

Propositional Logic

Symbols: \wedge - and; \vee - or; \neg not; \Rightarrow - implies; \Leftrightarrow is equivalent to
A is a proposition, $A \Rightarrow B$ is a proposition.

Truth Tables can be used to represent connectives and propositions using T and F to represent True and False.

E.g., the truth table for implication (\Rightarrow) is:

A	B	$A \Rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

and the truth table for P_3 would be similar to this:

Z	$\neg Z$	H	$\neg Z \Rightarrow H$
T	F	T	T
T	F	F	T
F	T	T	T
F	T	F	F

Note: the number of lines in a truth table is $2^{\text{Number of Inputs}}$.

Precedence:

- \neg binds the strongest, i.e. $\neg A \& B$ means $(\neg A) \& B$
- \wedge is stronger than \vee and \Rightarrow , i.e. $A \vee B \wedge C$ means $A \vee (B \wedge C)$
- \vee is stronger than \Rightarrow , i.e. $A \vee B \Rightarrow C$ means $(A \vee B) \Rightarrow C$
- \Rightarrow is right associative, i.e. $A \Rightarrow B \Rightarrow C$ means $A \Rightarrow (B \Rightarrow C)$

Puzzle:

- P_1 - If the sun shines, then we go to the zoo = $S \Rightarrow Z$
- P_2 - It either rains or the sun shines = $R \vee S$
- P_3 - If we do not go to the zoo, then we stay at home = $\neg Z \Rightarrow H$

Which of the following consequences are logical?

- C_1 - If we do not go to the zoo, then it rains ($\neg Z \Rightarrow R$)
- C_2 - If we do not stay at home, the sun shines ($\neg H \Rightarrow S$)
- C_3 - If the sun shines or we do not stay at home, then we go to the zoo ($S \vee \neg H \Rightarrow Z$)
- C_4 - If the sun does not shine, then we do not go to the zoo ($\neg S \Rightarrow \neg Z$)

If ALL the propositions imply the consequence, it is a logical consequence, i.e. it is a *tautology*. In practical terms, it is easiest to create a new proposition, P that represents all the propositions together, (i.e. $P_1 \wedge P_2 \wedge P_3 \wedge P_4$). Draw a truth table containing all the inputs (i.e. S, Z, R and H) and containing the output given by P, C_1 , C_2 , C_3 and C_4 if $P \Rightarrow C_x$, then C_x is a tautology. E.g.:

S	Z	R	H	P_1	P_2	P_3	P	C_1	$P \Rightarrow C_1$	C_2	$P \Rightarrow C_2$	C_3	$P \Rightarrow C_3$	C_4	$P \Rightarrow C_4$
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

If $P \Rightarrow C_x$, C_x is a tautology.

Defining Connectives

- \Leftrightarrow can be defined as $(A \Rightarrow B) \wedge (B \Rightarrow A)$
- \Rightarrow can be defined as $\neg A \vee B$
- \vee can be defined as $\neg(\neg A \wedge \neg B)$
- True can be defined as $\neg F$
- Hence, all connectives can be defined in terms of \wedge and \neg , which can be reduced to the single symbol, \wedge (nand) which is defined as $\neg(A \wedge B)$ as follows:
 - F is defined as $T \wedge T$
 - $\neg A$ is defined as $T \wedge A$
 - $A \wedge B$ is defined as $T \wedge (A \wedge B)$
 - $A \vee B$ is defined as $(A \wedge T) \wedge (B \wedge T)$
 - $A \Rightarrow B$ is defined as $A \wedge (B \wedge T)$

Using the 'Tutch' proof checker for Propositional Logic

In tutch, 'and' is represented by &, 'or' by |, 'not' by ~ and implies/equivalence by => / <=>
[means 'assume'.] signals the end of the assumption.

Rules:

- To prove an implication, assume the left-hand side and prove the right.
- To prove an equivalence, prove the implication in both directions
- If you know A or you know B, you know A | B.
- If you know A & B, you know A and you know B.
- To prove something from an 'or', prove it with each bit of the 'or'.
- To prove an 'or', prove for each bit of the 'or' – 'or' breakdown
- To prove 'not' something, assume the opposite and prove it false.

