

Computational Complexity
Prof. Subrahmanyam Kalyanasundaram
Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad



Lecture - 35
Formal Definition of Circuits

(Refer Slide Time: 00:15)

Lecture 35 - Definition of circuits
28 August 2020 10:48

Definition: A circuit is a directed acyclic graph, that have n inputs (source vertices) and a single output (sink vertex). The non-input vertices are the gates chosen from a 'basis' (usually AND \wedge , OR \vee , and NOT \neg).

→ Given a circuit, it computes a Boolean function, say $f: \{0,1\}^n \rightarrow \{0,1\}$.

Hello and welcome to lecture 35 of the course computation complexity. In the previous lecture we saw circuits many examples of circuits many example made different constructions of the same function, the threshold two functions. So, hopefully we have an idea of what really is a circuit. So, but now we will see the formal definition and some associated other definitions. So, as I had indicated in the previous lecture a circuit is nothing but a directed acyclic graph.

And that have n designated inputs and a single output. So, that if you see it as a dag or a directory cyclic graph, the n inputs are source vertices meaning all the edges are outgoing edges and the output is a sync vertex meaning there are only incoming edges, edges or edge. And all the edges or all the vertices that are not the input vertices are gates. And these gates are from a basis meaning these gates in all the examples that we saw, we allowed only AND gates and, OR gates and NOT gates.

But perhaps you could allow let us say XOR gates perhaps you could allow some other strange function may be threshold gates. But most of what we do will be only on the AND gate OR gate and NOT gate. But in general, a circuit could have could be formed out of other bases. But for the rest of this lecture and for the subsequent lectures we will assume that we have AND gate, OR gate and NOT gate and usually it will be fine bounded fine in do or something.

But we will make it clear when we have gates that have unbounded fan. So, again usually we will have De Morgan basis. So, the word De Morgan basis stands for AND gates OR gates and NOT gate with fan and variant and or have fan in 2, De Morgan basis. And one thing to note is that again given a circuit it computes a Boolean function from $0, 1$ to the n to $0, 1$. So, it is a Boolean function and it so all that we will be doing in this course is computing functions with only one output.

So, you could in theory compute a function like meaning from an n bit input you compute an m bit output that is also possible. But what we will be doing in this course is only we will only look at one bit outputs that itself is quite rich and captures a lot of interesting ideas and typically that goes along the lines of what we have been doing so far, we have been mostly dealing with decision problems given is this formula satisfiable? Is this graph three colourable?

Is this number prime? So, mostly we have been dealing with decision problems not computed function of a given input.

(Refer Slide Time: 03:50)



→ Given a circuit, it computes a Boolean function, say $f: \{0,1\}^n \rightarrow \{0,1\}$.

How does one use these circuits for computation?
When do we say a Boolean circuit decides, say SAT?

Notice that a circuit with n input gates can only handle inputs of length n .



So, now one thing to note is that it takes an n bit input and produces one bit output. But now it is not very clear or not very immediately clear how this can be used for a computation. Let us say you want to solve a Turing machine. A Turing machine is you just give it the input and it does whatever processing it does and then decides accepts, rejects and so on. But a certain fixed circuit let us say I have a circuit that has n bits.

Now n bit input, now this will take only inputs of a certain size, the end bit size. But given a fixed circuit it cannot possibly compute SAT senses of all sizes. So, now how do we formalize the notion of computing a language using a circuit? So, be it is be three colourable be it shortest path whatever, how can we use the circuits to compute a decision a language which have all inputs of all sizes.

(Refer Slide Time: 04:59)



only handle inputs of n

let $C_i(x)$ denote the output of circuit C_i on the input x .

