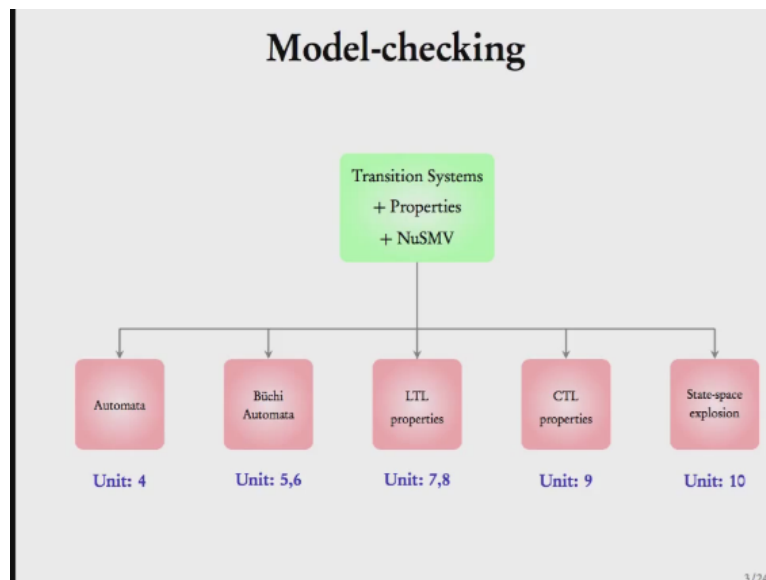


Model Checking
Prof. B. Srivathsan
Department of Computer Science and Engineering
Indian Institute of Technology – Madras

Lecture - 50
Introduction to BDDs

Welcome to Unit 11 of this course. In this unit, we will be looking at a data structure for representing transition systems and the name of this data structure is going to Binary Decision Diagrams (BDDs) in short. In first module, I will give you an introduction to BDDs.

(Refer Slide Time: 00:30)



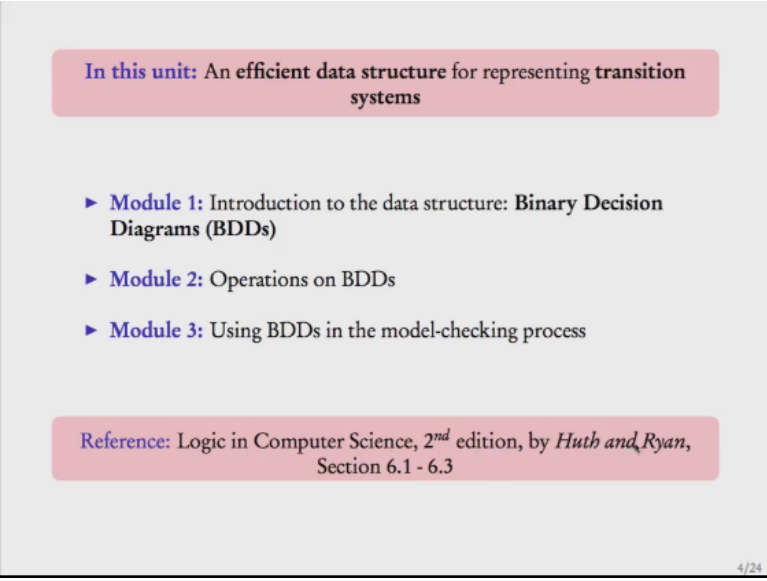
A short recap of what we have been seen in this course. Model checking is essentially, modeling systems as finite state machines and modeling requirements in a formal language and using automatic tools to check properties against these finite state machines. So, in our course we have been looking at transition systems these are diagrams with states and transitions. We have seen what kind of properties can be checked on transition systems in particular we have been looking at the LTL and CTL properties.

We have also seen the tool NuSMV that can read transition systems in a format of this tool and we can also write LTL and CTL properties and NuSMV can automatically check these properties on the transition system specified. In the last unit, we had seen that when we have multiple modules in NuSMV the number of states in the transition system increase exponentially. If you have

one Boolean variable there are two states.