Example:

1	proof distrib: A & (B C) => (A & B) (A & C) =	state the proposition
2	begin	beginning of the proof
3	[A & (B C);	assume the left hand side...
4	A;	know A & (B C) so know A
5	[B;	'or' breakdown (left hand side)
6	A & B;	know A from 4 and B from 5, so know A & B
7	(A & B) (A & C);	know A & B from 6, so know A&B anything
8	[C;	'or' breakdown (right hand side)
9	A & C;	know A from 4 and C from 8, so know A & C
10	(A & B) (A & C);	know A&C from 10, so know A&C anything
11	(A & B) (A & C);	...prove the right hand side
12	A & (B C) => (A & B) (A & C);	end result

Running this proof in Tutch gives the following output:

```

Proving distrib: A & (B | C) => A & B | A & C ...
1  [ A & (B | C);
2  A;                                by AndEL 1
3  B | C;                             by AndER 1
4  [ B;
5  A & B;                              by AndI 2 4
6  A & B | A & C ];                    by OrIL 5
7  [ C;
8  A & C;                              by AndI 2 7
9  A & B | A & C ];                    by OrIR 8
10 A & B | A & C ];                     by OrE 3 6 9
11 A & (B | C) => A & B | A & C        by ImpI 10
QED

```

Revision Notes

Another Example, $\neg A \wedge \neg B \Leftrightarrow \neg (A|B)$:

```

1    proof deM:  $\sim A \ \& \ \sim B \Leftrightarrow \sim (A|B) =$ 
2    begin
3    [ $\sim A \ \& \ \sim B$ ;
4     $\sim A$ ;
5     $\sim B$ ;
6    [ $A|B$ ;
7    [ $A$ ;
8    F];
9    [ $B$ ;
10   F];
11   F];
12    $\sim(A|B)$ ;
13    $\sim A \ \& \ \sim B \Rightarrow \sim(A|B)$ ;
14   [ $\sim(A|B)$ ;
15   [ $A$ ;
16      $A \ | \ B$ ;
17     F];
18      $\sim A$ ;
19     [ $B$ ;
20      $A \ | \ B$ ;
21     F];
22      $\sim B$ ;
23      $\sim A \ \& \ \sim B$ ];
24    $\sim(A|B) \Rightarrow \sim A \ \& \ \sim B$ ;
25    $\sim A \ \& \ \sim B \Leftrightarrow \sim (A|B)$ ;
26   end;
```

```

first prove it going left to right...
assume the left hand side...
from 3
from 3
assume the opposite...
'or' breakdown
from 4
'or' breakdown
from 5
... prove it false
... prove the right hand side
result
...then prove it going right to left
assumption...
goal is  $\sim A \ \& \ \sim B$ . Assume the opposite...
...prove it false
goal is  $\sim A \ \& \ \sim B$ . Assume the opposite...
... prove it false
...proof
result
end result
```

Output is therefore:

```

Proving deM:  $\sim A \ \& \ \sim B \Leftrightarrow \sim (A \ | \ B) \ \dots$ 
1  [  $\sim A \ \& \ \sim B$ ;
2   $\sim A$ ;
3   $\sim B$ ;
4  [  $A \ | \ B$ ;
5  [  $A$ ;
6  F ];
7  [  $B$ ;
8  F ];
9  F ];
10  $\sim (A \ | \ B) ]$ ;
11  $\sim A \ \& \ \sim B \Rightarrow \sim (A \ | \ B)$ ;
12 [  $\sim (A \ | \ B)$ ;
13 [  $A$ ;
14  $A \ | \ B$ ;
15 F ];
16  $\sim A$ ;
17 [  $B$ ;
18  $A \ | \ B$ ;
19 F ];
20  $\sim B$ ;
21  $\sim A \ \& \ \sim B ]$ ;
22  $\sim (A \ | \ B) \Rightarrow \sim A \ \& \ \sim B$ ;
23  $\sim A \ \& \ \sim B \Leftrightarrow \sim (A \ | \ B)$ 
QED
```

