Logic

March 29, 2021

0.1 Notes

1. The notebooks are largely self-contained, i.e, if you see a symbol there will be an explanation about it at some point in the notebook.
    - Most often there will be links to the cell where the symbols are explained
    - If the symbols are not explained in this notebook, a reference to the appropriate notebook will be provided
2. **Github does a poor job of rendering this notebook**. The online render of this notebook is missing links, symbols, and notations are badly formatted. It is advised that you clone a local copy (or download the notebook) and open it locally.

1 Contents

1.1 Importing Libraries

[1]:
```python
import random
import math
```

Boolean Operations

And

The Boolean And operation is denoted using $\wedge$

$$p \wedge q$$

[2]:
```python
p = bool()
q = bool()

p and q
```

[2]: False

Or

The Boolean Or operation is denoted using $\vee$

$$p \vee q$$

[3]:
```python
p = bool()
q = bool()

p or q
```

[3]: False

Not

The Boolean Not operation is denoted using $\sim$ or $\neg$ and sometimes just the text 'not' is used:

$$\sim p$$

or

$$\neg p$$

or

$$\text{not } p$$

[4]:
```python
p = bool()

not p
```

[4]:
```
True
```

Exclusive Or

The Exclusive Or operation is denoted using $\veebar$ or $\oplus$ or just XOR

$$p \veebar q$$

or

$$p \oplus q$$

or

$$p \text{ XOR q}$$

[5]:
```python
p = bool()
q = bool()

#Shorter code since bool Xor equivalent to bitwise Xor
XOR_short = p ^ q

#Longer code by definition
XOR_long = (not q and p) or (not p and q)

XOR_short, XOR_long
```

[5]:
```
(False, False)
```

Nand

The Nand operation is denoted using $\barwedge$

$$p \barwedge q$$

The Nand operator can be expanded to: $p \barwedge q = \neg(p \vee q)$

[6]:
```python
p = bool()
q = bool()

not (p and q)
```

[6]:
```
True
```

Proof Symbols

Implies

In mathematical proofs, the term 'implies' means 'if a then b' or 'a implies b' and this is denoted using the symbol: $\Rightarrow$

$$a \Rightarrow b$$

Here a and b are any mathematical concepts (or *logical predicates*)

Logically, $a \Rightarrow b$ is equivalent to $b \vee \neg a$

(See: Or)

[7]:
```
boo = [True, False]
print('a implies b truth table\n')
for a in boo:
    for b in boo:
        print('a: ', a, 'b: ',b,', a implies b :', b or (not a))
```

```
a implies b truth table

a:  True b:  True , a implies b : True
a:  True b:  False , a implies b : False
a:  False b:  True , a implies b : True
a:  False b:  False , a implies b : True
```

To prove a theorem of this form, you must prove that b is true whenever a is true. Example: if x is greater than or equal to 4, then $2^x \geq x^2$

$$\forall x \in \mathbb{R},\ x \geq 4 \Rightarrow 2^x \geq x^2$$

i.e: if $a = (x \geq 4)$, then $b = (2^x \geq x^2)$

Notice from the above truth table that a may be False when b is True. However, b **must** be True when a is True. Hence, we can prove that b is True whenever a is True for $a \Rightarrow b$ but also, if we show that b is False when a is True, we have invalidated the $a \Rightarrow b$ statement.

(See: For all)

For more information on the real set and the belongs to notation see the Collections notebook