If you have two one Boolean variable there are four states. If you have n Boolean variable there are 2^n states and so on. Clearly, we need an efficient way to represent transition systems since they are huge. In this unit we will look at

(Refer Slide Time: 02:22)



In this unit: An efficient data structure for representing transition systems

- ▶ **Module 1:** Introduction to the data structure: Binary Decision Diagrams (BDDs)
- ▶ **Module 2:** Operations on BDDs
- ▶ **Module 3:** Using BDDs in the model-checking process

Reference: Logic in Computer Science, 2nd edition, by Huth and Ryan, Section 6.1 - 6.3

4/24

a data structure for representing transition systems efficiently. In this module I will introduce you to this data structure called Binary Decision Diagrams. In module two we will see some operations that can be done on BDDs we will see algorithm for operations on BDDs and in module three I will explain how BDDs can be used to represent transition systems. This unit the reference would be the book Logic in Computer Science, second edition, by Huth and Ryan, Section 6.1 to 6.3.

(Refer Slide Time: 03:10)

x, y : <i>Boolean</i> variables		
$f(x, y) = x + y$ <i>Boolean function</i>		
$0 + 0 = 0$		
$0 + 1 = 1$ <i>Logical OR</i>		
$1 + 0 = 1$		
$1 + 1 = 1$		
x	y	$f(x, y)$
0	0	0
0	1	1
1	0	1
1	1	1
<i>Truth table</i>		

Let us start with what are called Boolean functions. During the course of the unit we will understand why we are talking about Boolean functions. What are Boolean variables? These are variables that can take values zero or one in other words false or true. Suppose x and y are Boolean variables, f of x, y a function given by $x + y$ is a Boolean function. Now, this plus is a Boolean addition. So $0 + 1$ is zero, $0 + 1$ is one, $1 + 0$ is one, $1 + 1$ would still be one.

So, this is the logical or so if look at it as false or false, it could be false. False or true will be true, true or false will be true and true or true will be true. And this is the Truth table corresponding to this Boolean function. When x and y are zero, f of x, y is zero, zero and one will be one, one and zero will be one. And one and one will be one. So, this is the standard Truth table for the Logical OR.

(Refer Slide Time: 4:44)

x, y : *Boolean* variables

$f(x, y) = x \cdot y$ *Boolean function*

$0 \cdot 0 = 0$
 $0 \cdot 1 = 0$
 $1 \cdot 0 = 0$
 $1 \cdot 1 = 1$

Logical AND

x	y	$f(x, y)$
0	0	0
0	1	0
1	0	0
1	1	1

Truth table

7/24

Now, let me give you an example of one more Boolean function. A Boolean function takes as input some Boolean variables and outputs either zero or one for each combination of inputs. So, this is a function defined as f of x, y equal to x dot y . Now what is this dot operation on Boolean variables? Zero dot zero is defined to be zero. Zero dot one is zero. One dot zero is defined to be zero. At one dot one is defined to be one.

So what is your guess? What is this dot? It is nothing but the Logical And. And here is the Truth table corresponding to AND.

(Refer Slide Time: 5:37)

x : *Boolean* variable

$f(x) = \bar{x}$ *Boolean function*

$\bar{0} = 1$
 $\bar{1} = 0$

Logical NOT

x	$f(x)$
0	1
1	0

Truth table

8/24

Now let me define yet another operation on Boolean variables f of x equal to x bar is another Boolean function. Now how do we define this bar operation? Zero bar is defined to be one and one bar is defined to be zero. So this is just the Logical NOT and here is the Truth table representation of this Boolean function.

(Refer Slide Time: 6:09)