Predicate Logic

A predicate is a relationship or a property of a given variables or constants, e.g. *is male* is a predicate and *Adam* is a constant, hence $\text{Male}(\text{Adam})$ is a proposition. An important feature of predicate logic is the use of quantifiers and variables. \forall is the universal quantifier and means 'for all', for example, $\forall x.\text{Love}(x, \text{Jenny})$ means 'For all x 's, x loves Jenny' in other words, Everybody loves Jenny. \exists is the existential quantifier which means 'There exists', for example, $\exists x.\text{Loves}(\text{Jenny}, x)$ means 'There exists an x such that Jenny loves x ', in other words, Jenny loves somebody.

Note that quantifiers bind *weaker* than connectives, so, $\forall x.\text{Happy}(x) \Rightarrow \text{Loves}(\text{Adam}, \text{Eve})$ means $\forall x.(\text{Happy}(x) \Rightarrow \text{Loves}(\text{Adam}, \text{Eve}))$ *not* $(\forall x.\text{Happy}(x)) \Rightarrow \text{Loves}(\text{Adam}, \text{Eve})$

Some propositions:

- 'Everybody loves their mother' - $\forall x.\text{Loves}(x, \text{mother})$
- 'If there is somebody who you love and who loves you, you are happy' (note that in English, 'you' in this constant means 'this is the case for everyone') - $\forall x(\exists y.(\text{Loves}(x,y) \wedge \text{Loves}(y,x)) \Rightarrow \text{Happy}(x))$
- 'Adam loves Eve' - $\text{Loves}(\text{Adam}, \text{Eve})$

Are the following logical consequences?

- If you are unhappy then you do not love yourself - $\forall x.\neg\text{Happy}(x) \Rightarrow \neg\text{Loves}(x,x)$
(Note the difference between $\forall x.\neg\text{Happy}(x)$ and $\neg\forall x.\text{Happy}(x)$. The former means 'Everybody is unhappy' and the latter means 'Not everybody is happy' - a subtle but vital distinction!)
- Somebody loves everybody - $\exists x\forall y.\text{Loves}(x,y)$
- Adam is unhappy only if Eve does not love him (Note: 'only if' means 'implication'):
 $\neg\text{Happy}(\text{Adam}) \Rightarrow \neg\text{Loves}(\text{Eve}, \text{Adam})$
- If your mother loves you then you are happy - $\forall x.\text{Loves}(\text{Mother}(x),x) \Rightarrow \text{Happy}(x)$
(Note that *mother* is a function, but *loves* is a predicate)

A proposition is a tautology only if it is always true.

It is a contradiction only if it is always false.

It is satisfiable if it is true in at least one interpretation.

So which of the consequences above are logical? We cannot draw truth table in this situation (they would be infinitely long!), so intuition is the best method:

1. *If you are unhappy then you do not love yourself.* This holds by proposition two which states that if you love someone who loves you back, you are happy. Even if you only love one person and they don't love, you could still be happy if you loved yourself, but if you don't, then you'd be unhappy. Hence, the only way to be unhappy is not to love yourself. (What about your mother?, you may ask. Proposition 1 says that everyone loves their mothers NOT that your mother will necessarily love you back!!)

2. *Somebody loves everybody.* This does not hold. Everyone loves their mothers, so it *does* hold that everybody loves somebody (their mothers) but not the other way around (unless everyone has the same mother!)

3. *Adam is unhappy only if Eve does not love him.* The only thing we know about Adam is that he loves two people - his mother and Eve. We are not told whether his mother loves him or not, so we cannot consider her. Hence for our purposes, the other thing we know about Adam is that he loves Eve. Therefore, the proposition holds - if she didn't love him, he would be unhappy. We cannot speculate as to whether he loves himself or whether his mother loves him.

4. *If your mother loves you then you are happy.* Everyone loves their mother (prop 1), so if she loves you back, you are happy (prop 2).

