Russell on Logic

"logic may be defined as . . . the study of what can be said of everything"

Propositional (Boolean) Sentences

In writing programs, one uses boolean or propositional logic variables, p, q, etc. which have values depending on the state of the variables. Propositional (or boolean) variables have 2 possible values, T (*True*) or F (*False*). In Digital Logic, the values 1 (*True*) and 0 (*False*) are used. Complex propositional sentences are built up by combining simpler sentences using logic operators.
Propositional sentences can be referred to as 'sentences'.
Propositional Logic is also referred to as Sentential Logic.
In some Logic texts sentences are called 'Well Formed Formulas (wff)' or 'formulas' if it is understood that they are well formed.

Logical Operators

The logical operators include: **not**, **and**, **or**

operator	not	and	or
symbol	$\neg$	$\wedge$	$\vee$

Mnemonic:
Use $\wedge$ for $\wedge$**nd**.
The Latin for "or" is "vel" and so use $\vee$ for $\vee$**el** .
The 'and' operator $\wedge$ is analogous to the Set Theory operator $\cap$,
the 'or' operator $\vee$ is analogous to the Set Theory operator $\cup$.
Other logical operators will be defined later.

Constructing Propositional Sentences

Similar to constructing arithmetic or boolean expressions in programming languages, complex sentences are constructed from simpler sentences.

- *True*, *False* are (constant) sentences
- A propositional variable, p, q, r etc. is a sentence.
  A sentence that is a propositional variable is also called an 'atomic sentence'.
- If P is a sentence then so is $\neg P$
- If P and Q are sentences the so are:
  $(P \wedge Q)$ and $(P \vee Q)$.

Example:
When p, q, r are propositional variables then

$$(p \vee q),\ (\neg p \vee q),\ ((p \wedge q) \vee r)$$

are three sentences.

Form of a sentence

The sentence:

$$\neg(p \wedge \neg(\neg q \vee r))$$

has the form $\neg P$ where
P is the sentence $(p \wedge \neg(\neg q \vee r))$.
The sentence $(p \wedge \neg(\neg q \vee r))$ has the form $(P \wedge Q)$ where
P is the propositional variable (atomic sentence), p, and Q is the sentence $\neg(\neg q \vee r)$.
The sentence Q in turn can be broken down into sentences leading eventually to 'atomic sentences' which are the propositional variables.

Construction Tree

We can create a 'Construction Tree' that better expresses the construction of the sentence $\neg(p \land \neg(\neg q \lor r))$ from its subparts:

$$\dfrac{\dfrac{p \qquad \dfrac{\dfrac{\dfrac{q}{\neg q} \qquad r}{(\neg q \lor r)}}{\neg(\neg q \lor r)}}{(p \land \neg(\neg q \lor r))}}{\neg(p \land \neg(\neg q \lor r))}$$

Evaluating a sentence in a state using Truth Table

Evaluating a sentence in a state

Before a propositional sentence can be evaluated it needs to be known how to evaluate sentences with the basic logic operators. This is done using **Truth Tables**. The constant, *True* evaluates to the boolean value, T, and the constant, *False*, to the boolean value, F. In Digital Logic, the boolean value, T, may be represented as, 1, and the boolean value, F, as, 0. To evaluate the logical operators in any state use the truth tables for the operators:

p	q	$p \wedge q$	$p \vee q$
F	F	F	F
F	T	F	T
T	F	F	T
T	T	T	T

p	$\neg p$
F	T
T	F

Digital Logic Truth Table

In Digital Logic, the truth tables for the operators, $\wedge$, $\vee$ and $\neg$ may be presented as:

p	q	$p \wedge q$	$p \vee q$
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

p	$\neg p$
0	1
1	0

Alternate Table

Alternate Table

$\wedge$	F	T
F	F	F
T	F	T

$\vee$	F	T
F	F	T
T	T	T

Evaluating in a State

Given a sentence, $\neg p \vee q$ and a state, s, represented by:

$$s = \begin{array}{c|c|c} \textbf{var} & p & q \\ \hline \textbf{val} & T & F \end{array}$$

then evaluate $\neg p \vee q$ in the state, s, as
$\neg T \vee F$
$= F \vee F$
$= F$

This is similar to evaluating a boolean expression in Java.