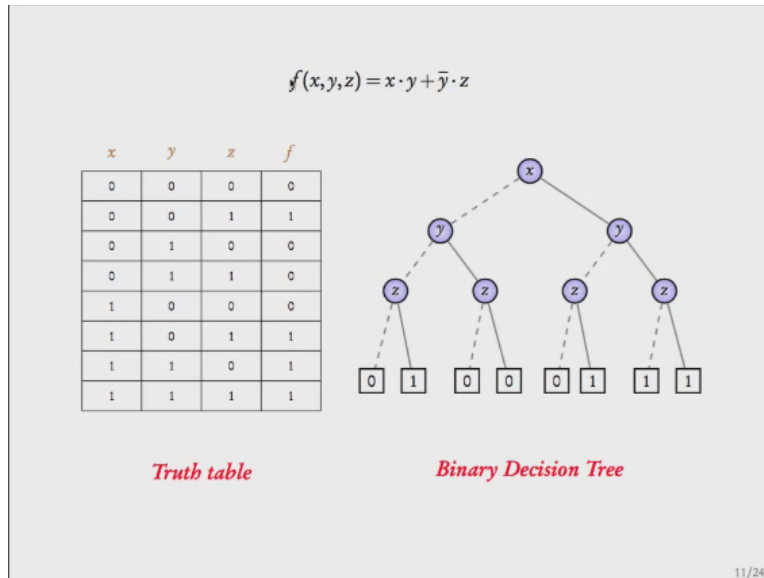
x_1, x_2, \dots, x_n : *Boolean variables*
 $f : \{x_1, x_2, \dots, x_n\} \mapsto \{0, 1\}$ *Boolean function*
 $+ \quad \cdot \quad -$ *Boolean operations*
 Examples: $f_1(x, y) = \bar{x} + y$, $f_2(x, y, z) = x \cdot y + \bar{y} \cdot z$, $f_3(x, y, z) = \overline{x + \bar{y} \cdot z}$

9/24

In general, if you have n Boolean variables a Boolean function is just a function that takes a valuation of x one to x n and maps it to either zero or one. It is a map from some valuation of x one to x n to zero or one and you can define Boolean functions using these basic Boolean operations plus, dot and bar. For example, this is a function x bar plus y . So this is a function over three variables. It says x dot y plus y bar dot z .

So this is formed out of these operations. Similarly f_3 of x, y, z is x plus y bar dot z the whole bar.

(Refer Slide Time: 7:16)



So, we have seen what Boolean functions are so far. Let us now see how to represent Boolean functions so far it is simple. So here is a Boolean function f of x, y, z is $x \cdot y$ plus $y \cdot z$. Now you can represent it using a truth table. So, this column is for x , this column is for y this column is for z and this is for f . Now if x is zero and y is zero $x \cdot y$ is zero. Now what about $y \cdot z$? Since z is zero $y \cdot z$ is going to be zero well and zero plus will be zero.

Let us see one more. $x \cdot y$, zero dot zero will be zero plus $y \cdot z$ will be one dot one which will be one. So this value is one. Similarly, you can check for the other n case. So, this is called a Truth table for this Boolean function. You just map each valuation to the corresponding value which f associates it with. There is another way of looking at this Truth table this is called the Binary Decision Tree. So the nodes are x, y and z . So, this is the root node x .

This dashed line says that the value associated to x is zero. The thick line says that the value associated to x is one. Similarly, here at y the dashed line says that the value associated to, y is zero. The thick line says that the value associated to y , is one. So, this path is zero, zero, zero and for zero, zero, zero f is zero. Now, what is this path zero for x , zero for y , and one for z . So, zero, zero, one is one. Now, zero, one, zero is zero and so it is zero here. Zero, one, one is zero again.

So, zero, one, one is zero again. Now what about this one, zero, zero is zero. One, zero, one is one. One, one, zero is one and one, one, one is one. So, this is just another representation of the

Truth table in the form of a Tree. Each path corresponds to an entry here. So, these two are equivalent representations for this Boolean function.

(Refer Slide Time: 10:27)

$$g(x,y,z) = \overline{f(x,y,z)} = \overline{x \cdot y + \bar{y} \cdot z}$$

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

x	y	z	g
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

What are the Operations that you can do on truth tables? We know operations that can be done on functions plus dot and the bar. Now, since we are representing Boolean functions using Truth tables we should be able to do these operations on Truth tables as well. Let us see how to do it. So, consider this Boolean function which is the f of x, y, z that we show the compliment of it. So, that is x dot y plus y bar z whole bar. Now, this is the Truth Table for f.

How do we get the Truth table for g? You look at the corresponding entry in f and just take the compliment if it is zero put it to be one. If it is one here put it to be zero. Zero here put it to be one. Zero here put it to be one. Now these three are one so here you will get zero, zero, zero. So, given the truth table of f you can compute the truth table of f bar by just swapping the values in this column. So, this is a simple concept.