Important Equivalencies

Commuting Quantifiers: $\forall x.\forall y.R(x,y) \Leftrightarrow \forall y.\forall x.R(x,y)$
 $\exists x.\exists y.R(x,y) \Leftrightarrow \exists y.\exists x.R(x,y)$

This is only acceptable when the quantifiers are the *same*. $\exists y\forall x.Loves(x,y)$ means ‘there exists a y such that everybody loves y, essentially ‘Somebody is loved by everyone’ but $\forall x.\exists y.Loves(x,y)$ means ‘Everybody loves Somebody’.

De Morgan: $\neg\forall x.P(x) \Leftrightarrow \exists x.\neg P(x)$
 (i.e. it is not the case that everyone is happy is the same as saying there exists someone who is unhappy)

$\neg\exists x.P(x) \Leftrightarrow \forall x.\neg P(x)$
 (i.e. it is not the case that someone is happy is the same as saying everybody is unhappy)

This rule applies in Propositional Logic, too, in that $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$ and $\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$.

Quantifiers and Propositional connectives: $(\forall x.P(x) \wedge Q(x)) \Leftrightarrow \forall x.P(x) \wedge \forall x.Q(x)$
 $(\exists x.P(x) \vee Q(x)) \Leftrightarrow \exists x.P(x) \vee \exists x.Q(x)$

Equality

‘Equals’ can be a useful tool in expresses things in predicate logic.

For example, to say ‘everyone has one father’ we could write $\forall x\exists y.Father(x,y)$ but this would mean everyone has one and maybe more fathers which is wrong. Instead, consider this:

$\forall x\forall y\forall z.Father(y,x) \wedge Father(z,x) \Rightarrow y = z$

which means If y is x’s father *and* z is x’s father then y and z must be the same person. This means everyone has at most one father (but maybe none). Combining the two ideas gives:

$\forall x\exists y.(Father(y,x) \wedge \forall z.Father(z,x)) \Rightarrow z = y$

Note, if you introduce a new variable, you must use a quantifier.

Predicate Logic in Tutch

\exists is represented by ? and \forall by !

In Tutch, instead of just writing $\forall x.P(x)$, write $\forall x:t.P(x)$ – this is just giving a name, t, to the universe.

Examples:

```
proof alphaAll : (!x:t.P(x)) => (!y:t.P(y)) =
begin
[!x:t.P(x);
 [y:t;
  P(y)];
!y:t.P(y)];
(!x:t.P(x)) => (!y:t.P(y))
end;
```

```

proof allAndCom : (!x:t.P(x) & Q(x)) <=> (!x:t.P(x)) & (!x:t.Q(x)) =
begin
[(!x:t.P(x)) & (!x:t.Q(x));
 !x:t.P(x);
 !x:t.Q(x);
 [x:t;
  P(x);
  Q(x);
  P(x) & Q(x)];
 !x:t.P(x) & Q(x)];
(!x:t.P(x)) & (!x:t.Q(x)) => (!x:t.P(x) & Q(x));

[!x:t.P(x) & Q(x);
 [x:t;
  P(x) & Q(x);
  P(x)];
 !x:t.P(x);
 [x:t;
  P(x) & Q(x);
  Q(x)];
 !x:t.Q(x);
 (!x:t.P(x)) & (!x:t.Q(x))];
(!x:t.P(x) & Q(x)) => (!x:t.P(x)) & (!x:t.Q(x));

(!x:t.P(x) & Q(x)) <=> (!x:t.P(x)) & (!x:t.Q(x))
end;

```

```

proof froeb : (?x:t.P(x) & Q) <=> (?x:t.P(x)) & Q =
begin
[?x:t.P(x) & Q;
 [x:t,P(x) & Q;
  P(x);
  ?x:t.P(x);
  Q;
  (?x:t.P(x)) & Q];
 (?x:t.P(x)) & Q];
(?x:t.P(x) & Q) => (?x:t.P(x)) & Q;

[(?x:t.P(x)) & Q;
 ?x:t.P(x);
 [x:t,P(x);
  Q;
  P(x) & Q;
  ?x:t.P(x) & Q];
 ?x:t.P(x) & Q];
(?x:t.P(x)) & Q => (?x:t.P(x) & Q);

(?x:t.P(x) & Q) <=> (?x:t.P(x)) & Q
end;

