CprE 310 / Spring 2019

Lecture 3: Predicate Logic

By now, we have had an initial taste of what propositional logic is all about. We have seen how to model declarative sentences as propositions, and discussed ways to manipulate propositional variables using a number of logical connectives. We also discussed the concept of *logical equivalence* of different propositional expressions.

Today, we will discuss a slightly more general system of logic known as *predicate logic*. But more on that in a moment; first, a few more concepts and applications of propositional logic.

Formal checking/verification

Formal verification is a super-important aspect of software and/or hardware design. Basically, the high level idea of formal verification is to prove/disprove the *correctness* of a system (or algorithm) and verify whether it is functioning as desired.

Now for systems involving millions of smaller components (think of a chip with millions of transistors, or a software program with millions of lines of code), it will be beneficial to *abstract* the logic underlying the system, and use the tools from logical analysis (that we have been discussing) to check whether the abstraction makes sense or not.

As a simple example, imagine yourself as a circuit designer working in Intel. You study a complex chip design for several weeks or months, and after a lot of sweat and toil, come up with a *different* design that has identical functionality. How will you convince someone else that your new circuit is equivalent to the one before? The way to abstract this is to represent the functioning of the original circuit via a system of logical expressions (say P), represent the new circuit as a *new* system (say P') and show somehow (using a clever analysis) that $P \iff P'$.

The starting point for such an analysis is usually via *tautologies*. Recall that some propositions are tautologies: a tautology is true no matter what truth values its variables may have. (Similarly, some propositions are contradictions: they are always false.) For example,

$$p \vee \neg p$$

is a tautology no matter what the value of p is. Tautologies can be thought of as logical expressions that capture certain fundamental truths.

But wow do we verify whether something is a tautology? The surest (and brute-force) way to do this is to write out a truth table. For example, if we want to show that

$$s := (p \wedge q) \implies (p \vee q)$$

is a tautology, then we simply write out the truth table:

p	q	$p \wedge q$	$p \vee q$	s
T	T	T	T	T
T	F	F	T	T
F	T	F	T	T
F	F	F	F	T

and verify that the last column is all "T".

Of course, this procedure is somewhat cumbersome. The above example only used 2 propositional variables (p and q). For a proposition involving n variables, the truth table will have 2^n lines (**Exercise: Why?**) and for even small logical systems (with, say, $n = 30$ variables), we are already talking about *billions* of lines to evaluate.

An alternate approach that *sometimes* helps is to appeal to logical equivalence. We already saw some examples of logical equivalences.

- $(p \implies q) \equiv (\neg p \vee q)$
- $\neg(p \wedge q) \equiv (\neg p) \vee (\neg q)$
- $\neg(p \vee q) \equiv (\neg p) \wedge (\neg q)$

Recall from previous lecture that the last two are called "De Morgan's Laws". Of course, there are many more that we can use: all axioms/properties/theorems of Boolean logic that you learned in CprE 281 are perfectly OK to be used as logical equivalences.

We can use some combination of these equivalences to simplify the expression. For example:

$$\begin{aligned} s &:= (p \wedge q) \implies (p \vee q) \\ &\equiv \neg(p \wedge q) \vee (p \vee q) \\ &\equiv (\neg p \vee \neg q) \vee (p \vee q) \\ &\equiv (\vee p \vee p) \vee (\neg q \vee q) \\ &\equiv T \vee T \\ &\equiv T. \end{aligned}$$

The above sequence of arguments can be viewed as a *proof* that the original expression s is a tautology. It is not always obvious which particular sequence of logical equivalences should be used to prove that something is a tautology, but that comes with a bit of practice.

Satisfiability

Many propositions are neither tautologies nor contradictions: their truth value depends on the assignment of truth values to their internal variables. Such statements are known as *contingencies*. If there is *at least one* assignment of truth values to its variables that makes the overall expression true, then the expression is called *satisfiable*.

If p is not satisfiable, then p is (by definition) a contradiction and $\neg p$ is a tautology. A different way to define satisfiability is as follows: p is unsatisfiable if and only if $\neg p$ is a tautology.

As before, we can ask the same question: how to check whether a compound proposition is satisfiable? Again, the brute-force (but safest) method is via truth tables. For example, if we want to check whether:

$$s := (p \vee \neg q) \wedge (q \vee \neg r) \wedge (r \vee \neg p)$$

is satisfiable, we can write out the truth table (consisting of $2^3 = 8$ rows) and verify that s is satisfiable if p,q,r are either all true or all false, i.e., $p \equiv q \equiv r$.

