# Week 1

## Lecture 1: Intro, Propositional Logic

Two assessed exercises worth 10%

Lectures (videos watched beforehand):

* (maybe) Wednesdays – Q&A
* Fridays – example sessions

Videos:

* 10-15min videos
* 3-5 videos per lecture

Propositional logic – the logic of compound statements built from simpler statements using Boolean connectives

* Propositions are declarative sentences that are either true or false
  + Denoted by lowercase letters (p,q,r,…)
  + - not – negation
* Compound propositions or formulae use connectives
  + Uppercase letters (P,Q,R,…)
  + Truth tables used to illustrate them
  + Connectives:
    - - and – conjunction
    - - or – disjunction
    - - xor – exclusive or
    - Implication -
      * is true with one exception: if p is true and q is false
    - Biconditional (“pq” means “p if and only if q”)
  + Precedence – using parentheses for order of connectives
    - Negation takes precedence
  + Tautology – always true
    - p implies p, p or not p
  + Contradiction – always false
    - p and not p
  + Contingency – neither a tautology nor a contradiction
  + Satisfiable formula – not a contraction

Relational implications (p implies q):

* Converse (q implies p)
* Contrapositive (not q implies not q)
  + Equivalent to original implication
* Inverse (not p implies not q)

## Lecture 2: Logical Equivalence

Logical equivalence – two syntactically (textually) different compound propositions are semantically identical (have the same meaning)

* (P is logically equivalent to Q)
* How to prove:
  + by laws of logical equivalence
  + by a truth table
  + by some other line of reasoning
* How to prove false:
  + one example where they’re different

Laws of logical equivalence

* Identity
* Domination
* Idempotent
* Double negation
* Commutative
* Associative
  + if there are multiple and’s or or’s (but just them separately), brackets don’t matter
* Distributive
* De Morgan
* Contradiction and tautology
* Implication
* Exclusive or
* Biconditional

# Week 2

## Lecture 3: Predicate Logic

Predicate logic culminated in Goedel’s Incompleteness Theorem:

* “given any finitely describable, consistent proof procedure, there will be some true statements that can never be proven by that procedure

Predicates allow the construction of statements with variables, where the variable is the subject of the statement

A predicate P is a propositional (Boolean) function

* a mapping from some domain U to truth values

Predicates become compound propositions if

* variables are assigned values or
* variables are **bound** with values from its domain U through **quantifiers**

Predicate P(y) is **free** (**unbounded**) because y is a variable. Its truthfulness is not known

* Anything involving P(y) is not a compound proposition

Quantifiers:

* - universal quantifier – for all (should use with domain) (same as *and* everything)
* - existential quantifier – exists (with domain) (same as *or* everything)

A variable is bound to a quantifier if it appears free withing the scope of the quantifier

* (because of inner scope)
* Ordering matters:
  + for all x there exists a y =/= there exists a y such that it holds true for all x
  + But can swap quantifiers if they are the same
* The universe of discourse – domain U

Quantifier Negation Laws

* (same with opposite quantifier)
* (same with opposite quantifier)

Distributivity holds only for universal and conjunction, existential and disjunction

## Lecture 4: Sets

Sets:

* A set A is an unordered collection of elements or members
  + Order and number of occurrences do not matter
* Notation
  + Enumeration and curly brackets
  + Floor of a real x, denoted or floor(x) is the largest integer smaller than x
  + Ceiling of a real x, denoted or ceil(x) is the smallest integer greater than x
  + Set builder
* Power set
* Russel’s Paradox
* Cartesian Product
  + A set of ordered tuples
* Subsets
  + Improper subset:
  + Empty set is a subset of every set
  + Proper (strict) subset:
    - |A|<|B|
* Equality
* Disjoint
  + Two sets are disjoint if they contain no common elements
* Binary representation:
  + Given a universal set, represent another set by giving 1 for existing element of the universal set and giving 0 for non-existing
* Set operations:
  + Union
    - Disjunction
  + Intersection
    - Conjunction
  + Complement
    - Negation
  + Set difference
  + Symmetric difference
    - Exclusive or
  + Set equivalences – same rules but:
    - U is true
    - empty set is false
    - union is disjunction
    - intersection is conjunction
    - complement is negation