```

```

proof exOrCom : (?x:t.P(x) | Q(x)) <=> (?x:t.P(x)) | (?x:t.Q(x)) =
begin
[?x:t.P(x) | Q(x);
[x:t,P(x) | Q(x);
[P(x);
?x:t.P(x);
(?x:t.P(x) | (?x:t.Q(x))];
[Q(x);
?x:t.Q(x);
(?x:t.P(x) | (?x:t.Q(x))];
(?x:t.P(x) | (?x:t.Q(x))];
(?x:t.P(x) | (?x:t.Q(x))];
(?x:t.P(x) | Q(x)) => (?x:t.P(x)) | (?x:t.Q(x));

[(?x:t.P(x)) | (?x:t.Q(x));
[?x:t.P(x);
[x:t,P(x);
P(x) | Q(x);
?x:t.P(x) | Q(x)];
(?x:t.P(x) | Q(x))];
[?x:t.Q(x);
[x:t,Q(x);
P(x) | Q(x);
?x:t.P(x) | Q(x)];
(?x:t.P(x) | Q(x))];
(?x:t.P(x) | Q(x))];
(?x:t.P(x) | (?x:t.Q(x)) => (?x:t.P(x) | Q(x));

(?x:t.P(x) | Q(x)) <=> (?x:t.P(x)) | (?x:t.Q(x))
end;

```

Primitive Recursion

Primitive Recursion is the method of defining a function by defining the function for 0 and then writing an expression for the result given for $n+1$ in terms of n . The successor notation is used – $s(n)$ means $n+1$, so $s(2) = 3$.

E.g., factorial can be defined as:

```

fac      :    nat → nat
fac(0)   =    s(0)
fac(s(n)) =    s(n) x fac(n)

```

and doubling can be defined as:

```

doubl    :    nat → nat
doubl(0) :    0
doubl(s(n)) :    s(s(doubl(n)))

```

The Tutch syntax for this is:

```

val dbl    :    nat -> nat
    = fn x => rec x of
        dbl 0 => 0
        |   dbl (s n) => s (s (dbl (n)))
    end;

```

Some functions take two numbers as input, such as 'add':

add : nat → nat → nat
 add(0)(m) : m
 add(s(n))(m) : s(add(n)(m))

Induction

Induction is a proof technique using the idea of primitive recursion. It involves two steps – an initial step (checking the function for 0) and an inductive step (checking the function for n+1).

For example,

To prove that $\sum_{i=0}^n i = \frac{n(n+1)}{2}$

Initial step: $0 = \frac{0(0+1)}{2} = 0/2 = 0$

Inductive step: $\frac{n(n+1)}{2} + (n+1) = \frac{(n+1)((n+1)+1)}{2}$
 $\frac{n(n+1) + 2(n+1)}{2} = \frac{(n+1)(n+2)}{2}$
 $\frac{n^2 + n + 2n + 2}{2} = \frac{n^2 + 2n + n + 2}{2}$
 $\frac{n^2 + 3n + 2}{2} = \frac{n^2 + 3n + 2}{2}$

The principal of induction is that if the formula works for any number n and also for the number after that, n+1, then the formula is valid.

Set Theory

Sets have elements: $x \in A$ means the object x is an element of the set A.

A set is determined by its members.

Finite Sets are defined by listing its elements, i.e. $A = \{1,3,5,7\}$.

Sets may contain other sets, i.e. $B = \{1,2,\{3,5\},7\}$

An empty set, $\{ \}$, may be represented by \emptyset

The \in relation concerns only the top-level elements of the set, i.e.

$\{1\} \in \{\{1\},\{2,3\}\}$
 but $1 \notin \{\{1\},\{2,3\}\}$

Two sets are equal iff they have the same members (order and repetition do not matter, so $\{1,2,3\} = \{3,1,1,1,2,2\}$).