Evaluate a sentence in a state

Evaluate the sentence

$$\neg(p \wedge \neg(\neg q \vee r))$$

in the state

$$s = \begin{array}{c|c|c|c} \textbf{var} & p & q & r \\ \hline \textbf{val} & T & F & F \end{array}$$

Cont'd

In the state, s, the value of p is T, the value of q is F and the value of r is F. To evaluate $\neg(p \wedge \neg(\neg q \vee r)$ in state, s, evaluate:

$$\neg(T \wedge \neg(\neg F \vee F))$$

$\neg(T \wedge \neg(\neg F \vee F))$
$= \neg(T \wedge \neg(T \vee F))$
$= \neg(T \wedge \neg T)$
$= \neg(T \wedge F)$
$= \neg F$
$= T$

In Java notation,
$\neg(T \wedge \neg(\neg F \vee F))$ is written as $!(true \,\&\&\, !(!false \,||\, false))$

Evaluation a sentence is a state

To evaluate a sentence in a state, s, use the form of the sentence. If the sentence is

- a constant, then in any state, *True* evaluates to T and *False* evaluates to F.
- a propositonal variable (atomic sentence), e.g. the variable, p, then find the value of p in the state, s.
- of the form $\neg P$ then evaluate P in state, s, and negate this value.
- of the form $P \wedge Q$ then evaluate both P and Q in state, s, and using the truth table for the operator, $\wedge$, find the value of $P \wedge Q$.
- of the form $P \vee Q$ then evaluate both P and Q in state, s, and using the truth table for the operator, $\vee$, find the value of $P \vee Q$.

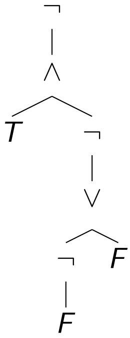
Evaluation Tree

To evaluate a sentence, the form of an sentence can be be represented as an 'evaluation tree' or an 'expression tree'. Consider the evaluation/expression tree for the sentence $\neg(p \wedge \neg(\neg q \vee r))$ In the the state, s, where

$$s = \begin{array}{c|c|c|c} \textbf{var} & p & q & r \\ \hline \textbf{val} & T & F & F \end{array}$$

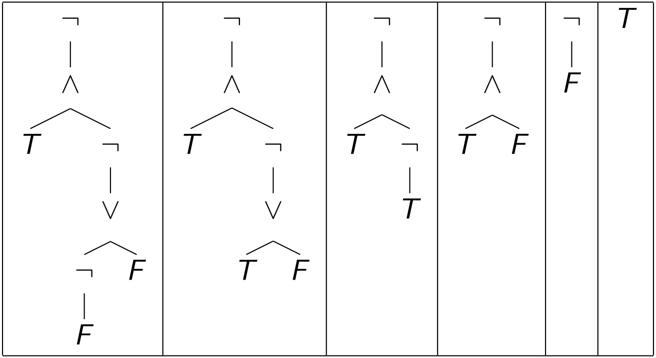
Evaluation Tree (cont'd)

The evaluation tree for the sentence $\neg(p \wedge \neg(\neg q \vee r))$ in state, s, i.e. the evaluation tree for $\neg(T \wedge \neg(\neg F \vee F))$ is:



Evaluate an evaluation tree

Evaluate the evaluation tree of $\neg(T \wedge \neg(\neg F \vee F))$ from 'bottom up'



Cont'd

The evaluation of the evaluation/expression tree corresponds to the evaluation:

$$
\begin{aligned}
&\neg(T \wedge \neg(\neg F \vee F)) \\
&= \neg(T \wedge \neg(T \vee F)) \\
&= \neg(T \wedge \neg T) \\
&= \neg(T \wedge F) \\
&= \neg F \\
&= T
\end{aligned}
$$

Meaning of Logic Operators

- Negation, $\neg$

Consider the mathematical proposition $\pi > 3$ where π is the mathematical constant that is the ratio of the circumference, C, to the diameter, D of a circle,
i.e. $\pi = \frac{C}{D}$.
The negation, $\neg(\pi > 3)$ is $\pi \leq 3$.
Also, $z \in \overline{X} \equiv \neg(z \in X)$. $z \notin X$ is an abbreviation for $\neg(z \in X)$
Consider the sentence P: "All soccer fans are well behaved"
Is the negation, $\neg P$

a: All soccer fans are badly behaved.

b: All non soccer fans are well behaved.

c: Some soccer fans are well behaved.

d: Some soccer fans are badly behaved.