[8]:
```
print('if a then b')

X   = [-10.00,-2.20,0.00,2.00,3.10,4.00,5.5,6.00,7.8] #Subset of R used as an␣
↪example

a_implies_b =[]
```

```
for x in X:
    condition_a = x >= 4
    condition_b = 2**x >= x**2
    print('x :', x, ', x>=4, a: ',condition_a,', 2^x>=x^2, b: ', condition_b)

    a_implies_b.append(condition_b or (not condition_a))

print('\nCompared with the truth table above')
print('a implies b: ', all(a_implies_b))
#Atleast for this subset of R
```

```
if a then b
x : -10.0 , x>=4, a:  False , 2^x>=x^2, b:  False
x : -2.2 , x>=4, a:  False , 2^x>=x^2, b:  False
x : 0.0 , x>=4, a:  False , 2^x>=x^2, b:  True
x : 2.0 , x>=4, a:  False , 2^x>=x^2, b:  True
x : 3.1 , x>=4, a:  False , 2^x>=x^2, b:  False
x : 4.0 , x>=4, a:  True , 2^x>=x^2, b:  True
x : 5.5 , x>=4, a:  True , 2^x>=x^2, b:  True
x : 6.0 , x>=4, a:  True , 2^x>=x^2, b:  True
x : 7.8 , x>=4, a:  True , 2^x>=x^2, b:  True

Compared with the truth table above
a implies b:  True
```

Implied by

In mathematical proofs, the term 'implied by' means 'if b then a' or 'a is implied by b' and this is denoted using the symbol: $\Leftarrow$

$$a \Leftarrow b$$

Here a and b are any mathematical concepts (or *logical predicates*)

Logically, $a \Leftarrow b$ is equivalent to $a \vee \neg b$

(See: Or)

[9]:
```
boo = [True, False]
print('a implied by b truth table\n')
for a in boo:
    for b in boo:
        print('a: ', a, 'b: ',b,', a implied by b :', a or (not b))
```

```
a implied by b truth table

a:  True b:  True , a implied by b : True
a:  True b:  False , a implied by b : True
```

```
a:  False b:  True , a implied by b : False
a:  False b:  False , a implied by b : True
```

To prove a theorem of this form, you must prove that a true whenever b is true. To explain the concept, lets expand on the previous example, but here let's assume that the opposite condition is true, i.e: $x \geq 4$ is implied by $2^x \geq x^2$

$$\forall x \in \mathbb{R},\ x \geq 4 \Leftarrow 2^x \geq x^2$$

So based on our truth table above we have: $a = (x \geq 4), b = (2^x \geq x^2)$. Interestingly, if this is true we would have proved that $x \geq 4$ if and only if $2^x \geq x^2$ (See: if and only if)

(See: For all)

For more information on the real set and the belongs to notation see the Collections notebook

Now based on the truth table above if we observe a is False when b is True we have essentially disproved the implied by assertion:

[10]:
```python
print('if b then a')

X   = [0.00,2.00] #Subset of R used as an example

a_implied_by_b =[]

for x in X:
    condition_a = x >= 4
    condition_b = 2**x >= x**2
    print('x :', x,', 2^x>=x^2, b: ', condition_b, ', x>=4, a: ',condition_a,)

    a_implied_by_b.append(condition_a or (not condition_b))

print('\nCompared with the truth table above')
print('a implied by b: ',all(a_implied_by_b))
```

```
if b then a
x : 0.0 , 2^x>=x^2, b:  True , x>=4, a:  False
x : 2.0 , 2^x>=x^2, b:  True , x>=4, a:  False

Compared with the truth table above
a implied by b:  False
```

If and only if

In mathematical proofs, the term 'if and only if' means 'if a then b' as well as 'if b then a' and this is denoted using the symbol: $\iff$ but it is also sometimes abbreviated as: iff

$$a \iff b$$

or

$$a \text{ iff } b$$

The concept of iff is also logically equivalent to $(a \Rightarrow b) \wedge (b \Rightarrow a)$

(See: And and Implies)

Here a and b are any mathematical concepts (or *logical predicates*).

