuit would be of the type with n-bit input and 1-bit output and s would be the number of internal nodes. We could say that in every node there is one operation, AND, OR, NEGATION. Therefore there are three possibilities for each internal node and as there are s internal nodes there are a total of 3^s different possibilities for the operations. For the first node the input comes from n ancestor nodes, for the next node the input can come from $n + 1$ ancestor nodes and for the last node the input can come from all other nodes. We could approximately say that for each node among the s internal nodes we could choose any other node as the ancestor from s internal nodes and n input nodes. Therefore, for a node there are $n+s$ ways for choosing the first ancestor and $n + s$ ways for choosing the second ancestor. We have s internal nodes, so in total there are $[(n + s)(n + s)]^s = (n + s)^{2s}$ ways for choosing the ancestors. Therefore, there are up to $3^s(n + s)^{2s}$ Boolean circuits of this type. If $s \leq 2^n/(10n)$ then this number is much smaller than 2^{2^n}. ■

Theorem 8 *If $n < T(n) < T'(n) < 2^n/(100n)$ and $T \cdot log^2 T \in o(T')$, then* $\mathsf{SIZE}(T) \subsetneq \mathsf{SIZE}(T')$.

Proof For every l, there is a function $f_l : \{0, 1\}^l \to \{0, 1\}$ not computable by a circuit of size $2^l/(10l)$ but f_l is computable by a circuit of size $2^l \cdot (10l)$. The difference of the circuits is not very big, its of logarithmic size as $100l^2$ is proportional to the square of the logarithm of 2^l. We shift the difference to between T and T'. Let $s : \mathbb{N} \to \mathbb{N}$ be such that:

- $2^{s(n)} \cdot (10s(n)) \leq T'(n)$
- $2^{s(n)}/(10s(n)) \geq T(n)$

We can conclude that such s exists if T and T' have log-squared difference.

Let $g : \{0, 1\}^* \to \{0, 1\}$ be the following

$$g(x) = f_{s(|x|)}(lsb_{s(|x|)}(x))$$

then $g \in \mathsf{SIZE}(T')\backslash\mathsf{SIZE}(T)$. This means that by running g, f is applied to a part of x, to the rightmost bits of x. The circuit that computes g is like the circuit that computes f but g does not use a large part of the input bits. ■