Ways to prove equality of sets:

* a membership table
* a containment proof
  + A is a subset of B
  + B is a subset of A
* set comprehension notation and logical equivalences
* Venn diagrams

# Week 3

## Lecture 5: Functions, Countability

Function: the assignment of elements of a set to elements of another set

A function is a mapping from elements of set X to elements of set Y

* f can be considered a subset of , i.e. by a set of tuples satisfying
  + (every x gets mapped to some value)
  + (only one value of some x)
* X – domain; Y – codomain
* if f(x)=y, then **y is the image of x** and **x is the preimage of y**
* there may be more than one preimage of y
  + there can exist distinct x1 and x2 such that f(x1)=y and f(x2)=y
* Range of f is the set of all images of all
* Image:
  + if S is a subset of the domain, the image of S is
    - i.e. the range of f when the domain of f is restricted to S

Operations:

* Addition and multiplication: first compute values, then add/multiply
* Composition
  + applying functions in sequence
  + Can only be done if the range of the inner function is a subset of the domain of the outer one

Types:

* Injective (one-to-one) f: X to Y
  + informally f maps different elements of X to different elements of Y
  + formally
  + Ex: strictly increasing or strictly decreasing
* Surjective (onto) f: X to Y
  + informally each value in the codomain has a preimage
  + formally
* Bijective (both)
  + maps different elements of X to different elements of Y
  + each element of codomain has a preimage

Inverses:

* For the inverse of a function to exist, the function must be bijective
* For a bijective function f: X to Y, the inverse of f is the function where if

Countability:

* Cardinality – size (number of elements) of a set
* The cardinality of set A is equal to the cardinality of a set B if and only if there exists a bijection from A to B (or B to A)
* Countable: if a set has the same cardinality as a subset of the natural numbers
  + i.e. indexable
  + Reverse is uncountable
* Countably infinite: if a set has the same cardinality as the natural numbers
  + The integers are countably infinite
  + The positive rationals are countably infinite (shown by traversing a matrix)
    - Thus, is countably infinite
* Uncountable sets:
  + The set of real numbers between 0 and 1
    - Proof by contradiction
* Computability:
  + A number between 0 and 1 is computable if there is a program which, when given the input i, produces the digit of the decimal expansion of that number
  + There are numbers between 0 and 1 that are not computable
    - Uncountable reals, countable computing programs => more numbers than programs

## Lecture 6: Sequences, Divisibility, Primes, FTA

Types of sequences:

* Arithmetic progression
* Geometric progression

Summation and product:

* – sum
  + Shifting indices
    - substitute and work out j
  + Can swap Sigma signs for sums, not for products
* – product

a|b means “a divides b”, “a is a factor of b”, “b is a multiple of a”

* If a|b and a|c, then a|(b+c)
* Division algorithm:
  + If a is an integer and d is a positive integer, then there exist unique integers q and r such that and
  + a – dividend
  + d – divisor
  + q – quotient
  + r – remainder (in the range {0,1,2,…,d-1})
  + – exists a unique

Prime: an integer bigger than 1 whose only positive factors are 1 and itself

* Composite number – non-prime

Fundamental Theorem of Arithmetic:

* every positive integer (>1) can be expressed as a unique product of primes
  + Proof uses Well Ordering Principle (WOP)
    - WOP: every non-empty set of positive integers has a least element

Checking prime factorisation:

* If n is composite, it has a prime divisor

# Week 4

## Lecture 7: Greatest Common Divisor, Euclidean Algorithm, Matrices

Greatest Common Divisor – gcd(a,b) is the largest positive integer d such that d|a and d|b

