Propositional Logic – Part 1 - Propositions

Slide 1

In part 1 on propositional logic, we will introduce propositions and logical operators and provide examples that illustrate how to translate between English and formal propositional logic.

Slide 2

Our first topics include the definition of a proposition and the six logical operators.

We begin by defining what constitutes a proposition or statement. To be a proposition, it must be possible to determine whether a statement is true or false. To make clear why this is not always possible, we will provide some examples of sentences that are not propositions shortly.

Next we consider the six logical connectives or operators that are used to form compound propositions.

The first is *and*. The symbol should be easy to remember since it is similar to an A.

The next is *or*. Its symbol is an upside down *and*. There are two types of *or* operators, an inclusive and exclusive. In propositional logic, *or* is always taken to be an inclusive or. When we examine the truth tables for these operators we will explain the difference between the two types.

The third operator is not. It is the only one of the six that is a unary operator, meaning it has a single operand. All the others are binary operators.

The fourth operator is symbolized by an arrow pointing to the right. It is the conditional operator, which is also often translated with the use of *if* and *then*.

The final operator is the biconditional operator symbolized by an arrow pointing in both directions. It is most often read as *if and only if*.

Slide 3

To help make it is clear why some sentences are not propositions we will give example of both several sentences that are propositions and several that are not.

We begin with those that are.

The first example is the sentence *Today is Friday*. At any time that this sentence is considered, one can clearly say that it is either true or false.

The next is a mathematical example $2 + 2 = 5$, which clearly is false.

Finally the sentence *Maryland is a state*, which is a true statement.

Next we consider some examples that don't constitute valid propositions.

The first example is *He is a student*. The problem with this sentence is that fact that it contains the pronoun *he*. Without knowing who he refers to, it is not possible to determine whether this statement is true or false, so it is not a valid proposition.

The next example is a mathematical one, $x + 2 = 4$. The problem here is similar to the previous example, although in this case it is the use of the variable x that makes it impossible to decide whether the statement is true or false without knowing what value x represents.

The final example is the sentence: *This statement is false*. The reason it is not possible to determine whether this is sentence is true is because it is self-referential. If we assume it is true, the sentence contradicts that because it states that it is false. A similar problem occurs if we assume it is false.

Slide 4

Next we provide some examples that illustrate the important skill of being able to take compound propositions stated in English and translating them to formal propositional logic.

To accomplish this task, we must symbolize the simple propositions in our English sentences with symbols for elementary proposition. p and q are commonly used for this purpose.

So we will represent the elementary proposition: *It is Monday* with the symbol p.

And we will represent the elementary proposition: *It is raining* with the symbol q.

Now let's examine some compound propositions

Our first example is the compound proposition: *It is Monday but it is not raining*.

What is important to understand is how certain words English are translated. In this case the word *but* translates to *and*. So in formal propositional logic, we write this compound proposition as p and not q.

The next example is the compound proposition *It is neither Monday nor raining.*

What is important again is properly translating the use of neither-nor. *Neither* translates as *not* and *nor* to *and not*. So this sentence translates to not p and not q.

Our final example is *It is not both Monday and raining*.

The key here is the significance of *both*. It is what leads us to the use of the parentheses giving not the quantity p and q. These are not the only English words that can translate to specific logical operators. *Unless* is another example. Think about how that might be translated. The reason that this skill is so important in programming is that program requirements are generally written in English. Consequently it is often necessary to translate compound English propositions such as these into conditions in *if* or *while* statements that are essentially written in formal propositional logic.

Slide 5

Equally important is the skill of translating mathematical statements to formal logic.

In particular, let us consider so examples that involve inequalities.

The first is $x \geq 5$.

The way that this inequality is read makes it clear that as a compound proposition it involves an *or*. Breaking it in elementary inequalities, it would be written as $x > 5$ or $x = 5$.

In the second example we consider how we often express a compound inequality in mathematics. It is not uncommon to write $0 < x < 6$ to mean that x is between 0 and 6 excluding the endpoints.

Properly translated to formal propositional logic, this compound inequality becomes $0 < x$ and $x < 6$.

The second example is again a good example of why understanding the proper translations of such compound propositions is important in programming languages. In some programming languages, such as C and C++, the compiler we accept the first version without detecting an error but misinterpret it. To achieve the correct interpretation, it must be written as it would when properly translated to formal propositional logic.

Slide 6

We conclude the part 1 on propositional logic by examining what constitutes correct syntax for compound propositions. The term well-formed formula is typically used to describe a proposition that has correct syntax. We define well-formed formulas with a recursive definition.

Recursive definitions must always have at least two parts, at least one base case and at least one recursive case. So we begin with the base case.

The base case is that the elementary propositions such as p, q and so on are well-formed formulas.

Next the recursive case. Assuming that p and q are well-formed formulas, then so are each of the following:

The quantity p and q.

The quantity p or q.

Not p.

The quantity p implies q.

The quantity p if and only if q. Notice that we have required parentheses around all but the one involving *not*. It is assumed that *not* being unary has highest precedence. Because no precedence is defined for the other operators, unlike what is typically true in most programming languages, parentheses are required. One final note is that according to this definition, our previous examples were technically not well-formed. So we should note that customarily we often omit the outer-most parentheses since they are assumed to be there.