And

- And, $\wedge$ *'Conjunction'*

In Mathematics and in Logic , $P \wedge Q = Q \wedge P$ and so the operator $\wedge$ is *commutative*.
In English, some uses of 'and' may involve time or causality.
e.g. Let P: "I put on my socks" Q: "I put on my shoes"
e.g. Let P: "Messi crossed the ball"
and Q: "Suárez headed a goal"
In these English cases, *P and Q* has not the same meaning as *Q and P*.

The mathematical expression, $x < y < z$ is an abbreviation of $x < y \wedge y < z$. Also,

- $z \in X \cap Y \equiv z \in X \wedge z \in Y$.
- $z \in X - Y \equiv z \in X \wedge z \notin Y$. i.e. $z \in X - Y \equiv z \in X \cap \overline{Y}$.

And

In Java, the operator, &&, is used in a conditional way and so not strictly commutative.

```
if ( b != 0  &&  a/b > 0 )
    print(a/b);
```

This is the same as:

```
if ( b != 0 )
    if ( a/b > 0 )
        print(a/b);
```

If `b=0` then the second conjunct, `a/b > 0`, is not evaluated.

Or

- Or, $\vee$ *'Disjunction'*

In logic, $\vee$, is used in the 'inclusive' sense,
i.e. $P \vee Q$ means P or Q or both.
e.g. if $x * y = 0$ then $x = 0 \vee y = 0$. Both x and y may be 0.
e.g. If $(x - 2) * (x - 3) = 0$ then $x = 2 \vee x = 3$. In this case it is not possible for both $x = 2$ and $x = 3$ to be true but in general, $P \vee Q$ allows for the possibility of both P and Q being true.
In mathematics, $x \leq y$ is an abbreviation for $x < y \vee x = y$.
The negation, $\neg(x \leq y)$ is $x > y$.

Or

Case Analysis

A proof may be of the form: Assume $P \vee Q$, show R.

To show R, use case analysis.

Case P:

Assume P, show R.

Case Q:

Assume Q, show R.

Or

In Java, the boolean operator, $||$, is used in a conditional way i.e. in an expression $P \,||\, Q$, if P is true then Q is not evaluated as $true \,||\, Q = true$.

Consider

```
if ( b == 0 || a/b > 0 )
    print(...);
```

This is the same as:

```
if ( b == 0 )
    print(...);
else if ( a/b > 0 )
    print(...);
```

Precedence of logical operators

Precedence of Logical Operators

Precedence rules avoid having to always use brackets. In Arithmetic $2 + 3 * 4$ is an abbreviation of $(2 + (3 * 4))$.
In Propositional Logic, the precedence rules are given as:

- expressions that involve the same operators $\vee$ or $\wedge$ are evaluated from left to right.
  $p \vee q \vee r$ is an abbreviation of $((p \vee q) \vee r)$.

Evaluation order

- The order of evaluation of different operators is:
  The operator $\neg$ is evaluated before the operators $\wedge$ and $\vee$
  i.e. $\neg$ has a higher precedence than either $\wedge$ or $\vee$,
  In Logic, the operators $\wedge$ and $\vee$ have the same precedence and so brackets should be used.