* If gcd(a,b)=1 then a and b are relative primes

Least Common Multiple – lcm(a,b) is the smallest x such that a|x and b|x

Modulo arithmetic

* a mod m is the remainder of a after dividing by m
  + a mod m is the integer r such that a = qm + r (and
* a is congruent to b modulo m if m divides a – b
  + if
* Properties
  + if and then
  + if and then
  + a is congruent to b modulo m if and only if a mod m = b mod m
* Use: Public key cryptography
  + encryption and decryption are carried out using 2 different keys
    - Public key for encryption (available to anyone)
    - Private key for decryption (available only to receiving party)
  + Can encrypt with private key to authenticate sender
  + RSA (Rivest, Shamir and Adleman) public-key encryption algorithm is the basis of most modern secure communications
    - Each party has:
      * a modulus n=pq where p and q are large primes
        + p and q can be found relatively quickly
      * an exponent e that is relative prime to (p-1)(q-1)
    - Messages are translated into sequences of integers
    - Encryption transforms int M into int C = Me mod n
      * from C can retrieve M quickly using the fact where d is a multiplicative inverse of e mod (p-1)(q-1)
        + d is such that

Euclidean Algorithm

* If (all ints), then
  + Continue recursively

Matrices

* Rectangular arrays (usually, of numbers)
* Summation
  + Must have same dimensions
* Multiplication
  + Scalar products of vectors
  + Not commutative (AxB =/= BxA)
  + Associative (brackets don’t matter)
  + Distributive
* Identities
  + Identity matrix – with 1’s along upper-left to lower-right diagonal and 0’s everywhere else
  + Multiplying with any matrix gives the same matrix
* Inverses
* Transpose
  + [a­­j,i­]
* Symmetric if and only if A=AT

## Lecture 8: Rules of Inference

Inference rules

* modus ponens (“mode that affirms”)
  + From the premises p and p implies q, conclude q
  + A rule of Propositional logic
  + Based on the tautology
* modus tollens
  + Based on the tautology
* Hypothetical syllogism
  + Based on the tautology
* Disjunctive syllogism
  + Based on the tautology
    - If both the statement p or q is true and p is false (not p true), then q must be the one that’s true
* Resolution
  + Based on the tautology
  + Expanded disjunctive syllogism
* Addition (or “or introduction”)
  + Based on the tautology
* Simplification
  + Based on the tautology
* Conjunction (or “and introduction”)
  + Based on the tautology

Argument – a sequence of compound propositions

* all but the final compound proposition are called premises
* the last compound proposition is the conclusion
* The argument is valid if and only if the truth of all premises implies the conclusion is true
  + Argument is valid when is a tautology
* In practice, start from conclusion to see the sequence in reverse order

Quantifiers

* Universal quantifier
  + Rule: Universal instantiation
  + Rule: Universal generalisation
    - Converse of universal instantiation
* Existential quantifier
  + Rule: Existential instantiation
  + Rule: Existential generalisation
    - Converse of existential instantiation

Other rules:

* Type rules
  + If two numbers are integers, their sum/subtraction is an integer
  + If one integer variable is assigned another integer variable, it will be int
* Type set rules
  + ???

# Week 5

## Lecture 9: Methods of Proof

A proof must be a logical, convincing argument

There are varying degrees of formality

A proof shows that from some premise P some conclusion Q holds

* Direct proofs (based on implication)
  + Assume P is true
  + Show Q is true using
    - rules of inference
    - theorems already proved
* Indirect proofs (based on contrapositive of implication)
  + Assume Q is false
  + Show P is false using
    - rules of inference
    - theorems already proved

Proving “if and only if”

* Prove implication both ways

Proving “equivalence of statements”

* To prove P, Q and R are equivalent, sufficient to show
  + P implies Q
  + Q implies R
  + R implies P
* Can also use hypothetical syllogism to complete the proof
  + Q implies P
  + R implies Q
  + P implies R

Trivial proof

* Tautologies

Proof by contradiction

* Assume the negation and show that this assumption leads to a contradiction

Proof by cases

* To prove P implies Q
  + find a set of premises P1, P2, …, Pn
  + such that P implies (P1 or P2 or…or Pn)
* Then prove
  + P1 implies Q
  + P2 implies Q
  + …
  + Pn implies Q
* Proof uses distributivity

Vacuous proof

* When proving P implies Q, if we can show P does not hold, we are done

Existence proof:

* (Dis)Prove something by presenting an instance (a witness)
  + Producing an actual instance
  + Showing how to construct an instance
  + Showing it would be absurd if an instance did not exist