The alternative is to simplify the expression via inspection. Since s is a conjunction of 3 propositions, for s to be true *each* of the sub-expressions $(p \vee \neg q), (q \vee \neg r), (r \vee \neg p)$ should be true. Therefore, if p is true then both r and q have to be true. and if p is false, then both q are r are false.

Exercise: verify that:

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

is *unsatisfiable*.

In fact, the general problem of checking whether an expression is satisfiable is called the SAT problem. Since the truth table contains 2^n rows, the brute-force algorithm to solve the SAT problem is said to have *exponential* running time. A famous open problem in computer science is whether there is any algorithm that runs substantially faster (i.e., with polynomial running time.)

Predicates

Not all statements are propositions *per se*. For instance:

$$x + 1 = 6$$

is true if $x = 5$, and false otherwise. Such statements are called *predicates*. We have already (briefly) discussed predicates before; if propositions can be modeled as variables, then predicates are analogous to *functions*.

A predicate, or a *propositional function*, is a statement whose truth value depends on one or more variables. We use the notation:

P(x): x + 1 = 6

and say that the *truth value* of *P(x)* depends on *x*. As another example, if we define:

Q(n): n is a perfect square

then *Q(1), Q(4), Q(25), Q(100)* are all true, while *Q(23)* is false.

Similarly, let:

P(x,y): x = y + 2.

Then, *P(4,2)* is true and *P(1,2)* is false.

The variable in a given predicate is assumed to possess values from a certain (unspecified) set called the *domain of discourse.* Consider:

P(n): n is a prime.

P(σ): σ contains at most 1 vowel.

In the first example, the domain of discourse is the set of positive integers. In the second, the domain of discourse can be the set of all words (or strings). Whenever you come across a predicate, it is important and useful to clearly identify the universe of discourse.

Quantifiers

Predicates can be true or false; some predicates are always true and some always false. a *quantifier* expresses the *extent* to which a predicate is true.

There are two main types of quantifiers. Given a predicate $P(x)$, consider the assertion:

"For all *x*, *P(x)* is true"

is an assertion that can be concisely written as:

$\forall x \; P(x)$

The symbol $\forall$ is called a *universal quantifier.* Observe that application of the quantifier has transformed the predicate *P(x)* to a *proposition* which has a specific truth value. For example,

$\forall x, \; x^2 \geq 0$

is a *proposition* that is true if the domain of discourse is the set of real numbers. Similarly, the

$\forall x, \; x + 1 = 6$

is false (it is only true if x is equal to 5). Synonyms for "for all x" include "for any x", "for each x", and "for every x".

On the other hand, consider the assertion:

"For some x, *P(x)* is true"

concisely written as:

$\exists x \; P(x)$

The symbol $\exists$ is called an *existential quantifier*. It stipulates that there is at least one element in the domain of discourse for which the predicate is true.

For example,

$\exists A$, A knows how to program in Java

is true if the domain of discourse is the set of CprE 310 students. On the other hand,

$\exists x, \; x + 1 = 6.5$

is true if the domain of discourse is the real numbers (the choice $x = 5.5$ makes it true), but false if the domain of discourse is the natural numbers.

Exercises: Convince yourself that the propositions

- $\forall x, (x+1)^2 = x^2 + 2x + 1$
- $\forall x, \quad x^2 \geq x$
- $\exists x, \quad x^2 = \pi$
- $\exists x, \quad x + 1 < x$

are true, false, true, and false respectively, assuming that the domain of discourse is the set of real numbers.

We can also *combine* universal and existential quantifiers. As an example: if we wish to express the proposition:

For every prime number, there exists a prime number that is larger.

using predicate logic, we can write it as:

$\forall p \; \exists q \quad p < q$

where the domain of discourse is the set of prime numbers. (In fact, this is a *true* proposition.)

Just as how the order of logical connectives is important, the order of writing down quantifiers is also important! In the above example, if we wrote down:

$\exists q, \; \forall p, \quad p < q$

then this would represent the statement:

There exists a prime number that is larger than every prime number.

which is a completely different statement! (one that is obviously false.)

Exercise: Suppose *P(x,y)* represents the predicate:

P(A,B):= A and B are related

where the domain of discourse is the set of all people that ever lived. Which of the following statements is true and which one is false?

$\forall A,\ \exists B,\quad P(A,B)$

$\exists A,\ \forall B,\quad P(A,B)$