Def: $\mathcal{C} = \{C_i\}_{i=1,2,\dots}$ is a circuit family where each C_i has i inputs. We say that the family \mathcal{C} decides a language $A \subseteq \{0,1\}^*$ if

$$L(C_n) = \{x \mid C_n(x) = 1\} = A \cap \{0,1\}^n$$

$x = n$ bits $(s(n))$ if there is



So, what we will do is to use a family of circuits. We will have a family of circuits one for each input size. And each circuit of the corresponding input size is supposed to handle that input size. So, what we will have is if family of circuits let us so we will use script \mathcal{C} to denote a family of circuits sub \mathcal{C} sub i . So, C_i is a circuit that takes i bits as input and outputs a single bit. And we say that this decides a language A , the family decides the language A .

If the i th circuit in the family decides A correctly for all the i bit instances of A . So, if the language corresponding to the so I have used n here, maybe I will replace it with i in the language corresponding to the i C_i is nothing but all the i bit instances of A . So, the language corresponding to C_i and what is the language corresponding to C_i ? Language corresponding to C_i is those sets of inputs those sets of i bit inputs. Remember C_i only takes as input i bit strings.

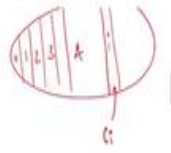
Those i bit strings for which C_i evaluates to 1 so given any i bit input any input there are some inputs which evaluate 1 some which evaluate to 0. So, $C_i(x)$ denotes those inputs that evaluate to 1 or sorry $C_i(x)$ is just the evaluation the language corresponding to C_i denotes those strings for which $C_i(x)$, C_i evaluates to 1. So, where C_i is just the evaluation of C_i on the input x .

(Refer Slide Time: 07:13)

we say

$$L(C_i) = \{x \mid C_i(x) = 1\} = A \cap \{0,1\}^i$$

We say $A \in SIZE(S(n))$ if there is a family $\{C_i\}$ of size $S(n)$ that decides A .



We say B is a family of size $S(n)$ if $|C_i| \leq S(i) \forall i$. Similarly for the other measures.



And we say that a language A is in the size $S(n)$ if there is family of circuits of size $S(n)$ that decides A . So, just once more so suppose A is a big language and so all 0 bit instances 1 bit instance is so this is A . So, you can slice A into 0 bit instances, one bit instances all of them in A and we want so the i th slice this is the i th slice of A which means it is the set of in set of strings in A that are of length i , length exactly i that are in A .

Now we want the circuit C_i to accept exactly or to evaluate on to one to exactly these inputs that of length i that are in A . So, that is the requirement. So, you can think of the circuit family looking at each slice. And we say that A belongs to size of $S(n)$ if there is a family of circuits of size $S(n)$ that decides A . So, when do I say that so now what do I mean by saying? When I say that a family is C of size $S(n)$ what I mean is that the family is C_1, C_2, C_3 , it is a family of circuits.

So, I want C_1 to be at most of size at most $S(1)$, C_2 to be of size $S(2)$, C_3 to be of size at most of 3 and so on. So, I want C_i to be of size at most $S(i)$ for all i . This is why this is when I say C is a family of size $S(n)$. So, basically every time for any i , the size of C_i should be at most $S(i)$. So, hope this is clear and one thing to note is that this important distinction between circuits and Turing machines which you may have noticed already.

Turing machine that does something with say palindrome or decides the shortest path whether or it just there is a certain algorithm that it encodes and it does not it will take all inputs of all sizes. So, if it wants if you have to run a Turing machine that tests palindrome it you could give it any length input and it will take care of it. Whereas circuits, it needs a specific circuit for each input size for each input size.

So, this is what is called non-uniformity. So, potentially each input size could be handled differently because we need different circuits for each input size. And now I could have a certain structure for a certain like $i = 10$ and a different structure for $i = 11$ and yet another different structure for $i = 12$. This is not really something that happens in Turing machines. So, this particular thing the feature is called non-uniformity.

And we will see how this plays an important role in in the upcoming lectures. So, this non-uniform way of computing is something that circuits have. There are also other models that have this property but this is something the circuits have. And this sometimes if it is not restricted properly, it gives a lot of power to the circuits. So, let us see we will see that in the upcoming lectures. So, coming back to a family of size S_n family of size S_n is where each C_i is of size S_i .