Precedence conventions

Since $\wedge$ and $\vee$ have the same precedence brackets are used to clarify the order of evaluation $\therefore$ use $p \vee (q \wedge r)$ instead of $p \ \vee \ q \wedge r$.
The convention that $\wedge$ and $\vee$ have the same precedence is not agreed by all logicians as some logic texts (and Digital Logic) give $\wedge$ a higher precedence than $\vee$ using the arithmetic analogy of $\wedge$ corrresponding to arithmetic multiplication, *, and $\vee$ corresponding to addition, $+$.
This can be misleading as in Logic $\vee$ distributes over $\wedge$
i.e. in Logic $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$ but generally in Arithmetic
$p + (q * r) \neq (p + q) * (p + r)$.
Also, in Logic $p \vee T = T$ but in Arithmetic (and also Modular Arithmetic) $p + 1 \neq 1$. The operator $\vee$ does not correspond to Arithmetic, $+$.

- If in doubt use brackets to clarify evaluation order.

Constant propositions: *True* and *False*

The constant proposition, *True*, is equivalent to the always true sentence, $p \vee \neg p$ as in any state, $p \vee \neg p$ evaluates to T and the constant proposition, *False*, is equivalent to the always false sentence, $p \wedge \neg p$

Negation property: $\neg \mathit{True} = \mathit{False}$ and $\neg \mathit{False} = \mathit{True}$.

Identity for $\wedge$ and $\vee$

The constant proposition, *True*, is the *Identity* for $\wedge$ i.e. for any proposition, p, $p \wedge \mathit{True} = p$ and $\mathit{True} \wedge p = p$.
The constant proposition, *False*, is the *Identity* for $\vee$ i.e. for any proposition, p, $p \vee \mathit{False} = p$ and $\mathit{False} \vee p = p$.
In Arithmetic, the *Identity* for $+$ is 0 and the *Identity* for $*$ is 1.
Note:
$p \wedge \mathit{False} = \mathit{False} \wedge p = \mathit{False}$ and $p \vee \mathit{True} = \mathit{True} \vee p = \mathit{True}$.

Alternative notation: $\top$ (*True*) and $\perp$ (*False*)

It is convenient to use the symbol, $\top$, for the constant *True* and the symbol, $\perp$, for the constant *False*, $\therefore$

- $p \wedge \top = p$ and $\top \wedge p = p$. Also, $p \wedge \perp = \perp \wedge p = \perp$.
- $p \vee \perp = p$ and $\perp \vee p = p$. Also $p \vee \top = \top \vee p = \top$.

Truth Table for Propositional Sentence/Function

A Truth Table can be constructed for any sentence. A propositional sentence can represent a propositional function in the variables that are used in the sentence.
Consider the example, $(p \wedge q) \vee (\neg p \wedge r)$, with 3 variables $\therefore$ it represents a function in the 3 variables, p, q and r.
This is similar to a quadratic expression, $x^2 + 4 * x + 2$, which defines a function in the one variable, x.

Truth Table

The truth table for a 3 variable propositional function requires 8 rows. A $n-$variable function requires 2^n rows.

row	p q r	$(p \wedge q) \vee (\neg p \wedge r)$
0	F F F	F
1	F F T	T
2	F T F	F
3	F T T	T
4	T F F	F
5	T F T	F
6	T T F	T
7	T T T	T

Implement any Propositional Function using $\neg$, $\wedge$ and $\vee$

Any propositional function can be implemented using just $\neg$, $\wedge$ and $\vee$.

Given any propositional function $b(p_1, p_2, \ldots p_n)$ in n variables, a truth table of 2^n rows gives the output value for all the possible 2^n inputs.

p_1 p_2 $\ldots$ p_n	$b(p_1, p_2, \ldots p_n)$
F F $\ldots F$	$b(F, F, \ldots F)$
$\vdots$	$\vdots$
T T $\ldots T$	$b(T, T, \ldots T)$

Consider the rows where the output is T.

Truth Table for Propositional Function

Consider an example with 3 variables and therefore an 8 row truth table is needed.

row	$p\ q\ r$	$b(p,q,r)$
0	$F\ F\ F$	F
1	$F\ F\ T$	T
2	$F\ T\ F$	F
3	$F\ T\ T$	T
4	$T\ F\ F$	F
5	$T\ F\ T$	F
6	$T\ T\ F$	T
7	$T\ T\ T$	T

Rows 1,3,6,7 return T .

