In the first lecture we will be going much faster than in the rest of the course: much of this material should be known. It is most of Chapter 1 of Rosen.

1 Boolean variables, formulas, and circuits. Sentential Logic

1.1 Definitions

I assume you are familiar with Boolean variables (from programming). They range on the set \{F,T\} or \{0,1\} (False, True). They are also called bits. There are Boolean operators acting on them: \(\neg\) (NOT), a unary operator, defined by \(\neg 0 = 1, \neg 1 = 0\), and binary operators AND, OR, and IMPLIES, defined as

\[
\begin{array}{c|cc}
\text{AND} & 0 & 1 \\
0 & 0 & 0 \\
1 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{c|cc}
\text{OR} & 0 & 1 \\
0 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{c|cc}
\text{IMPLIES} & 0 & 1 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
\end{array}
\]
In the last table, we give the truth table of $p \rightarrow q$ with the values of $p$ in the first column.

These can be composed to make formulas (use parenthesis to disambiguate, if needed). For example

$$((p \rightarrow q) \land (q \rightarrow r)) \rightarrow r$$

An assignment consists of giving a value to each variable of a formula. This yields a value (by applying the operations as specified by their truth tables.) So an $n$-variable Boolean formula $E$ represents a Boolean function \{0, 1\}^n \rightarrow \{0, 1\}

Two Boolean formulas, $S$ and $T$ are equivalent iff they represent the same function. (Rosen calls this logical equivalence) For example, the formula $p \land q$ is equivalent to the formula $\neg(\neg p \lor \neg q)$

A formula $E$ is satisfiable if there is an assignment that makes $E$ have the value 1. If no such assignments exist, $E$ is unsatisfiable, if every assignment makes $E$ have the value 1, $E$ is a tautology.

**Example 1.** ($p, q$ are Boolean variables).

The formula $p$ is satisfiable. So is $p \land q$, and $p \rightarrow (p \land q)$

The formula $p \land \neg p$ is unsatisfiable, and so is $p \land ((q \lor \neg q) \rightarrow \neg p)$

The formula $p \lor \neg p$ is a tautology. So is $((p \rightarrow q) \land (q \rightarrow r)) \rightarrow r$

Easy exercise: formula $E$ is a tautology iff $\neg E$ is unsatisfiable.

**Notation:** we also use $p$ for $\neg p$, even to indicate negation in formulas, as in $\overline{a \lor b}$ for $\neg(a \lor b)$.

### 1.2 Usefulness

We can build gadgets that realize the Boolean operators. They can be mechanical, chemical, or electronic. The latter are particularly useful, as there are simple, fast implementations of Boolean operations by Boolean gates (see Rosen for pictures...)

In particular, we can interpret a sequence of Boolean variables (bits) as representing binary integers. It is not hard to devise compositions of Boolean gates (called circuits) that compute the sum of two numbers, or the result of their comparison. These. (together with registers that keep bit values over time) make up (in principle) digital computers...

Another important application is in Logic.

Declarative sentences can be true or false. (Examples of such sentences:
It is hot here now.
I have blue eyes.
There are 653 letters in the English alphabet.
As an aside, there are sentences that do not have a truth value. (For example:
Come here!
Integer x is bigger than integer y
Green ideas sleep furiously.)
We can use Boolean operations to express (fairly complicated) situations. For example we can express a Sudoku game by a (large) Boolean formula, stating for each blank position that it is 1 OR 2 OR 3 ... OR 9, that it is different from every other square in the same row, same column, or same subsquare. The resulting formula is satisfiable iff there is a solution.
Rosen presents a number of such encodings of dependencies between sentences in logic.

1.3 Results

**Definition 2.** A literal is a (Boolean) variable, or the negation of a Boolean variable. A clause is the AND of literals. A formula is in **Conjunctive Normal Form** (CNF) if it is the OR of clauses.

**Theorem 3.** Every Boolean function can be represented by a Boolean formula in Conjunctive Normal Form.

*Proof. Let* \( f(x_1, \ldots, x_n) \) *be a Boolean function of* \( n \) *variables* \( x_1, x_2, \ldots, x_n \). Consider the set \( S \) of Boolean \( n \)-tuples that cause \( f \) to output 1: \( S = \{ (a_1, a_2, \ldots, a_n) | f(a_1, a_2, \ldots, a_n) = 1 \} \). For each tuple \( s = (a_1, a_2, \ldots, a_n) \), \( s \in S \) define the clause \( t_s = t_1 \land t_2 \land \cdots \land t_n \), where for \( i = 1, 2, \ldots, n \) if \( a_i = 1 \) then \( t_i = x_i \) else \( t_i = \overline{x_i} \).

Note that the clause \( t_s \) will evaluate to 1 for exactly one input of \( f \), namely the \( n \)-tuple \( s \).

Now taking the OR of all such clauses results in a formula that represents \( f \). [Write out the proof that the formula gives the correct value for each input! Deal separately with the inputs where \( f \) is 1 and the inputs where \( f \) is 0.]

Note that the resulting formula describes a 2-level circuit: the inputs are all variables and their negations, then there is a level of ”fat” AND gates (\( n \) input bits instead of 2), and a ”very fat” single OR gate.
**Corollary 4.** Every Boolean function can be expressed using only OR gates and negations

This is true because we can always substitute AND gates using that $p \land q$ is equivalent to the formula $\neg(\neg p \lor \neg q)$. We can assume without loss of generality that NOT gates are at the bottom level [Prove!] and we can substitute an OR gate with many inputs by a tree of 2-input OR gates.

There are other small sets of Boolean operations with the property that they are functionally complete (can simulate all the others.) For example, NOT, AND are complete (because you can substitute $p \lor q$ by $\neg(\neg p \land \neg q)$ . The operation NAND by itself is functionally complete, as is NOR. On the other hand EOR, NOT is not functionally complete. [Prove these statements!]

### 1.4 Negative results

One would like to have some procedures to test whether a given formula is satisfiable, or whether it is a tautology. Note that in the interpretation of sentential logic, this would be asking whether a given statement is a theorem.

The good news is that there are several methods to decide whether a Boolean formula is a tautology.

One would be to compute the truth table of the function then verify that it evaluates to 1 on every input. Unfortunately this would involve $2^n$ evaluations for an $n$-variable formula.

One could use algebraic simplification and manipulate the formula with transformations that keep truth values, until one got to the constant 1. Unfortunately, we do not know of any good method that works efficiently for all formulas.

One could use proof rules that guarantee that if the formula is a tautology there is a proof of it. Again, we know of no procedure that would do this effectively.

It turns out that the question whether there is a polynomial time algorithm for testing tautologies is a fundamental unsolved problem in Complexity Theory! (Bets are that no such procedure exists...)

### 2 Predicate Logic

This is very informal.
Sentential logic is quite restricted. We need to express more complicated statements, involving variables (not necessarily Boolean variables) inside expressions. A predicate $P(x)$ is a function that to each value of $x$ associates a Boolean value. The domain of $x$ determines the kind of predicate – for example in can be an integer, a pair of real numbers, etc.

$x^2 > x$ is a predicate – its truth value depends on the parameter $x$ (and on the domain $x$ is defined – integers, reals, etc.)

$x + y > x$ is a predicate that is true if the domain of the variables is positive integers, but depends on the particular value of $x$ and $y$ over the natural numbers, the integers or the reals.

We can make predicates into sentences using quantifiers. The meanings of $\exists x P(x)$ and $\forall x Q(x)$ are clear. Make sure you understand that $\forall x \exists y U(x, y)$ does not mean the same as $\exists x \forall y U(x, y)$!