(Refer Slide Time: 11:30)



Measures
Size = No. of gates
Uses = No. of edges
Depth = length of the longest path
from output to input



Note: We will largely focus only on
Boolean functions and Boolean circuits.



And similarly, we could define other measures like number of wires and depth you can depth this you can view the circuit as a tree or a directed acyclic graph. So, what is the maximum length of it? So, the output will be at the top the red one and let us say the green ones are the inputs. So, what is the length of the longest path from an input to the output that is the depth. So, if you look at it as a tree then it is just the height of the tree or depth of the tree.

So, what is the longest path from the red to any green. So, you could also define depth a family of circuits that have some d and depth in a similar manner.

(Refer Slide Time: 12:24)

NPTEL

Symmetric function: A function whose value is unchanged upon permuting the input bits.

e.g. $f(x_1, x_2, x_3) = f(x_2, x_1, x_3)$
 $= f(x_3, x_2, x_1)$

Monotone function: A function f such that $x \leq y \Rightarrow f(x) \leq f(y)$.

Monotone circuits: Uses only \wedge and \vee

And another point is that I have not explicitly said but we will only be focusing on Boolean functions and Boolean circuits. Boolean function speeds $0, 1$ to the n going to $0, 1$, there are you could have arithmetic circuits that deal with numbers over a field and so on. But we will not be looking at that at least in the next few lectures. Now couple of more definitions that I just want to state now, we will not be immediately using them but couple of more definitions.

These are very important definitions both in terms of circuits as well as functions. So, symmetric functions are those functions when that again everything is Boolean from now on. It takes a bunch of input bits and out produces $0, 1$ output. A symmetric function is a function whose value is unchanged when you permute the input bits. So, if f is a symmetric function, then f of x_1, x_2, x_3 will be the same as f of x_2, x_1, x_3 it does not matter how you permute it.

X is the same as f of x 3, x 2, x 1. So, any way you permute it is the same, all six ways will be the same. So, many functions that we can think of OR symmetric so AND is a symmetric function. It does not matter in what order you give the inputs OR is a symmetric function, XOR is also a symmetric function. Unless you have some function like negation of x and y so that will not be symmetric. So, examples are AND, OR etcetera.

So, you need more this is defined only for multi functions of multiple inputs. So, if you have like single bit input then this is because this becomes a kind of a not very meaningful definition.

(Refer Slide Time: 14:29)

NPTEL

Monotone function: A function f such that $x \leq y \Rightarrow f(x) \leq f(y)$.
→ Bitwise

Monotone circuits: Uses only \wedge and \vee gates

Exercise: Function $f: \{0,1\}^n \rightarrow \{0,1\}$ is a monotone function \Leftrightarrow it has a monotone circuit representation.

Theorem: For any language L , we have



The next definition is a monotone function, a monotone function is a function where if x is less than or equal to y then f of x will be less than or equal to f of y. So, in terms of like in the regime that we are in, we are talking about n bit vectors as inputs and 0, 1 as output. So, when I say x is less than or equal to y what I mean is in a bitwise manner. So, you can you can look at it each bit, bit by bit x should be less than or equal to y.

So, there should not be a position k where x is 1 and y is 0 everything else is okay, x could be 0 y could be 0, x could be 1 y could be 1 and x could be 0 y could be 1. But x less than or equal to y means there could not be any bit position where x is 1 and y should be 0. So, if x is less than or

equal to y then f of x is less than or equal to f of y . And monotone being monotone is a rather natural property. So, for instance given a graph does it have a clique of size 10.

And suppose a graph is expressed as a string of edges. So, you have let us say graph has a 20 vertices or 100 vertices then it has 100 choose two edges, so it expresses a string of edges. Now you are asking if the graph has a clique of size 10. So, now if I add an edge then the original graph had a clique of size 10. The addition of an edge will still continue to have that clique. However, if you did not have a clique of size 10 then the addition could potentially create a clique of size 10.

But it need not create. But if you add an edge one thing is for sure it will not remove a clique that is already there, so this is a monotone property. So, many natural properties especially in the context of graphs are monotone functions. So, a function that takes a graph as input and denotes whether it has a clique of a certain size is a monotone function or a matching of a certain size that will also be a monotone function.