Disjunctive Normal Form (DNF)

Disjunctive Normal Form (DNF)

Consider row 1: the value of the variables p, q, r are F, F, T. Associate with this row the **conjunction** $\neg p \wedge \neg q \wedge r$ as the conjunction has the output T when p, q, r are F, F, T.
Similarly,
asociate with row 3 (F, T, T) the conjunction $\neg p \wedge q \wedge r$
asociate with row 6 (T, T, F) the conjunction $p \wedge q \wedge \neg r$
asociate with row 7 (T, T, T) the conjunction $p \wedge q \wedge r$
Form the **Disjunction**:
$(\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r) \vee (p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r)$.

Disjunctive Normal Form (DNF) [Cont'd]

This disjunction has the same truth table as $b(p, q, r)$ and hence:
$b(p, q, r) = (\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r) \vee (p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r)$
This particular implemenation of $b(p, q, r)$ is in **Disjunctive Normal Form** i.e. $b(p, q, r)$ is expressed as a disjunction of conjunctions where each conjunction is the 'anding' of propositional variables or their negation.
The DNF form may not be the simplest way to implement a function as for example, the function
$b(p, q, r) = (p \wedge q) \vee (\neg p \wedge r)$. In particular, the sentence

$$(\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r) \vee (p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r)$$

can be simplified to

$$(p \wedge q) \vee (\neg p \wedge r)$$

Simplification calculation

Simplify $(\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r) \vee (p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r)$

$(\neg p \wedge \neg q \wedge r) \vee (\neg p \wedge q \wedge r) \vee (p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r)$
{Distributivity: $(a \wedge b) \vee (a \wedge c) = a \wedge (b \vee c)$ }
$= ((\neg p \wedge r) \wedge (\neg q \vee q)) \vee (p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r)$
{ $\neg a \vee a = \top$ and $a \wedge \top = a$ }
$= (\neg p \wedge r) \vee (p \wedge q \wedge \neg r) \vee (p \wedge q \wedge r)$
{ Distributivity: 'take out' $p \wedge q$ }
$= (\neg p \wedge r) \vee ((p \wedge q) \wedge (\neg r \vee r))$
{ $\neg a \vee a = \top$ and $a \wedge \top = a$ }
$= (\neg p \wedge r) \vee (p \wedge q)$
$= (p \wedge q) \vee (\neg p \wedge r)$

Expressively Complete set of operators

Since any propositional function can be implemented or expressed using the operators $\wedge, \vee, \neg$ the set of operators $\{\neg, \wedge, \vee\}$ is said to be **expressively complete** or **functionally complete.**
The operator $\vee$ can be implemented using $\wedge$ and $\neg$
i.e. $p \vee q = \neg(\neg p \wedge \neg q)$
as $p \vee q$ and $\neg(\neg p \wedge \neg q)$ have the same truth table.
Therefore, the set of operators $\{\wedge, \neg\}$ is also expressively complete.
Any propositional (boolean) function can be implemented using the operators in $\{\wedge, \neg\}$.
Also, the operator, $\wedge$ can be implemented using $\vee$ i.e.
$p \wedge q = \neg(\neg p \vee \neg q)$ and so the set of operators $\{\vee, \neg\}$ is functionally complete.

Sheffer Stroke (nand) operator

Consider a binary operator, $|$, which is named the Sheffer stroke or in modern terms the '*nand*' operator as it is defined as '*not and*' i.e. $p \,|\, q = \neg(p \wedge q)$.
The operator $\neg$ can be implemented using the *nand* operator by $\neg p = p \,|\, p$, since $p \,|\, p = \neg(p \wedge p) = \neg p$.
Also the operator, $\wedge$, can be implemented using the *nand* operator.

Sheffer Stroke (nand) operator (cont'd)

Since $p \mid q = \neg(p \wedge q) \therefore \neg(p \mid q) = \neg\neg(p \wedge q)$, $\therefore p \wedge q = \neg(p \mid q)$ but $\neg$ can be defined using the *nand* operator i.e. $\neg(p \mid q) = (p \mid q) \mid (p \mid q) \therefore p \wedge q = (p \mid q) \mid (p \mid q)$.
Since the set of operators $\{\wedge, \neg\}$ is expressively complete then so is the set $\{\mid\}$ since both $\wedge$ and $\neg$ can be expressed using just the *nand* operator.
Therefore, all propositional functions can be expressed using just the *nand* operator.
By De Morgan's Law, we also have $p \mid q = \neg p \vee \neg q$.

