Notes for Lecture 9

Mathematical logic is the rigorous study of the way in which we prove the validity of mathematical statements. One of the applications of mathematical logic is that it provides a precise language to express mathematical statements and to write completely rigorous proofs. The simplest type of logical system is *propositional logic*. Propositional logic formalizes simple reasoning by cases and the meaning of negation, of implication, and of other ways to combine simpler statements into more complex ones. First-order logic is a more expressive formalism that, with some hacks, suffices to represent statements about integers and about computer programs, induction and so on. Second-order logic, which we will not talk about, is a generalization of first-order logic that works well to talk about real and complex numbers, and to formalize calculus, real and complex analysis, and more or less all of pure mathematics.

1 Propositional Logic

A *proposition* is a statement that can be true or false. Propositional logic studies how to reason about the truth of propositions and how the truth of a proposition relates to the truth of another one.

Every statement in propositional logic is an expression that is built up from *propositional variables*, by combining them with *logical connectives*.

Propositional variables are usually denoted with letters like p, q,[1] etc., and they are meant to stand for the propositions that we are interested in, that is statements like "$2 < 4$," "life will pass me by," "I don't open up my eyes," and so on. The connectives express how propositions are related, for example one can connect two propositions with an implication as in "if I don't open up my eyes, life will pass me by." Propositional logic allows us to reason that the above proposition means the same as "if life doesn't pass me by, then I have opened up my eyes" and also the same as "either I open up my eyes, or life will pass me by"

[1] We will also use letters like p, q, ..., to stand for *propositions*. Hopefully this will not cause confusion.

The logical connectives of propositional logic are:

Logical NOT written $\neg$. If p is a proposition, then $\neg p$ (pronounced "not p") is a proposition, and $\neg p$ is true if p is false, and vice versa;

Logical AND written $\wedge$. If p and q are propositions, then $p \wedge q$ (pronounced "p and q") is a proposition, and it is true if and only if both p and q are true;

Logical OR written $\vee$. If p and q are propositions, then $p \vee q$ (pronounced "p or q") is a proposition, and it is true if at least one of p and q are true;

Logical Implication written $\rightarrow$. If p and q are propositions, then $p \rightarrow q$ (pronounced "p implies q") is a proposition, and it is true unless p is true and q is false;

Biconditional written $\leftrightarrow$. If p and q are propositions, then $p \leftrightarrow q$ (pronounced "p if and only if q") is a proposition, and it is true if and only if the truth values of p and q are the same. (Meaning they are either both true or both false.)

We also have two special symbols: $\top$ is a proposition, and it is always true, and $\bot$ is a proposition and it is always false.

For a given proposition p, it is often helpful to construct its *truth table*, that is the list of all the possible truth values of the propositional variables occurring in p, together with the corresponding truth value of the proposition.

Here are the truth values of the propositions obtained by applying the logical connectives to propositional variables.

p	$\neg p$
F	T
T	F

p	q	$p \wedge q$
F	F	F
F	T	F
T	F	F
T	T	T

p	q	$p \vee q$
F	F	F
F	T	T
T	F	T
T	T	T

p	q	$p \to q$
F	F	T
F	T	T
T	F	F
T	T	T

p	q	$p \leftrightarrow q$
F	F	T
F	T	F
T	F	F
T	T	T

The precedence of the operators is that $\neg$ binds the closest, followed by $\wedge$, $\vee$, $\to$ and $\leftrightarrow$. This means that the proposition

$$\neg p \wedge q \vee r \to q \wedge r$$

is parsed as

$$(((\neg p) \wedge q) \vee r) \to (q \wedge r)$$

To construct the truth table of a complex proposition p one can first parse it, and then construct the truth tables of all the smaller propositions into which we have parsed p, starting from the smallest ones. For example,

p	q	r	$\neg p$	$\neg p \wedge q$	$q \wedge r$	$(\neg p \wedge q) \vee r$	$((\neg p \wedge q) \vee r) \to (q \wedge r)$
F	F	F	T	F	F	F	T
F	F	T	T	F	F	T	F
F	T	F	T	T	F	T	F
F	T	T	T	T	T	T	T
T	F	F	F	F	F	F	T
T	F	T	F	F	F	T	F
T	T	F	F	F	F	F	T
T	T	T	F	F	T	T	T

Two propositions are *logically equivalent* if they have the same truth value for every possible assignment of truth values to the propositional variables that they contain. If propositions p and q are logically equivalent, then we write $p \equiv q$.

