Predicate Logic

Carol Zander

Propositions cannot adequately express complex statements in mathematics and human language. Predicate logic gives us the power to express a wide variety of statements. It is the foundation used in the field of expert systems. A predicate describes a property of items, or a relationship among items.

Predicate -- A generalization of a propositional variable. Compared with propositions, predicates may have several variables; they are essentially functions with arguments.

For example,

```
person(x) says x is a person
female(x) says x is a female
male(x) says x is a male
genderUnspecified(x) says x is gender nonconforming
likes(x,y) says x likes y
gives(x,y,z) says x gives y to z
```

The statement "Susan gives the computer to Robert" can be expressed using the constants Susan, Computer, and Robert: `gives(Susan, Computer, Robert)`

Similarly, `gives(Andre, Book, Martin)` means "Andre gives the book to Martin" and `likes(John, Mary)` means "John likes Mary."

Relationships between predicates are created using operators. For example,

```
person(x) → male(x) OR female(x) OR genderUnspecified(x)
```

is read

```
if x is a person, then x is either a male or a female or is an unspecified gender
```

Technically there is universe of discourse, a domain where the `x` values come from. (My convention: Variables are lowercase. There are constants such as strings and numbers. A word starting with uppercase designates a string constant.)

Two quantifiers are commonly used:
Universal quantifier (notation is upside-down A, ∀) -- means for all, for every, for each
Existential quantifier (notation is backwards E, ∃) -- means there exists

If I don't have access to the upside-down A symbol, the word "all" or "for all" is used; to represent the backwards E, the word "exists" ot "there exists" may be used.

For example, the sentence `"John likes everybody who likes Mary"` is expressed as

```
(∀x)(likes(x,Mary) → likes(John,x))
```

This can be read as `for all x, if x like Mary, then John likes x.`
The `x` here represents the "everybody." The concept of *bound* connects the quantifier and variable; the variable `x` Is bound to the universal quantifier. The `x` used in the parenthetic expression is the same `x`, meaning the same item.

The sentence `"somebody who likes Mary likes John"` is expressed as

```
(∃x) (likes(x,Mary) and likes(x,John))
```

This can be read as `there exists an x where x likes Mary and that x likes John`

Problem from the Rosen Discrete math text – Express in terms of quantifiers and logical connectives

```
Domain consists of all students at your school.
Russian(x) means x can speak Russian
C++(x) means x knows C++
```

(a) There is a student at your school who can speak Russian and who knows C++.

```
    We assume that this sentence asserts that the same person has both talents:
          (∃x) (Russian(x) and C++(x))
```

(b) There is a student at your school who can speak Russian but who doesn't know C++.

```
    Note "but" really means "and" here:  (∃x)( Russian(x) and ~C++(x))
```

(c) Every student at your school either can speak Russian or knows C++.

```
    This time it is a universal statement:  (∀x)( Russian(x) or C++(x))
```

(d) No student at your school can speak Russian or knows C++.

```
    This sentence is asserting the nonexistence of anyone with either talent:
            (~∃x)( Russian(x) or C++(x))

    Alternatively, we can think that everyone fails to have either of these
    talents, so we obtain the equivalent (by De Morgan's law):
            (∀x)((~Russian(x) and (~C++(x)))
```

Problem from the Rosen Discrete math text - Express in terms of predicates, quantifiers, and logical connectives

(a) When there is less than 30mb free on the hard disk, a warning message is sent to all users.

```
    free(x) – There is less than x mb free on the hard disk
    warning(x) – user x is sent warning message

    free(30) → (∀x) warning(x)
```

(b) No directories in the file system can be opened and no files can be closed when system errors have been detected.

```
    open(x) -- directory x can be opened
    closed(x) -- file x can be closed
    error(x) --  system error x has been detected

    (∃x)(error(x)) → ((∀x)(~open(x)) and (∀x)(~closed(x)))
```

Negation

Consider css(x) which says x is a css student.

```
(∀x) css(x) – every student is a css student
```

Now consider the negation: It is not the case that every student is a css student.

```
(~∀x) css(x)
```

Or you could equivalently say that there is at least one student who is not a css student.

```
(∃x) (~css(x))
```

In general, `(~∀x)(p(x))` is equivalent to `(∃x)(~p(x))`
And `(~∃x)(p(x))` is equivalent to `(∀x)(~p(x))`
These are called De Morgan's laws for quantifiers.

Problem from the Rosen Discrete math text – Express using quantifiers and form the negation of the statement. Express the negation in simple English. (Do not use the words "It is not the case that.")

(a) All dogs have fleas.

```
flea(x) means x has fleas with domain of dogs.
Original statement:
        (∀x)(flea(x)
Negation:
        (∃x)(~flea(x))
In English:  There is some dog that does not have fleas.
```

(b) There is a horse that can add.

```
add(x) means x can add with domain of horses.
Original statement:
        (∃x)(add(x))
Negation:
        (∀x)(~add(x))
In English:  No horse can add.
```

Nested quantifiers

Nested quantifiers express more complex ideas. The order of the quantifiers matters. For example,

```
(∀x) (∃y) likes(x,y)    everybody likes somebody
(∃x) (∀y) likes(x,y)    somebody likes everybody
```

A complex thought based on an old saying can be expressed "you can fool everybody some of the time (except me), and you can fool somebody all of the time, but you can't fool me any of the time" is expressed as

```
(∀x) (∃t) (fool(x,t) and ~equals(x,Zander))
```

for all x, there exists a time t so that x, but not Zander, can be fooled at time t

```
and (∃x) (∀t)(fool(x,t)) and (∀t)(~fool(Zander,t))
```

and there exists an x, for all t so that x is fooled at time t, but for all the times t, Zander isn't fooled.