$P \to Q$

if-then operator $\to$

The operator, $\to$, is also referred to as the 'conditional operator'.
We can read $P \to Q$ as 'if P then Q'.
$P \to Q$ can be defined in terms of $\neg$ and $\vee$

$$P \to Q = \neg P \vee Q$$

or $P \to Q$ can be defined in terms of $\neg$ and $\wedge$

$$P \to Q = \neg(P \wedge \neg Q)$$

Negation in terms of $\rightarrow$

Negation in terms of $\rightarrow$

$$\neg P = P \rightarrow \bot$$

where $\bot$ is the constant for False i.e. $\bot$ is a contradiction.

Material Implication

In Propositional Logic, in a sentence of the form, $P \rightarrow Q$ there need not be a logical or causal connection between P and Q as the truth of $P \rightarrow Q$ depends only on the truth of P and the truth of Q, even when there is no causal or logical connection. For example, Let P: "it is raining" and let Q: "it is cold" then the sentence $P \rightarrow Q$ is true when it is not raining or when it is cold i.e $P \rightarrow Q$ is true when P is false or Q is true. In Logic, the operator, $\rightarrow$, is referred to as 'material implication'

$P \equiv Q$

The 'Equivalent' operator $\equiv$

The operator, $\equiv$, is also referred to as the 'bi-conditional operator'.
The operator $\equiv$ is similar to the operator $==$ in Java.
We can define $P \equiv Q$ in terms of $\wedge$ and $\rightarrow$

$$P \equiv Q \;=\; (P \rightarrow Q) \wedge (Q \rightarrow P).$$

Using the definition of $P \rightarrow Q = \neg P \vee Q$

$$P \equiv Q \;=\; (\neg P \vee Q) \wedge (\neg Q \vee P).$$

Truth Table for $\rightarrow$ and $\equiv$

p	q	$p \rightarrow q$	$p \equiv q$
F	F	T	T
F	T	T	F
T	F	F	F
T	T	T	T

From the Truth Table for $\equiv$, we can implement $p \equiv q$ in DNF as

$$P \equiv Q \;=\; (P \wedge Q) \vee (\neg P \wedge \neg Q)$$

$P \to Q$ and Logical Implication

$P \to Q$ is related to the mathematical 'P implies Q.'
i.e. if ($P \to Q$) is **proven** then (P implies Q).
In mathematics, the statement (P implies Q) indicates there is a logical connection between P and Q .
e.g. n is odd implies n^2 is odd,
i.e. if we assume n is odd then we can prove that n^2 is odd.

$P \to Q$ and Logical Implication

Because the operator $\to$ is related to 'logical implication' via the concept of proof, the truth table for $\to$ is expected to satisfy 3 'proof' properties.

- **Detachment Law**: i.e. from $P \to Q$ and P we can conclude Q i.e. if both $P \to Q$ and P are True then Q must be True.
- **Contraposition**: $P \to Q$ has same Truth Table as $\neg Q \to \neg P$ i.e. $P \to Q$ and $\neg Q \to \neg P$ are logically equal.
  e.g. "If n is odd then n^2 is odd" is logically equal to "If n^2 is not odd then n is not odd" i.e. "If n^2 is even then n is even".
  In other words,
  to prove "If n^2 is even then n is even"
  show "If n is odd then n^2 is odd" .
- **Converse not equal**: The converse of $P \to Q$ is $Q \to P$ and these should not have the same Truth Table, i.e. $P \to Q$ and its converse $Q \to P$ are not logically equal.