Here are some important logical equivalences. The first two are called De Morgan's laws.

$$\neg(p \wedge q) \equiv \neg p \vee \neg q \tag{1}$$

$$\neg(p \vee q) \equiv \neg p \wedge \neg q \tag{2}$$

$$p \to q \equiv \neg p \vee q \tag{3}$$

$$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p) \tag{4}$$

Note that these equivalences imply that every proposition can be transformed into an equivalent one that involves only the use of the connectives $\wedge$ and $\neg$, or only the use of the connectives $\vee$ and $\neg$.

Another example of equivalence is that the propositions $p \to q$ and $\neg q \to \neg p$ are logically equivalent. An easy way to verify equivalence of simple propositions is to construct their truth tables, and verify that they are identical. Note that for two propositions to be logically equivalent they do not need to be defined over the same set of propositional variables: for example, $p \to (\neg p \wedge q)$ is logically equivalent to $\neg p$.[2]

The notion of logical equivalence is one of the useful features of propositional logic: given a complex mathematical statement, we can replace various parts of the statement with propositional variables and obtain a proposition, then change the resulting proposition to a logically equivalent one, and finally replace back the propositional variables with the original statements. The new mathematical statement will be equivalent to the original one, regardless of the meaning and of the truth-value of the original statement.

For example, applying this approach to the equivalence $p \to q \equiv \neg q \to \neg p$ we obtain proofs by contrapositive.

A proposition that is true for all the possible truth-values of its propositional variables is called a *tautology*. For example, $p \to (p \vee q)$ is a tautology. Note that $p \equiv q$ holds if and only if the proposition $p \leftrightarrow q$ is a tautology.

2 First-Order Logic

In first-order logic, besides having propositional variables and boolean connectives, we have *quantifiers*, *predicates*, and *functions*.

[2] We can use truth-tables in order to check the logical equivalence of two propositions that involve different sets of propositional values: we just list, in the construction of the truth-table, all the possible truth-values of all the variables occurring in either of the two propositions.

It is easier to start from a couple of concrete examples of what a first-order *sentence* looks like.

The first-order sentence corresponding to the statement "If n is a natural number, then n^2 is odd if and only if n is odd" becomes

$$\forall n.nat(n) \rightarrow (odd(square(n)) \leftrightarrow odd(n))$$

The statement "you can fool some of the people all of the time" becomes

$$\exists x.person(x) \wedge (\forall t.time(t) \rightarrow canfool(x,t))$$

The statement "you can fool all of the people some of the time" becomes

$$\forall x.person(x) \rightarrow \exists t.(time(t) \wedge canfool(x,t))$$

In the above examples, $odd(\cdot)$, $person(\cdot)$, $time(\cdot)$ and $canfool(\cdot,\cdot)$ are *predicates*, that is, they are properties that may or may not be true for the *objects* that we give to them in input. The *arity* of a predicate is the number of objects that it takes as input. Then we have *variables*, like x, t, n, which are *bound* by the *quantifiers* $\forall$ and $\exists$. A sentence of the form $\forall x.S$ is true if the sentence S is true for every possible assignment of x to any object; a sentence of the form $\exists x.S$ is true if there if there is an object whose value we can assign to x so that S is true. We also have *functions*, such as $square(\cdot)$ that take as input a certain number of objects and output an object. The *arity* of a function is the number of inputs it takes. A predicate can have arity zero, in which case we call it a *propositional symbol*, and a function can have arity zero, in which case we call it a *constant*.

Finally, we have the same boolean connectives that we have already studied in the setting of propositional logic.

In order to understand the meaning of a sentence in first-order logic we need to know what is the set of all possible objects (called the *universe*) that the quantifiers refer to, what are the values of the propositional symbols and constants, and what are the values that the various predicates and functions return for each possible input. This information is called a *model* for the sentence. In a given model, a sentence is either true or false.

A sentence is *valid* if it is true in every model. The useful feature of first-order logic is that if we can prove that a given sentence is valid, then we know that the statement it formalizes is true, without having to know what any of the functions or predicates appearing in the sentence even mean.