[11]:
```
boo = [True, False]
print('a iff b truth table\n')
for a in boo:
    for b in boo:
        print('a: ', a, 'b: ',b,', a iff b :', (b or (not a)) and (a or (not␣
↪b)))
```

```
a iff b truth table

a:  True b:  True , a iff b : True
a:  True b:  False , a iff b : False
a:  False b:  True , a iff b : False
a:  False b:  False , a iff b : True
```

To prove a theorem of this form, you must prove that a and b are equivalent. Not only is b true whenever a is true, but a is true whenever b is true. Example: The integer n is odd if and only if n^2 is odd.

$$\forall n \in \mathbb{Z}, \; n \text{ is odd} \iff n^2 \text{ is odd}$$

i.e: $(a = n \text{ is odd})$ iff $(b = n^2 \text{ is odd})$

(See: For all)

For more information on the integer set and the belongs to notation see the Collections notebook

[12]:
```
def is_odd(x):
    """returns whether x is Odd"""
    return not (x%2 == 0)

a_iff_b = []

print('if a then b')
N = [random.randint(-1000,1000) for i in range(5)] #Subset of Z used as an␣
↪example

for n in N:
    condition_a = is_odd(n)
    condition_b = is_odd(n**2)
    print('n: ', n, ', odd(n), a:',condition_a,
          '\nn2: ',n**2,', odd(n2), b:', condition_b,'\n')
```

```
    a_iff_b.append((condition_b or (not condition_a)) and (condition_a or (not␣
↪condition_b)))


print('if b then a')
N_new = [random.randint(-1000,1000)**2 for i in range(5)] #Subset of Z used as␣
↪an example

for n_squared in N_new:
    n = int(math.sqrt(n_squared))
    condition_b = is_odd(n_squared)
    condition_a = all([is_odd(n), is_odd(-n)])

    print('n2: ',n_squared,', odd(n2), a:', condition_b,
          '\nn: ', n, ', odd(n), b: ', condition_a, '\n')

    a_iff_b.append((condition_b or (not condition_a)) and (condition_a or (not␣
↪condition_b)))

print('\nCompared with the truth table above')
print('a iff b: ',all(a_iff_b))
```

```
if a then b
n:  505 , odd(n), a: True
n2:  255025 , odd(n2), b: True

n:  -449 , odd(n), a: True
n2:  201601 , odd(n2), b: True

n:  117 , odd(n), a: True
n2:  13689 , odd(n2), b: True

n:  -33 , odd(n), a: True
n2:  1089 , odd(n2), b: True

n:  -690 , odd(n), a: False
n2:  476100 , odd(n2), b: False

if b then a
n2:  14884 , odd(n2), a: False
n:  122 , odd(n), b:  False

n2:  67081 , odd(n2), a: True
n:  259 , odd(n), b:  True

n2:  82944 , odd(n2), a: False
n:  288 , odd(n), b:  False
```

```
n2:  35721 , odd(n2), a: True
n:  189 , odd(n), b:  True

n2:  123904 , odd(n2), a: False
n:  352 , odd(n), b:  False


Compared with the truth table above
a iff b:  True
```

Therefore

The term *therefore* is denoted by: $\therefore$

$$r^2 + \lambda^2 c^2 = 0$$

$$\therefore r = \pm\lambda ci$$

Because

The term *because* is denoted by: $\because$

$$\because x + 1 = 10 \therefore x = 9$$

Contradiction

Contradiction in a proof is denoted by: $\Rightarrow\!\Leftarrow$

Used to show that the supposition was False

End of Proof

The end of a proof is show using the following notations or text:

Just a square box:

$$\square$$

a filled square box:

$$\blacksquare$$

or the text:

$$\text{QED}$$

Quantifiers

For all

Also called a universal quantifier. The 'for all' symbol is used simply to denote that a concept or relation (or *logical predicates*) is applied to every member of the domain. Denoted by ∀

For example: squares of all real numbers are positive or zero can be expressed through:

$$\forall x \in \mathbb{R}, x^2 \geq 0$$

Which can be read as, for all x belonging to the set of real numbers (essentially any real number), the square of x is always greater or equal to zero.

