1988

Artificial Intelligence through Prolog by Neil C. Rowe

Rowe, Neil C.

Prentice-Hall

http://hdl.handle.net/10945/36984

Appendix A: Basics of logic

Logic is the study of things that are either true or false and never anything in between. Such things are called *propositions* or *statements*. Propositions can be combined to form more complex propositions using "and", "or", and "not". The "and" of some propositions is true if each is true; the "or" of some propositions is true if one or more are true; and the "not" of a proposition is true if the proposition is false. Otherwise, the result is false. Figure A-1 summarizes these three operations for simple cases; together they define the *propositional calculus*.

Other operations can be built from these three. For instance, logical implication: A implies B if whenever A is true, B is true too. It's usually interpreted to mean (see discussion in Section 14.2) that this can only be false when A is true and B is false, so an equivalent proposition is "B or not A".

The "and", "or", and "not" operations obey some of the same laws as arithmetic operations. Order in an "and" or "or" doesn't matter, and you can do the operations in pieces, so "and" and "or" are commutative and associative. "And" is distributive with respect to "or", and vice versa:

A and (B or C) equals (A and B) or (A and C)
A or (B and C) equals (A or B) and (A or C)

This means that we can always "factor out" repeated propositions, or "factor in" a single proposition, and have an equivalent form. You can do this more than once for complex proposition, as for instance

(A and B) or (C and D) equals ((A and B) or C) and ((A and B) or D)
equals (A or C) and (B or C) and (A or D) and (B or D)

A *literal* is either a simple proposition (no "and"s, "or"s, or "not"s in its description) or the "not" of a single simple proposition. Then by repeated application of the distributive law, you can always convert any complex proposition to an "or", each item of which is an "and" of literals; this is *disjunctive normal form*, and it looks like this:

(A and B and...) or (C and D and ...) or (E and F and ...) or ...

By repeated application of distributivity in just the opposite way, you can always get an "and" each item of which is an "or" of literals. This is *conjunctive normal form*, and it looks like this:

(A or B or ...) and (C or D or ...) and (E or F or ...) and ...

where A, B, C, etc. are literals. Conjunctive normal form is easier to use in Prolog, and is especially important for the resolution methods in Chapter 14 of this book. In putting expressions into normal form, these simplifications help:

A and A equals A
A or A equals A
A and (not A) equals false
A or (not A) equals true

A and true equals A
A and false equals false
A or true equals true
A or false equals A
(not true) equals false
(not false) equals true

Here are four important examples:

A and (not(A) or B) equals (A and not(A)) or (A and B) equals A and B
A and (A or B) equals (A or false) and (A or B) equals A or (false and B) equals A
A or (not(A) and B) equals (A or not(A)) and (A or B) equals A or B
A or (A and B) equals (A and true) or (A and B) equals A and (true or B) equals A

Distributivity does not hold for "not"s with respect to "and"s and "or"s. But two related formulas hold (DeMorgan's Laws):

not(A and B) equals (not A) or (not B)
not(A or B) equals (not A) and (not B)

And

not(not(A)) equals A

hence two important relationships between "and" and "or":

not(not A and not B) equals not(not A) or not(not B) equals A or B
not(not A or not B) equals not(not A) and not(not B) equals A and B

"And"s, "or"s, and "not"s can be represented graphically with the standard gate symbols used in hardware design (see Figure A-2). Such a *logic gate* display can help you see patterns you wouldn't see otherwise. The left sides of the gates in the figure represent inputs (operands), and the right sides outputs (operation results). In such designs, it is usually assumed that lines crossing do not connect unless a there is a black dot at the crossing. Venn diagrams are another useful graphic representation; see Section 8.3.

So far we have only *propositional logic* or the *propositional calculus*. If we let propositions have variables, we get a more powerful kind of logic called *predicate logic* or the *predicate calculus*. We extensively illustrate predicate logic by example in this book, so we won't say much here, but it is important to explain the key feature of *quantification*. If some predicate expression is true for any value of an included variable X, then that the expression is *universally quantified* with respect to X. Another way of saying it is that the expression holds for every X. If an expression is true for some (but not necessarily all) values of an included variable X, then it is *existentially quantified* with respect to X. Alternatively we can say that there exists an X such that the expression is true.

Universal quantification has more practical applications than existential quantification, so we will refer to it more. Some additional laws of logic cover universal and existential quantification, but only two are particularly important in this book:

the "not" of something existentially quantified is equivalent to the universal quantification of the negation of that something;

the "not" of something universally quantified is equivalent to the existential quantification of the negation of that something;

As an example, suppose an appliance works properly. Then you can say that there does not exist a component X of the appliance such that X is faulty. But that is equivalent to saying that for every component X of the appliance, that component is not faulty.

Go to book index