(Refer Slide Time: 11:52)

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$f(x,y,z) = x \cdot y + \bar{y} \cdot z$$

$$g(x,y,z) = x$$

$$h(x,y,z) = f(x,y,z) \cdot g(x,y,z)$$

x	y	z	h
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

How about doing a plus? So this is a truth table for s . This is the truth table for function g now we define h to be this plus this. Now how the truth table of h look like. We look at the corresponding values in this one and this one and just do a plus. So, zero plus one will be one. One plus one would be one again. Zero plus zero, zero. Since here everything is zero we just copy this truth table here. So what you did?

You systematically went through each entry and the corresponding entry here and then filled it by doing a plus here as well as I mean by doing a plus of this and this. This is how you do plus of two truth tables. Similarly, how do you do a dot? You look at corresponding entries and instead of doing a plus you just do a dot. So, zero dot zero is zero. One dot zero will be zero and so on. Wherever there is zero you will get zero, zero, zero and here the last three are one and once, one, one.

(Refer Slide Time: 13:21)

Truth table representation for boolean functions

- ▶ **Space:** For n variables, needs to store $2^n \cdot (n + 1)$ bits
- ▶ **Operations:** Visit each entry of truth table
- ▶ **Sequential circuits** can be modeled using boolean functions

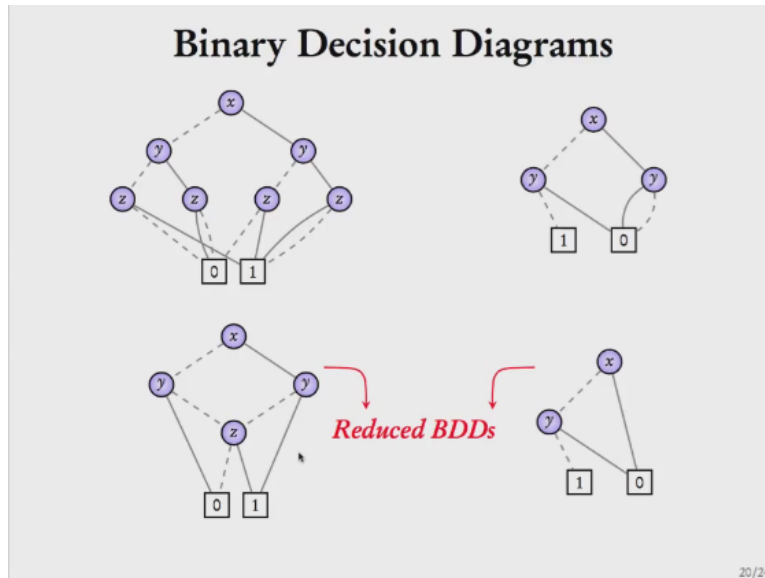
If boolean functions are represented using truth tables, a circuit with 100 variables needs **more than 2^{100} bits!**

16/24

What you need to observe? If you are representing Boolean functions using truth tables what is the space required? If there are n variables note that there are two power n rows and each row has n plus one pits. So, here there were three variables there were eight rows and each row has four pits. So for n variables the truth table needs to store two power n dot n plus one bits. And for computing operations you need to visit each entry of the truth table.

So, if you have n variables you need to look at two power n values. Note that sequential circuits can be modeled using Boolean functions. Sequential circuits are gates with possible feedback and they can be modeled using Boolean functions and a circuit with hundred variables will need more than two power hundred bits. And two power hundred is a huge number clearly we need a better way of representing Boolean functions.

(Refer Slide Time: 15:10)



This is what we are going to see next an efficient representation for Boolean formulas. So consider the Boolean formula $x \bar{y}$ this is the binary decision tree representations. This is equivalent to the truth table. Zero, zero goes to one because $x \bar{y}$ zero bar is one and y bar is, I mean zero bar is one again. So, one bar one may be one. Zero one is will go to zero. One zero will go to zero and one, one will go to zero again.

Fine, can we represent this in a better way? Firstly, there is no point of having three zeros it is enough if we have one representation for the bit one and one representation for the bit zero. Now, here this y both its successor edges go to the same node if you noticed this decision point see this is a decision node right. At this node if you choose to take one it goes here if you choose to take zero it will go to the node.

Here if you choose to take zero you go here. If you choose to take one you go here. Clearly this decision is unnecessary. What can you do to just remove this node and say that whenever x is one the value will be zero no matter what y is. This says that if x is zero then depending on whether y is zero or one there are differences but however when x is one we don't care what y is the value assigned will always be zero.