Infinite Sets can obviously not be defined by listing the elements. Instead, they are defined by comprehension. Some common infinite sets are \mathbb{N} (natural numbers $\{0,1,2,3..\}$), \mathbb{Z} (integers $\{-2,-1,-0,1,2..\}$), \mathbb{Q} (rational numbers, i.e. fractions) and \mathbb{R} (real numbers, i.e. non-terminating numbers, etc.).

Defining by Comprehension:

$\{x \mid x \in \mathbb{N} \wedge 1 \leq x \wedge x \leq 3\}$ defines the set made up of x 's, where x is a natural number AND is greater or equal to 1 AND is less than or equal to 3, i.e. $\{1,2,3\}$.

Subsets and Powersets

X is a subset of Y iff every element of X is also in Y . Denoted by $X \subseteq Y$.

An empty set is a subset of any set

Every set is a subset of itself

A powerset is the set of subsets of a given set. I.e., $P(\{1,2,3\}) = \{\{\}, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$

A powerset always contains an empty set and the set itself. If X has n elements, $P(X)$ has 2^n elements.

Boolean Algebra of Sets

Given two sets, A and B , $A \cup B$ represents the elements which are in A and the elements which are in B .

$A \cap B$ represents the elements that are in both A and B .

Complement is represented by a line over the expression, e.g., if $A \subseteq U$, \overline{A} represents all the elements in U that aren't in A .

Difference: The elements that are in A but not in B is given by $A \setminus B$ (which is equivalent to $A \cap \overline{B}$)

If two sets have the same number of elements, they are said to be *isomorphic*.

Rules:

\cup is associative: $A \cup (B \cup C) = (A \cup B) \cup C$

\cup is commutative: $A \cup B = B \cup A$

\emptyset is a unit for \cup : $A \cup \emptyset = A$

U is a zero for \cup : $A \cup U = U$

\emptyset is a unit for \cap : $A \cap \emptyset = \emptyset$

Law of complement for \cap : $A \cap \overline{A} = \emptyset$

Distributivity for \cap : $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

Distributivity for \cup : $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

Double complement: $\overline{\overline{A}} = A$

De Morgan for \cap : $\overline{A \cap B} = \overline{A} \cup \overline{B}$

De Morgan for \cup : $\overline{A \cup B} = \overline{A} \cap \overline{B}$

Russell's Paradox (from www.jimloy.com):

There was once a barber. Some say that he lived in Seville. Wherever he lived, all of the men in this town either shaved themselves or were shaved by the barber. And the barber only shaved the men who did not shave themselves. That is a nice story. But it raises the question: Did the barber shave himself? Let' s say that he did shave himself. But we see from the story that he shaved only the men in town who did not shave themselves. Therefore, he did not shave himself. But we again see in the story that every man in town either shaved himself or was shaved by the barber. So he did shave himself. We have a contradiction.

Some sets contain themselves. Let' s say that S is a set which contains S and $\{A, B\}$. Then this is S : $\{S, \{A, B\}\}$. It contains two sets, itself and $\{A, B\}$. The set of all sets obviously contains itself. Well, let' s construct a very interesting set, the set of all sets which do not contain themselves. There is something wrong here. Does "the set of all sets which do not contain themselves" sound like "the barber who shaves all men who do not shave themselves?" The story of the barber was inconsistent. The set of all sets which do not contain themselves is inconsistent for the same reason. Does the set of all sets which do not contain themselves actually contain itself, or not? If it contains itself, then it cannot contain itself. If it does not contain itself, then it must contain itself. It is inconsistent.

The way out of this problem was given by Zermelo and Fraenkel who declared that we may only define subsets of already existing sets.

Relations

The *cartesian product* of two sets, A and B (written $A \times B$) is defined as the set of pairs of the form (a,b) where $a \in A$ and $b \in B$. In other words, $A \times B = \{(a,b) \mid a \in A \wedge b \in B\}$.