So, even AND is a monotone function just because so and OR is also a monotone function. If you flip an input from 0 to 1 then the output can possibly flip from 0 to 1, but it cannot flip from 1 to 0. And another definition is of monotone circuits, monotone circuits are those circuits that use only AND and OR gates. So, it you can only have two types of gates AND or OR. And one thing that is interesting extremely interesting is that it is not a coincidence that monotone functions.

And monotone circuits have the same name. This is because monotone functions; function f is a monotone function if and only if it has a monotone circuits representation. So, the set of all functions that can be expressed using monotone circuits are exactly the class of monotone functions. So, that is why I said the naming is not an accident. So, this is something that you can try to show it is a maybe a bit maybe it may require a bit of thought, but it is something that you can try.

(Refer Slide Time: 19:09)

Theorem: For any language L , we have
 $L \in SIZE(O(2^n))$.

Proof: Consider $f(x_1, x_2, \dots, x_n)$. Let f correspond to the strip of L of input size n .

$$f(x_1, x_2, \dots, x_n) = [x_1 \wedge f(1, x_2, \dots, x_n)] \vee [\bar{x}_1 \wedge f(0, x_2, \dots, x_n)]$$

↑ output



Now this one important theorem it is that given any language. This language has a circuit family of size of exponential size, exponential meaning order 2 power n size for any language. So, this is something that is interesting and something that we do not have in the case of Turing machines. It is not like we cannot say that any language has a 2 power n time algorithm there are languages that require more time. So, let us see how that is. It is a fairly straightforward argument.

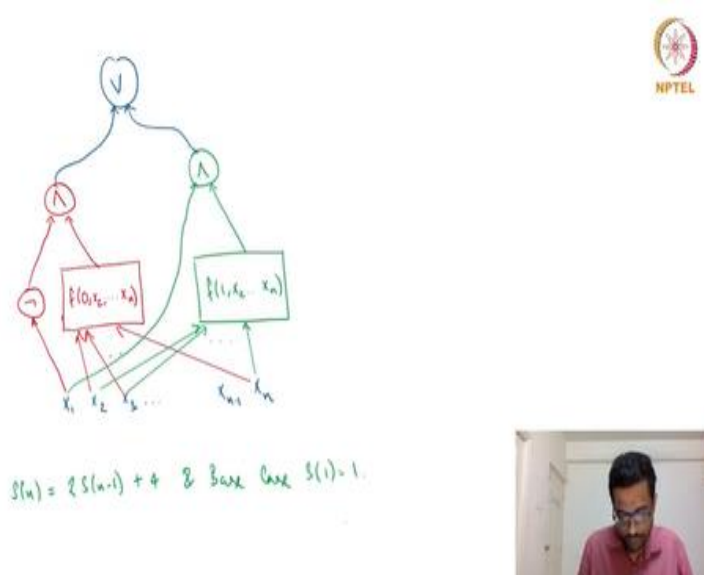
So, consider the function, so first of all when I say language we can as I said already, we can restrict it into strips of a certain size. So, consider any function, let f correspond to the strip of L of input size. So, now so f is an arbitrary function that corresponds to the so L is an arbitrary language, f is an arbitrary function. So, f corresponds to the strip of L length n . So, set of all f will be 1 only exactly on those inputs which are in L and of length n .

So, f is an n bit function; on n bits now consider that. So, now all we will do is that we will just show that f has a order 2 power n sized circuit representation and that is enough. So, we will show that odd f has a order 2 power n sized circuit representation. So, now let us see how f can be written. So, we could decompose f like this. So, we could consider f where x_1 is set to 1 and f where x_1 is set to 0, x_1 is set to 1 and x_1 is set to 0.

And you take the AND of the first with x_1 and the second term with x_1 complement that is the AND of the first term with x_1 and second term with x_1 complement and then the OR of these two factors. So, when x_1 is equal to 1, the first term is only the first term comes into play because the second one becomes negation of x_1 is 0, so it just vanishes and we have f of x_1, x_2 etcetera. And similarly; when x_1 is 0 only the second term comes to play so again it is fine.