For more information on the real set and the belongs to notation see the Collections notebook

[13]:
```
trials = 5

for i in range(trials):
    x = random.uniform(-100000, 100000)**2
    print(x >= 0)
```

```
True
True
True
True
True
```

The for all ∀ notations can be extended to denote complex statements. For example the commutative property of addition can be denoted using:

$$\forall x \in \mathbb{R}, \forall y \in \mathbb{R}, x + y = y + x$$

There exists

Also called an existential quantifier. This symbol can be interpreted as 'there exists', 'there is at least one', or 'for some' and applied to a mathematical concept (or *logical predicates*); it is denoted by: ∃

For example: there exists at least one real number x whose square equals 2

$$\exists x \in \mathbb{R}, x^2 = 2$$

This can be read as there exists at least one x belonging to the real set of numbers such that $x^2 = 2$. With this symbol, an assertion is being made about an object's existence which fulfills a criteria, which is true in this case since both $x = +\sqrt{2}$ and $x = -\sqrt{2}$ satisfy this condition.

Sometimes for readability, some authors will use the abbreviation for **such that** (*s.t.*) :

$$\exists x \in \mathbb{R} \text{ s.t. } x^2 = 2$$

For more information on the real set and the belongs to notation see the Collections notebook

[14]:
```python
x_square = 2
x_1 = math.sqrt(2)
x_2 = -math.sqrt(2)

type(x_1) == float, type(x_2) == float

#There may be more x's but we've shown enough to prove this statement to be true
```

[14]:
```
(True, True)
```

There exists uniquely

When the existential quantifier symbol is followed by an exclamation point, it means there exists a **unique** object that fulfills a given criteria: $\exists!$

For example: there exists a unique real number x whose square equals 0

$$\exists! x \in \mathbb{R}, x^2 = 0$$

which is true in this case since only $x = 0$ satisfies this condition.

For more information on the real set and the belongs to notation see the Collections notebook

[15]:
```python
#range(-5,5) is shown as an example, in reality you would have to look at each␣
↪element
#in the infinite real set to prove uniqueness
for x in range(-5,5):
    print('x: ',(x/10),', x2 == 0', (x/10)**2 == 0)
```

```
x:  -0.5 , x2 == 0 False
x:  -0.4 , x2 == 0 False
x:  -0.3 , x2 == 0 False
x:  -0.2 , x2 == 0 False
x:  -0.1 , x2 == 0 False
x:  0.0 , x2 == 0 True
x:  0.1 , x2 == 0 False
x:  0.2 , x2 == 0 False
x:  0.3 , x2 == 0 False
x:  0.4 , x2 == 0 False
```

Combining quantifiers

The for all $\forall$ and exists $\exists$ notations can be combined to denote complex statements.

For example: For all x in the real number set, there exists at least one real number y such that $x + y = 0$

$$\forall x \in \mathbb{R}, \exists y \in \mathbb{R}, x + y = 0$$

This statement means that if we were to pick **any** real number x, we can find at least one number y so that we get $x + y = 0$. We know this to be a True statement since we can always find a number $y = -x$

For more information on the real set and the belongs to notation see the Collections notebook

[16]:
```
R_sample = [random.random() for i in range(10)] #checking for a small subset of␣
↪the real number
#set as an example. In reality we would have to loop over the entire infinite␣
↪Real number set
#to check for all x

condition = []

for x in R_sample:
    y = -x  #this y will always have the property 'x + y = 0'; for any x in R
    condition.append(x+ y == 0)

all(condition) #Check if our condition was tested as True for ALL x in R
```

[16]: True

Note: The statement order is very important since it evolves logically and combines to form a logical assertion. The above example was well ordered but consider the following example:

$$\exists y \in \mathbb{R}, \forall x \in \mathbb{R}, x + y = 0$$

In this case we make an assertion that there exists a real number y which will have the property $x + y = 0$ for any real number x. Such a real number y does not exist and so this assertion is **False**.

[17]:
```
R_sample = [random.random() for i in range(10)] #checking for a small subset of␣
↪the real number
#set as an example. In reality we would have to loop over the entire infinite␣
↪Real number set
#to check for all y

all_y_conditions = []

for y in R_sample: #Searching for the 'magic' y
    condition = all([ x + y == 0 for x in R_sample]) #The magic y has the␣
↪property 'x + y = 0' for any x
    all_y_conditions.append(condition)

any(all_y_conditions) #Check if we found at least one 'magic' y ?
```

[17]: False