So, this is a more efficient way of representing it because we have lesser space. This kind of a diagram is called a Binary Decision Diagram. Let us see some more examples. Look at this

Binary Decision three. Let us see if we can make it smaller. First step, reduce all zeros to one zero and all ones to one, one just keep one leaf to zero and one leaf. Now this z it is zero child goes to zero and it is right I mean the one child goes to one.

Look at this both of them goes to zero both it is just goes to zero. Here the edge the dash edge goes to zero and the thick edge so the dash edge represents the decision point at z is sing zero it goes to zero and this thick edge goes to one. Now what about this thing both of them go to one. So when you reduce the leafs to this situation this is how the edges will look like. Now clearly this decision point and this decision point are unnecessary.

Once you come to this y and you say that when you take one you have to go to zero. Similarly, from here when you take one you have to go one. So, you can remove this z and this z you are left with this z and this z but note that the subtrees are isomorphic, in the sense look at this node. This z and this node z it is dashed edge goes to zero and the thick edge goes to one. Here again it is dash edged goes to zero and its thick edge goes to one.

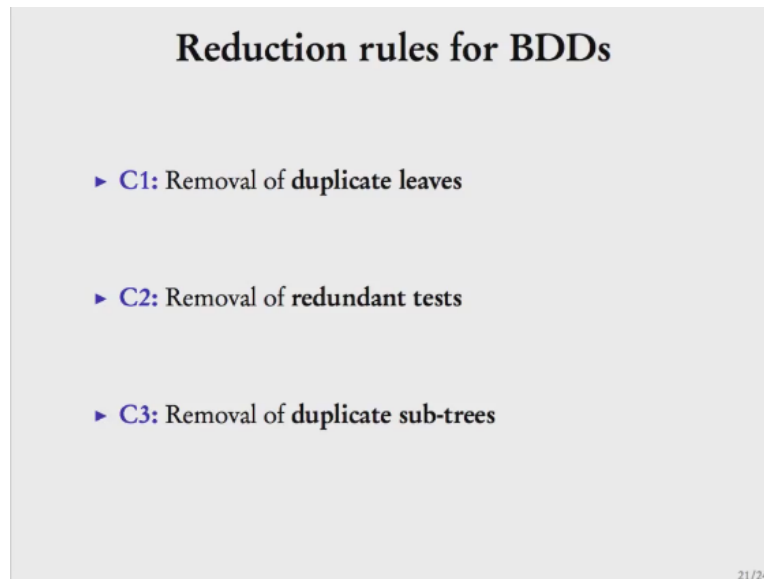
So, these two are identical so why do we need to have two representations for them instead we can just keep one representation for this particular tree and what do we get? We get one representation here and twice dash edge goes to this z. Similarly, this y dash edge goes here and the rest are same. This one dash edge goes here and the thick edge goes here and this ones one edge. So, I can also call it the one edge this goes to zero similar to this you just copy it.

Now, these usages less nodes then this one and this is a Binary Decision Diagram. So, these kind of diagrams are called Binary Decision Diagrams note that these two are called Reduced BDDs, BDDs is what? Binary Decision Diagram you look at this. This has the potential to be further reduced because this is an unnecessary node. Similarly, this is an unnecessary node. After doing the reductions we got this.

This cannot be reduced further see this y is different from this y because it is one edge goes to zero. Here it is one edge goes to one. So you cannot say that these two are identical. So, here we cannot merge any nodes so these are reduced BDDs similarly this is again a reduced in the –In

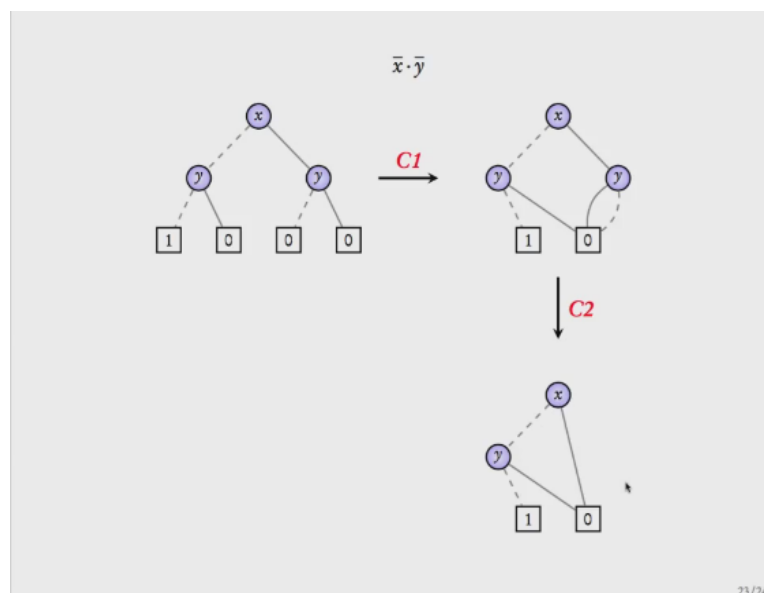
general such diagrams are called BDDs.