So, you can easily verify that this equation is valid. Now all that we are going to do is to just build on this equation and rely on an inductive argument or inductive or recursive recurrence relation based argument for deriving the size of f .

(Refer Slide Time: 22:38)



So, f is x_1 and f of 1×2 and so on or x_1 complement and f of 0 and so on. So, now suppose we have two boxes or two black boxes, the red black box computes f of x_1, x_2 etcetera up to x_n and the green black box computes f of 1×2 up to x_n . And but then these needs the input x_2 up to x_n , they do not need x_1 but they need the rest of the input base. So, we give that and then you can easily verify that this computes exactly what we have here.

You take an AND of f of the red box with x_1 complement and the output of the green box with x_1 and then have an OR and that is it. So, this is a circuit that corresponds to this function f . So, you can verify that this is a circuit that corresponds to the above expression. Now so suppose S_n

is the size required to construct a function of a certain arbitrary function of from 0 1 to n, 0 1 to n, n to 0 1.

So, f requires an S_n size circuit and inductively speaking we are we are relying on two S_{n-1} size circuits. Because once we hard code 0 or one the rest of it is actually a function on $n-1$ bits. So, the green box and red box are both actually functions on $n-1$ bits because one bit is fixed and you could simplify whatever. And so, there are two so let us say S_{n-1} is the size bound for any function that requires $n-1$ bits.

Similarly, S_n is the size bound for any function that requires n bits. So, S_n is actually size of n is actually if you use this construction, it is 2 times S_{n-1} , one for the red box and one for the green box. And then how many other gates are there, so size is the total number of gates so number of gates in the red box number of gates in the green box and then there are four other gates so plus 4. And what is the base case? The base case is $S_1 = 1$.

When you have a one bit input and output basically you just you have at most a single wire $x_1 = x_1$, $x_1 = 0$, $x_1 = 1$ there will be four possible functions or x_1 equal the function is the complement of input. But in each of these cases the number of gates required is at most one, so $S_1 = 1$.

(Refer Slide Time: 25:41)

$$S(n) = 2S(n-1) + 4 \quad \& \quad \text{Base Case } S(1) = 1$$

$$\text{This solves to } S(n) = \frac{5}{2} 2^n - 4 = O(2^n)$$

Problem (Lecture): $L \in \text{SIZE}(O(2^n/n))$
for any L .



And if you solve this recurrence relation you will get this expression S_n is 5 divided by 2 multiplied by 2^{n-4} and this is ordered 2^n and that, is it. So, this shows that any function can be constructed in order 2^n in size in to be more particular to be more precise it is 5 divided by 2 times 2^n . So, it is an inductive argument because a recursive argument. In fact, this is a very simple idea but if you look a bit more closer and study a bit more using a bit slightly more involved idea.

You could show that actually for any language L you have a 2^n divided by n size circuit. So, here we show that it is order 2^n size circuit so now I am saying that actually this order 2^n can be reduced to 2^n and divided by n and this theorem is due to Lyapunov. So, in fact there is a tighter bound available and this 2^n divided by n bound is actually somewhat tight.

You cannot make it, you cannot really improve it in the sense that the you could probably improve the constant, but nothing beyond that. It so that is what I mean by subordinate. So, any function has a circuit of order 2^n divided by n size. And what we saw was in his function has circuit of size order two power in a function or language. And with that I will close this lecture. So, we saw the definition of A of a circuit we saw what time what one means when we say that a circuit can compute a language.

So, we by dividing this language into strips of each size, we saw so this is basically the idea of non-uniformity. And we also saw when we need a family of circuits to decide a certain language. So, we saw the definition of what we mean when we say the family has a certain size or a certain depth and then we saw this theorem said that any language has a order 2^n in size circuit and we also stated that any languages order 2^n divided by n size circuit, but did not prove it and with that I will close this lecture. Thank you.