Two sentences are *equivalent* if, in every model, they have the same truth-value.

Here are some more details (but not a completely rigorous treatment) of what are sentences in first-order logic.

- Variables and constants are *terms*
- If $f(\cdot)$ is a k-ary function, and $t_1, \ldots, t_k$ are k terms, then $f(t_1, \ldots, t_k)$ is also a term
- A propositional symbol is a sentence
- If $P(\cdot)$ is a k-ary predicate, and $t_1, \ldots, t_k$ are k terms, then $P(t_1, \ldots, t_k)$ is a sentence
- If t_1 and t_2 are terms, then $t_1 = t_2$ is a sentence
- If S is a sentence and x is a variable, then $\forall x.S$ is a sentence
- If S is a sentence and x is a variable, then $\exists x.S$ is a sentence
- If S and T are sentences, then the following are sentences: $\neg S$, $S \vee T$, $S \wedge T$, $S \rightarrow T$, $S \leftrightarrow T$.

A sentence is *well-formed* if all its variables are *bound*. The definition of bound versus free variables is as follows: in a term t, all the variables occurring in t are *free*. If $P(\cdot)$ is a k-ary predicate and $t_1, \ldots, t_k$ are terms, then $P(t_1, \ldots, t_k)$ is a sentence whose free variables are all the variables occurring in the terms. If S is a sentence in which x is a free variables, then $\exists x.S$ and $\forall x.S$ are sentences in which x is *bound* by the quantifier. (Being bound is the same as not being free.) When we construct a sentence by joining other sentences with boolean connectors, then the set of free variables of the resulting sentence is the union of the sets of free variables of the sentences that we started from.

For example, in

$$\forall x.(A(x, y) \rightarrow (\exists z.B(x, y, z)))$$

the variables x and z are bound, but y is free, and so the above sentence is not a well-formed sentence. Also note that in the sentence

$$\forall x.A(x, y) \rightarrow (\exists y.B(x, y))$$

the first occurrence of y is free, but the second occurrence is bound, which means that the above sentence is not well formed.

There is a general theory of how to write proofs of validity of first-order sentences, but it goes beyond the scope of this course to discuss it.

In "practice," mathematical statements as they appear in textbooks and research papers are formulated using a mix of first-order logic notation and plain English, just like pseudo-code in programming textbooks and computer science research papers is a mix of plain English and of valid commands in programming languages like C or Python.

In order to understand those statements, it is enough to understand the meaning of the symbols described above, and in order to follow proofs involving this notation, it is enough to be familiar with certain basic equivalences between first-order sentences.

First of all, all the equivalences of propositional logic are still true, and they can be applied within a sentence of first-order logic. For example, the sentence

$$\forall x \exists y.(A(x) \to B(y))$$

is equivalent to

$$\forall x \exists y.(\neg B(y) \to \neg A(x))$$

Another observation is that if you have two quantifications of the same type binding variables in the same sentence, then the order of quantification is not important. For example, the sentence

$$\forall x.\forall y.S$$

where S is a sentence in which x and y are free is equivalent to

$$\forall y.\forall x.S$$

and

$$\exists x \exists y.S$$

is equivalent to

$$\exists y \exists x.S$$

If you have different quantifiers, however, you cannot exchange them: the sentence

$$\forall x \exists y.(integer(x) \to (integer(y) \wedge smaller(x, y)))$$

which says that for every integer x there is a bigger integer y is true (in the model in which $integer(x)$ is true if x is an integer and $smaller(x, y)$ is true if x is smaller than y), but the sentence

$$\exists y \forall x.(integer(x) \to (integer(y) \wedge smaller(x, y)))$$

means, in the same model, that there is an integer y which is bigger than all other integers, which is false. So the two sentences are not equivalent because there is a model in which the first is true but the second is false.

When doing proofs by contradiction, it is important to be able to correctly negate a sentence. From the previous section, we know how to negate any expression involving only boolean connectives. Regarding quantifiers, the sentence

$$\neg \exists x.S$$

is equivalent to

$$\forall x.\neg S$$

which is intuitive: saying that it is not true that there is an x for which S is true is the same as saying for every x we have that S is false. Similarly,

$$\neg \forall x.S$$

is equivalent to

$$\exists x.\neg S$$