What not to do:

* Circular reasoning
* Fallacies: Inference rules or other proof methods that are not logically valid and therefore can yield a false conclusion
  + Affirming the conclusion (if P implies Q is true and Q is true, P must be true)
  + Denying the hypothesis (if P implies Q is true and P is false, Q must be false)

## Lecture 10: Induction, Bubble Sort

Induction – a technique for proving universally quantified theorem over inductively defined sets

* Proof by mathematical induction (over integers/naturals)
  + Base case
  + Inductive step (using inductive hypothesis)

Induction and Computing:

* Analysing algorithms (big O, correctness)

# Week 6

## Lecture 11: Recursion

Inductive/Recursive functions:

* Specify a function at its lowest/minimum level(s) (0,1,empty)
* Give a rule for finding a value from existing (lower) values
* Ex:
  + Factorial
  + Fibonacci sequence
    - Applications in CS:
      * Fibonacci search algorithm
        + similar to binary search when searching a sorted array
      * Fibonacci heap data structure
        + implements a priority queue
        + improved worst case running time over other data structures (like heap)
  + Can express addition, multiplication, exponentials, Euclid’s algorithm recursively

Recursively defined sets

* State what is in the set to start with (base case)
* State how to construct new elements of the set depending on base case (inductive step)
* Ex:
  + Strings
    - Concatenation
    - Length

Recursive trees:

* base case: nil is a tree over X (X is a data set, equals natural numbers)
* inductive step: if t1 and t2 are trees over X and x in X, then node(t1,t2,x) is a tree over X
* Non-empty proper binary trees over X:
  + base case: if x in X, then node(nil, nil, x) is a tree over X
  + inductive step: if t1 and t2 are trees over X and x in X, then node(t1, t2, x) is a tree over X
  + Non-empty: trees have at least one node
  + Proper: every non-leaf node has two children
* Counting recursively:
  + Nodes
  + Levels

Well-formed formula:

* Uses
  + true and false
  + propositions
  + logical operators (not, or, and, implies)
* Recursive definition
  + base case: true, false and proposition p are well formed
  + inductive step: if P and Q are well formed, then are well formed

## Lecture 12: Induction Over Recursive Functions

Induction over Strings

* Recursive definition:
  + base case: epsilon in sigma\* (empty string in set)
  + inductive step: if w in sigma\* and x in sigma (letter x in alphabet), then xw in sigma\*
* Proving a theorem over strings
  + base case: show P(epsilon)
  + inductive step: assume P(w) and show P(xw) where w in sigma\* and x in sigma arbitrary
* Another method for proving theorem over strings:
  + base case: show P(w) for all strings w of length <= k
    - including empty string epsilon
  + inductive step: assume P(w), show P(xw) where w of length >= k & x arbitrary

Induction over Binary Trees

* Recursive definition of non-empty proper (all nodes either have 2 children or are a leaf) binary trees over X (X – data set)
  + base case: node(nil, nil, x) is a tree over X and x in X
  + inductive step: if t1 and t2 are trees over X and x in X, then node(t1, t2, x) is a tree over X
* Proving P(t) holds over all non-empty proper binary trees inductively
  + base case: for arbitrary x in X, show P(node(nil, nil, x)) holds
  + inductive step: for arbitrary x in X, t1 and t2, assume P(t1) and P(t2), then show P(node(t1, t2, x))
* Recursive definition for (general) binary trees over X (can be empty, can have 0/1/2 children)
  + base case: nil is a tree over X
  + inductive step: if t1 and t2 are trees over X
* Prove P(t) for general binary trees
  + base case: P(nil) holds
  + inductive step: for arbitrary x in X, t1 and t2, assume P(t1) and P(t2), show P(node(t1,t2,x))

# Week 7

## Lecture 13: Counting Rules

CS applications for counting (combinatorics)

* determining complexity of algorithms
* finding number of operations to be executed by a nested lop in a program
* computing number of different IP addresses
* computing probabilities in AI and machine learning
* finding number of paths between vertices in a graph
* security: counting number of available passwords

Product rule:

* If there are M elements in set A and N elements in set B, then there are ways to combine one element from A with one element from B
* Tasks: If there are M ways to do the first task and N ways to do the second and you must perform both, then there are options
  + Can be extended to 3+ cases

Sum rule:

* If there are M elements in set A, N elements in set B and the elements of A and B are disjoint, then there are M+N ways to choose an element from either A or B
* Tasks: if a first task can be done in M ways, a second in N ways, and you must perform either the first or second task (not both), then there are M+N options
  + Extended to 3+ cases

Inclusion-Exclusion Principle

* The sum rule is a special case of a more general formula when the 2 sets can overlap, namely
* Tasks: if there are M ways to do task A and N ways to do task B and both can be done at the same time, then using the sum rule we over-count by K, where K is the number of ways A and B are done at the same time
  + Extension for 3 cases:

Pigeonhole principle:

* If n objects are placed in k container, then at least one container has objects
  + ceil(x) (ceiling of x) – smallest integer greater than or equal to x

## Lecture 14: Permutations, Combinations

Permutations

* An ordered arrangement of objects
* An r-permutation is an ordered arrangement of size r
* The number of r-permutations of a set of size n is
* Properties:
  + Number of r-permutations of a set of n objects with repetition equals
  + The number of different permutations of n objects where each is indistinguishable is

Combinations:

* An unordered arrangement of objects
* And r-combination is an unordered arrangement of size r
  + If ordering is ignored, permutations overcount
* The number of r-combinations of a set of size n is
* Often pronounced “n choose r”
  + Also called the binomial coefficient
* Properties:
  + The number of r-combinations of a set of size n is the same as the number of (n-r)-combinations
  + Recurrence relation:
  + Considering r-combinations from a set of n elements with repetition:

# Week 8

## Lecture 15: Probability

Basics

* Uses of probability
  + Weighing evidence
  + Diagnosing problems
  + Analysing situations the exact details of which are unknown
* Probability theory deals with random experiments
  + Processes/actions whose outcome cannot be predicted with certainty and may differ if the experiment is repeated
* Definitions:
  + Experiment (trial) – an occurrence with an uncertain outcome
  + Outcome – result of an experiment
    - One particular state of the world
  + Sample space – the set of all possible outcomes for the experiment
  + Event – subset of possible outcomes with some common property
  + Probability – the degree of certainty that an event will occur (P[A] to denote probability of event A)
  + Frequency – how often an outcome occurs in a sequence of experiments
  + Probability distribution – function
* Views of probability
  + Bayesian view
    - Probability is a calculus of belief
      * probabilities are measures of degrees of belief
      * P[A] = 0 means a belief that cannot hold
      * P[A] = 1 is a belief that is absolutely certain
      * quantifying belief about phenomena given the info we have
    - Bayesians allow for belief in states to be combined and manipulated through the rules of probability
    - The key process in Bayesian logic is updating our beliefs
  + Frequentist view
    - Probability is only the long-term behaviour of repeated events
      * Calculating average proportion of outcomes over infinite times
    - Does not make sense to talk about probability of events that happen only once, e.g. raining now
  + Laplacian, Axiomatic views

Axioms of probability

* (for set of elementary outcomes and events )

1. If A and B are mutually exclusive, then

* Derivations:
  + for any events A and B

Random variables

* Variables that can take different values
* Discrete or continuous
* Probability theory allows manipulating random variables without having to assign them a fixed value
* Probability distribution defines how likely different values x of a random variable X are
  + Write P[x] as shorthand for P[X=x]
  + Can be considered as a mapping from a sample space to a set of values
    - e.g., X – no. of heads on coin toss; X(TTT) = 0, X(HTH) = 2
* If X is a random variable, given Y and function , Y can also be considered to be a random variable where

Joint probabilities and distributions

* If X and Y are (discrete) random variables, the joint probability distribution of X and Y is the function
* X and Y are independent if and only if for all x and y
  + e.g., X is the outcome of a coin toss and Y is the outcome of a dice roll
* Events A and B are independent if and only if

Marginal probabilities and distributions

* Marginalisation refers to the process of ‘removing’ the influence of 1+ events from a probability
* The functions given by
* are the marginal distributions of X and Y
  + where either X or Y has been ‘removed’

