CS 2742 (Logic in Computer Science)
Lecture 8

Antonina Kolokolova

September 28, 2009

3.1 Complete set of connectives

From what we have done it is easy to see that any possible truth table (we will start saying "boolean function" soon) can be represented by a formula written with just $\wedge, \vee$ and $\neg$. Such set of connectives is called *complete.*

A stronger result tells us that even $\vee$ is not necessary in this set (homework exercise). But $\neg$ is necessary, because every formula without $\neg$ is true when all of its variables are set to true. So contradiction cannot be described by such a formula, and any truth table with a 0 in the last column when all of its inputs are true cannot be represented by a formula using only $\vee$ and $\wedge$.

We have seen that there are two connectives that allow us to construct formulas representing all possible truth tables. Is there a single connective that can be used to construct formulas with all possible truth tables? The answer is yes, and such connective is called NAND (stands for "not-and") in digital circuit design, or Sheffer's stroke (written as |). It is defined so $p|q$ is false when both p and q are true, and true otherwise.

p	q	$p\|q$
1	1	0
1	0	1
0	1	1
0	0	1

You can see here that it has values exactly opposite of the values of $p \wedge q$, hence the name NAND. To show that it is complete, let us look at how to simulate $\neg$ and $\wedge$ with it. Once we have both $\neg$ and $\wedge$, we can rewrite any formula to an equivalent one which only uses | as a connective using ideas from the problem in your homework assignment.

To show how to represent $\neg p$, note that the first row of the truth table has 0 for the value of the Sheffer's stoke, and the last row has a 1. So $\neg p \iff p|p$. Similarly, $p \wedge q \iff \neg(p|q) \iff (p|q)|(p|q)$, since Sheffer's stroke is the negation of $\wedge$ and we just showed how to do $\neg$.

4 Boolean functions and circuits

The propositional logic is a special case of Boolean algebra. In Boolean algebra 0 corresponds to false (F), 1 to true (T), $+$ is $\vee$ and $\cdot$ it $\wedge$; here, $\neg$ corresponds to a unary $-$ of arithmetic, and $\neg a$ is written as $\bar{a}$. For this to be a proper Boolean algebra, the identities such as commutativity, associativity, distributivity, identity have to hold for $+$ and $\cdot$. We will skip the more general definition, and say for now that the rules are that $\bar{0} = 1, \bar{1} = 0$, $0 + a = a, a + 1 = 1, 0 \cdot a = 0$ and $1 \cdot a = a$ (note that there are more general Boolean algebras which we will define later). Note also that logic identities hold for Boolean algebras as well, and in fact can be derived from the axioms of Boolean algebra.

Just as arithmetic functions take as arguments some list of numbers often represented by variables, and output a number as an answer, Boolean function on n variables takes n inputs which have values 0 or 1, and produces a 0 or a 1 as an output. It is easy to see that, with that notational substitution, Boolean algebra and propositional logic are very much related: a propositional formula represents a boolean function when the formula is true on its variables iff the function on the corresponding values of its arguments outputs 1.

The easiest way to describe a Boolean function is to give its truth table (from which a CNF or DNF formula representing this function can be constructed). A boolean function is fully described by its truth table: there can be several formulas descibing the same function,but they must be logically equivalent. Usually when writing a truth table for a Boolean function we write 0s and 1s rather than Fs and Ts. Although we often can describe a function by a formula much smaller than its truth table, there are functions that cannot be described by anything smaller than the table itself. You will see a proof of this later if you take an advanced course on theory of computation.

Example 1. Consider a Boolean function $Majority(x, y, z)$ which outputs 1 (true) if at least two of its inputs x, y, z are 1s. It has the following truth table:

x	y	z	$Majority(x, y, z)$
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	0

This can be generalized by defining a Boolean function $Majority(x_1, x_2, \ldots, x_n)$ for every value of n; such a function outputs 1 if more than half of its inputs are 1. In that case, for every n there would be a different truth table describing the function.