The *disjoint union* of two sets (written $A + B$) is defined as pairs (t,x) of a boolean value t where $x \in A$ if t = true, or $x \in B$ if t = false. $A + B = \{(T,a) \mid a \in A\} \cup \{(F,b) \mid b \in B\}$

E.g.:

$A = \{0,1\}$

$B = \{1,2,3\}$

$A \times B = \{(0,1), (0,2), (0,3), (1,1), (1,2), (1,3)\}$

$A + B = \{(T,0), (T,1), (F,1), (F,2), (F,3)\}$

N.B. $|A \times B| = |A| \times |B|$ and $|A+B| = |A| + |B|$

A *relation* is a subset of a cartesian product, i.e. the relation $R \subseteq A \times B$

Properties of Relations

Reflexive iff every element can be related to itself.

Transitive if whenever x is related to y, and y to z, then x is related to z ('shortcut')

Symmetric if whenever x is related to y, then y is related to x.

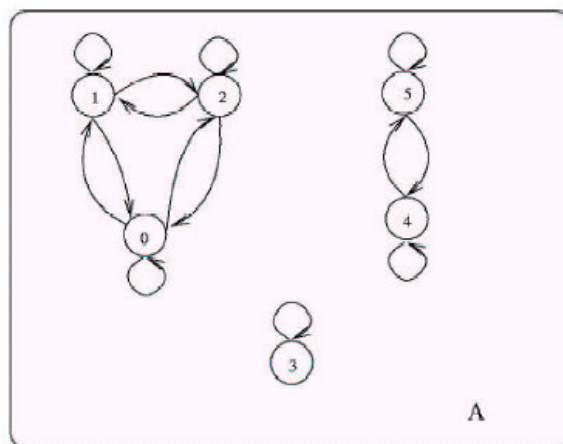
Antisymmetric if x is related to y but y is never related to x.

A relation is *equivalent* if it is reflexive, transitive and symmetric.

E.g., $A = \{0,1,2,3,4,5\}$

$R \subseteq A \times A$

$R = (0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2), (3, 3), (4, 4), (4, 5), (5, 5), (5, 4)$



In the graph above, three clusters of related elements formed. The set of sets of clusters is called *the quotient set of A by R*.

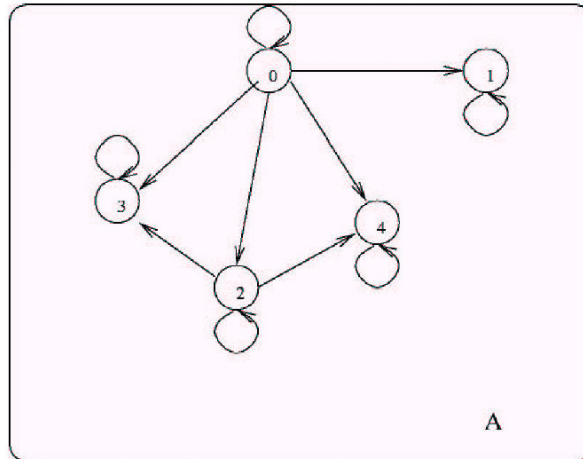
Revision Notes

A relation is a partial order if it is reflexive, antisymmetric and transitive, e.g., 'less than or equal to':

$$A = \{0, 1, 2, 3, 4\}$$

$$R = A \times A$$

$$R = \{(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (2, 2), (2, 3), (2, 4), (3, 3), (4, 4)\}$$



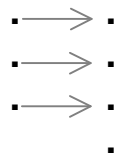
Functions

A function is a relation which is both total and unique, i.e. every input maps to exactly one output.

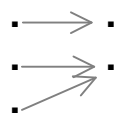
The inputs are called the *domain* and the outputs are called the *co-domain* or *range*.

Properties of functions

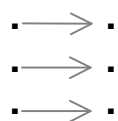
A function is *injective* iff no element of the range appears twice (i.e. no duplicate outputs)



A function is *surjective* iff each element of the range appears at least once (i.e. every element on the right is hit)



A function is *bijective* iff it is injective and surjective.



If a bijection is possible between two sets, it can be held that both sets contain an equal number of elements.