Conditional probabilities and distributions

* For events A and B, if P[B]>0, then the conditional probability of A given B is defined by:
  + If A and B are independent, then P[A|B] = P[A]
  + 3 cases:
* Partition theorem (the law of total probability)
  + If forms a partition of the sample space
    - i.e.
  + then
  + Uses:
    - probability of a second card taken from a deck being diamond where the partition is: 1st card diamond, 1st card not diamond
* Distributions:
  + - Let X and Y be (discrete) rando variables
    - f(x,y) = P[X=x, Y=y] is the joint probability distribution of X and Y
    - g(x) = P[X=x] is the marginal distribution of X
    - h(y) = P[Y=y] is the marginal distribution of Y
    - fixed
  + The functions is the conditional distribution of X given Y=y
  + Can similarly define conditional distributions of Y given X=x

## Lecture 16: Probability (Bayes’ Rule, Variance, Entropy)

Bayes’ rule:

* where:
  + P[A|B] – posterior
    - what we want to know/will compute
  + P[B|A] – likelihood
    - how likely the event is to produce the evidence we see
  + P[A] – prior
    - how likely A is regardless of evidence
  + P[B] – evidence
    - how likely B is regardless of event
* often phrased where A – hypothesis H and B – data D

Expected value:

* For a random variable X having a numeric domain the expected value of is defined as
  + the weighted average value or arithmetic mean value
* Properties:

Variance

* The variance of a random variable X is the expected value of the square of the difference between its value and its expected value
* Standard deviation is the square root of the variance
* Both measure how far the possible values differ from average
* Alternatively, quantifies the dispersion in a set of data values

Entropy

* A measure of the ‘surprise’ an observer would have in the result
* A precise quantification of the info in a distribution
* Defined as
  + log\_2 if units of info are bits
* Entropy gives how many bits are needed (at min) to communicate a value from a distribution to an observer who knows the distribution

# Week 9

## Lecture 17: Graphs

Undirected graphs:

* G = (V, E)
  + V – finite set of vertices (the vertex set) (pictorially represented by a point)
  + E – set of edges, each edge is a subset of V of size 2 (edge set) (pictorially represented by a line joining the relevant point pair
  + Definition doesn’t allow (for a **simple graph**):
    - Loops
    - Multiple edges between two vertices
* Terminology
  + Adjacent vertices – joined by line (the pair is an element of the edge set E)
  + Non-adjacent vertices – not joined by line (not in E)
  + Vertices can be **incident** to edges (having an edge)
  + Degree of a vertex – no. of edges it is incident to

Directed graphs (digraphs):

* D = (V, E)
  + V – finite set of vertices
  + E – finite set of edges (**ordered** pairs) (pictorially represented as directed lines/arrows)
* Terminology:
  + Source (initial) vertex – origin point of arrow
  + Target (final) vertex – vertex being pointed to
  + Source vertex is **adjacent to** target vertex, and target vertex is **adjacent from** source vertex
  + **In-degree** of vertex – no. of edges that have the vertex as its target ( deg+(x) )
  + **Out-degree** of vertex – no. of edges that have the vertex as its source ( deg-(x) )

Isomorphic graphs

* Undirected graphs and are isomorphic if:
  + there is a bijection f from V1 to V2 such that if and only if
    - i.e. v and w are adjacent if and only if f(v) and f(w) are adjacent
  + there is a bijection f from V1 to V2 such that if and only if
    - i.e. an edge from v to w if and only if and edge from f(v) to f(w)
* So far, the best algorithm for checking 2 graphs are isomorphic is exponential in the worst case
  + However, there are simple necessary (but not sufficient) conditions
    - V1 and V2 must be the same cardinality
    - E1 and E2 must be the same cardinality
    - All vertices must have same degree