(Refer Slide Time: 21:31)



Now, we informally show to reduce. Let me give you the reduction rules. The rule C1 says we can remove duplicate leaves if you have multiple zero leaves and multiple zero ones we just make it to be one leaf for zero and one leaf for one. Removal of redundant tests, we were removing this unnecessary decision points and C3 removal of duplicate sub-trees. If there are two sub-trees which are identical we can just keep one representation fault.

(Refer Slide Time: 22:10)

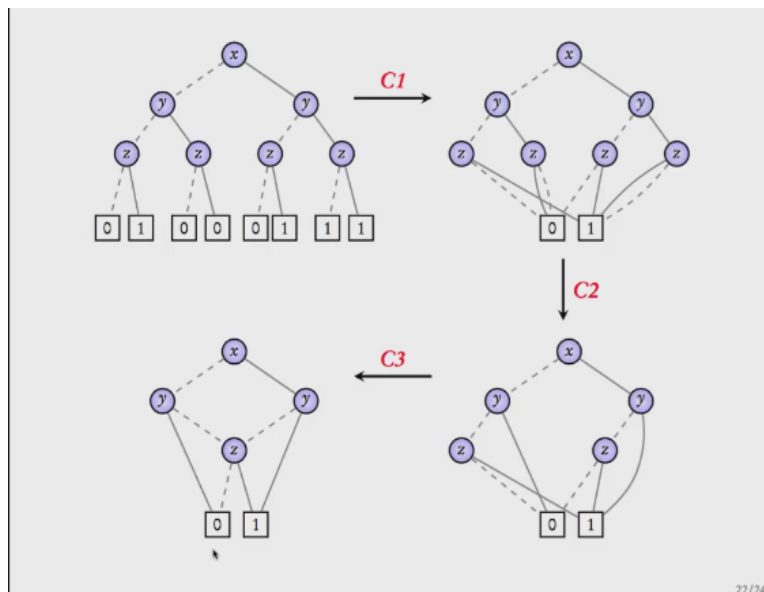


Let us go through the examples. So, what did we do first we applied the C1 rule to get one

representation for the leaves and one representation for the one. I mean one representation for zero and one representation for one. On this we applied C2. See there are redundant tests. This z and this z are redundant tests so by applying C2 we got –we remove this z's. So from this y we went directly to one and from this y on one we went directly to zero and we apply the C3 to remove duplicate Sub-Trees.

We just kept one Sub-Tree to represent these two which were identical. So, after applying these rules we get the reduced BDD. Similarly, here we applied C1 first to get one representative for the leaf and then we just had to apply C2 we could not reduce this any further. So this was a reduced BDD.

(Refer Slide Time: 23:24)



That is the introduction to this simple complex called BDDs. What do we want? We want to represent Boolean functions. A standard way of doing this is to use Truth tables. However, we can do better by making use of these BDDs. We have seen some examples and then there are ways of reducing these BDDs using the rules that we show. C1, C2, C3. They are very intuitive rules. Finally, they represent the same Boolean function.

The same truth table. For example, here this says that on an x if x is one, no matter what y is the value assigned is zero. Similarly, here it says that when x is one and y is one no matter what z is we get one if x is zero and y is one, no matter what z is we get zero. So, this is an introduction to

BDDs we will see in module III why we are interested in Boolean functions. We will be representing transition systems as Boolean functions and then for representing Boolean functions we will use reduced BDDs.

So, this ends the introduction to BDDs. You should be able to construct a binary decision tree from a Truth table and then apply the reduction rules to get the reduced BDDs if you know this you can jump to the next module.