Predicate Logic

CS236 - Discrete Structures
Instructor: Brett Decker
Fall 2021

Predicates

Propositional logic can only express propositions: statements that have a true or false value. A *predicate* is a parameterized proposition, denoted as $P(x)$, where x is constrained to some set of values. We call this set the *domain*. Essentially, a predicate is a function that maps each value of the domain to a truth value (this is why predicates are referred to as propositional functions). Thus, $P(x)$ has a truth value for each distinct value of x in its domain. Consider the following predicate's truth table:

x	$P(x)$
1	T
2	T
3	F
4	F

The predicate $P(x)$ above represents the proposition "$x < 3$" for $x = 1, 2, 3, 4$. Now consider the predicate $Q(y)$ which represents "y is odd" for the domain of $y = 1, 2, 3, 4, 5$:

y	$Q(y)$
1	T
2	F
3	T
4	F
5	T

Quantifiers

We often want to quantify predicates: we define the extent to which a predicate is true or false. We will cover the universal quantifier and the existential quantifier.

Universal Quantification: Definition 1, Section 1.4*

The *universal quantification* of $P(x)$ is the statement

"$P(x)$ [is true] for all values x in the domain."

The notation $\forall x\ P(x)$ denotes the universal quantification of $P(x)$. Here $\forall$ is called the *universal quantifier*. We read $\forall x\ P(x)$ as "for all x P of x." An element [value of x] for which $P(x)$ is false is called a *counterexample* of $\forall x\ P(x)$.

Existential Quantification: Definition 2, Section 1.4*

The *existential quantification* of $P(x)$ is the proposition

"There exists an element x in the domain such that $P(x)$ [is true]."

We use the notation $\exists x\ P(x)$ for the existential quantification of $P(x)$. Here the $\exists$ is called the *existential quantifier*.

Quantifiers Example:

The universal quantification is simply the conjunction of $P(x)$ for all values of x:

$\forall x\ P(x) := P(x_1) \wedge P(x_2) \wedge P(x_3) \wedge \ldots \wedge P(x_n)$

The existential quantification is simply the disjunction of $P(x)$ for all values of x:

$\exists x\ P(x) := P(x_1) \vee P(x_2) \vee P(x_3) \vee \ldots \vee P(x_n)$

Given the predicate, $P(x)$ represents "x is odd" for $x = 1, 2, 3, 4, 5$ is $\forall x\ P(x)$? If no, what is a counterexample?

Any even value for x will give a counterexample. Does $\exists x\ P(x)$? If so, what elements of x satisfy the quantification?

$x = 1, 3, 5$ are the values of x such that $\exists x\ P(x)$ (note we need only have one value of x where $P(x)$ is true to satisfy existential quantification on $P(x)$).

Precedence and Scope

The quantifiers $\forall$ and $\exists$ have higher precedence than all propositional logic operators.

Does $\exists x P(x) \wedge Q(x)$ mean, (a) or (b)?

(a) $\exists x[P(x) \wedge Q(x)]$
(b) $[\exists x P(x)] \wedge Q(x)$

The answer is (b). This may be confusing because the parameter x is used for both predicates. Thinking of predicates as propositional functions, it is important to realize that the predicate variables have *scope*: they are either *bound* or *free*. In (b), the value x is bound to the predicate $P(x)$, but the value x is free in the predicate $Q(x)$ (it is not bound by any value or quantifier). In order for us to evaluate predicates, all variables must be bound. Thus, we have to know what to do with the unbound x and $Q(x)$. In this course, are convention will be to transform all unbound variables in predicates to the universal quantifier—note that this matches how predicates are used in the Datalog programming language for `Rules`. Thus $\exists x P(x) \wedge Q(x)$ is transformed into $\exists x P(x) \wedge \forall x Q(x)$. It can be confusing to now have two x that are independent, so we often rename on of the quantified variables. We end up with $\exists x P(x) \wedge \forall y Q(y)$ after renaming the second x to y. We will formalize this process into a Rule of Inference in the next reading for Proof with Predicate Logic. The rule is called *Universal Generalization*.

Think about the problem of unbound variables with the following C++ code:

```
for (int i = 0; i < size; i++) {
    ...
}
std::cout << \i after for loop: " << i << std::endl;
```

The variable `i` outside of the `for` loop is free (undefined), so the program will not compile. How to handle unbound variable is important for logic and programming languages.

Logical Equivalences

The logical equivalences we saw in propositional logic apply to predicate logic as well. Here are some logical equivalences in predicate logic:

1. $\forall x\ [P(x) \wedge Q(x)] \equiv \forall x\ P(x) \wedge \forall x\ Q(x)$
2. $\neg\forall x\ P(x) \equiv \exists x \neg P(x)$
   Think in these terms: $\neg[P(x_1) \wedge P(x_2) \wedge \ldots \wedge P(x_n)] \equiv [\neg P(x_1) \vee \neg P(x_2) \vee \ldots \vee \neg P(x_n)]$
   In English: "Not every x in $P(x)$ is true" or "There exists an x in $Q(x)$ that is false"
3. $\neg\exists x\ P(x) \equiv \forall x\ \neg P(x)$
   In English: "There does not exist an x such that $P(x)$ is true" or
   "For all x there is not a $P(x)$ that is true"

Conclusion

Predicate logic is more expressive than propositional logic and is often referred to as first-order logic. Predicate logic is essential to many areas in Computer Science, such as Formal Methods (proving correctness of software). See the book* for further examples and details.

*All definitions are from *Discrete Mathematics and Its Applications*, by Kenneth H. Rosen.