$P \to Q$ and Logical Implication

Consider the Truth Table for $P \to Q$ $\neg Q \to \neg P$ and $Q \to P$

P	Q	$P \to Q$	$\neg Q \to \neg P$	$Q \to P$
F	F	T	T	T
F	T	T	T	F
T	F	F	F	T
T	T	T	T	T

$P \rightarrow Q$ (cont'd)

The logical operator, $\rightarrow$, has a more precise meaning in Logic than in English.
Consider $x > 5 \rightarrow x > 3$, this is always true as it is a property of numbers
i.e. it is not possible for $x > 5$ and $x \leq 3$.
Consider particular values for x.

- Case when $x > 5$ is True and $x > 3$ is True:
  With $x = 6$, $6 > 5$ is true and $6 > 3$ is true and so also $6 > 5 \rightarrow 6 > 3$ is true, i.e.
  $T \rightarrow T = T$

$P \rightarrow Q$ (cont'd)

- Case when $x > 5$ is False and $x > 3$ is True:
  With $x = 4$, $4 > 5$ is False and $4 > 3$ is True. In this case, $4 > 5 \rightarrow 4 > 3$ is true, i.e. $F \rightarrow T = T$
- Case when $x > 5$ is False and $x > 3$ is False:
  With $x = 2$, $2 > 5$ is False and $2 > 3$ is False. In this case $2 > 3 \rightarrow 2 > 5$ is true,
  $F \rightarrow F = T$
- Case when $x > 5$ is True and $x > 3$ is False:
  it is not possible for $x > 5$ and $x \leq 3$.

$P \rightarrow Q$

Given that $P \rightarrow Q$ is True and also that P is True then Q is True.
i.e. from $P \rightarrow Q$ and P conclude Q
This is similar to:
If $P \vee Q$ is True and also that P is False (i.e. $\neg P$ is True) then conclude that Q is True.
i.e. from $P \vee Q$ and $\neg P$ conclude Q.
Replacing P with $\neg P$, we get
from $\neg P \vee Q$ and P conclude Q.
This is the same as
from $P \rightarrow Q$ and P conclude Q.

Definition

$P \rightarrow Q \equiv \neg P \vee Q$.
i.e. using De Morgan: $P \rightarrow Q \equiv \neg(P \wedge \neg Q)$.

Precedence $\rightarrow$

Precedence of $\rightarrow$

Sentences that involve the same operator $\rightarrow$ are evaluated from right to left i.e. the operator $\rightarrow$ associates to the right.

$$p \rightarrow q \rightarrow r \text{ is an abbreviation of } p \rightarrow (q \rightarrow r) \; .$$

Note:
In Arithmetic, exponentiation, x^y, is associatives to the right, i.e. x^{y^z} is read as $x^{(y^z)}$. Some programming languages use $x\hat{\ }y$ for x^y
$\therefore$ $x\hat{\ }y\hat{\ }z$ is read as $x\hat{\ }(y\hat{\ }z)$

Precedence $\equiv$

Precedence of $\equiv$

The operator, $\equiv$, associates to the left i.e.

$$p \equiv q \equiv r \text{ is an abbreviation of } ((p \equiv q) \equiv r) \ .$$

In Logic, the operator, $\rightarrow$ has a higher precedence than the operator, $\equiv$.
e.g. $p \rightarrow q \equiv \neg q \rightarrow \neg p$ abbreviates $((p \rightarrow q) \equiv (\neg q \rightarrow \neg p))$

Precedence Order

The order of evaluation of different operators is:
The operator $\neg$ is evaluated before the operators $\wedge$ and $\vee$
i.e. $\neg$ has a higher precedence than either $\wedge$ or $\vee$,
the operators $\wedge$ and $\vee$ have the same precedence and so brackets should be used. Next in order of precedence is $\rightarrow$ and then $\equiv$.
(highest to lowest precedence): $\neg$ higher than ($\wedge$ and $\vee$) higher than $\rightarrow$ higher than $\equiv$.

Examples:

$\neg p \equiv q \wedge r$ is an abbreviation of $(\neg p \equiv (q \wedge r))$
$p \rightarrow q \rightarrow r \vee s$ is an abbreviation of $(p \rightarrow (q \rightarrow (r \vee s)))$
$p \wedge q \equiv p \equiv q \equiv p \vee q$ is an abbreviation of
$((((p \wedge q) \equiv p) \equiv q) \equiv (p \vee q))$