Connectivity

* In an undirected graph G=(V,E) a path of length n from vertex v to vertex w is a sequence of vertices that go from v to w by traversing n edges
  + **Simple** path – no vertex is repeated
* A cycle (circuit) is a path of length 2+ that begins and ends with the same vertex
* In a directed graph, the path must follow direction of edges
* Terminology
  + **Connected** graph – every pair of vertices is joined by a path
  + Non-connected graph has 2+ connected components
  + Tree – a connected and acyclic (no cycles) graph
    - A tree with n vertices has n-1 edges
      * at least n-1 to be connected
      * at most n-1 to be acyclic
  + Forest – acyclic graph with its component being trees

Euler Cycle problem

* Is there a cycle in the graph that contains every edge?
* A connected undirected graph has an Euler cycle if and only if each vertex has even degree

Hamiltonian cycle problem

* Is there a cycle in the graph that visits each vertex exactly once? (given graph G=(V,E))
* Dirac’s theorem: if and the degree of every vertex is at least , then G has a Hamiltonian cycle
* Ore’s theorem: if and for every pair of non-adjacent vertices v and w, then G has a Hamiltonian cycle

Graph colouring problem

* Colour the map so that adjacent states are different colours with as few colours as possible
* Given graph G and target integer K, can one of K colours be attached to each vertex of G so that adjacent vertices always have different colours?

## Lecture 18: Relations

A binary relation R over sets A and B is a subset of A x B

* R is a set of ordered pairs of the form (a,b) (a in A and b in B)

N-ary relations:

* There can be a relation between n sets, i.e. a set of ordered n-tuples
* A relation between the sets is a subset of
  + an element of the relation is of the form
* Terminology:
  + n=1: a unary relation (singletons)
  + n=2: binary (pairs)
  + n=3: ternary (triples)

Representation of relations:

* Directed graph
  + where if and only if (the relation over A x B)
  + NB: loops possible (not simple graphs)

Properties:

* A binary relation R over AxA is
  + **reflexive**: every element is related to itself
  + **symmetric**: a is related to b if and only if b is related to a
    - (a,b) in R, then (b,a) in R
  + **anti-symmetric**: if a is related to b and they are distinct, then b is not related to a
  + **transitive**: if a is related to b and b is related to c, then a is related to c
* **Equivalence relations**: reflexive, symmetric and transitive

Composition of relations:

* Let R be a relation over A x B and S be a relation over B x C
* Composition of R and S () is the relation over A x C such that if and only if there exists
* Composing a relation with itself:
  + If R is a relation over A x A for set A
  + The powers Rn of the relation Rn where n=0,1,2,… are defined inductively:

Closures:

* If there is a relation R, then the closure of R with respect to some property P is given by the relation S, where S is “R union [the minimum number of tuples that ensures property P holds”
  + Property P: reflexivity/symmetry/transitivity
* Reflexive closure
  + - (diagonal relation)
* Symmetric closure
  + - (inverse of relation)
* Transitive closure
  + Finding all (a,b) such that there is a path from a to b in the digraph representing R

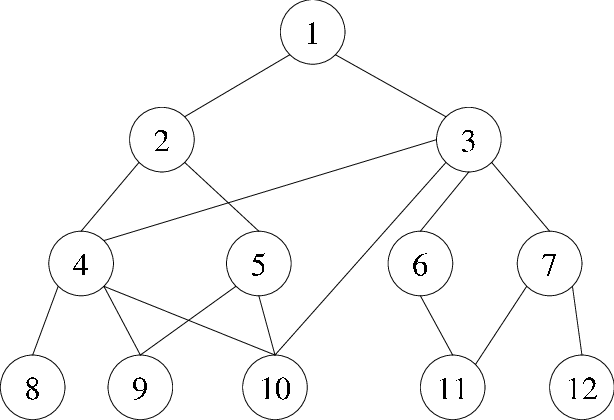
Partial orders:

* A relation R over S x S is a partial order on S x S if it is:
  + reflexive
  + anti-symmetric
  + transitive
* Standard notation:
* A set S with a partial order on S is called a partially ordered set (poset) and is denoted (S,)
  + ordered: due to anti-symmetry
  + partial: because pairs of elements may not be related
* Properties:
  + A partially ordered set (S,) cannot have cycles

Lexicographic ordering

* (used for ordering sets constructed as products, strings, words)
* (required ordering on the original sets)
* (for a partial order relation on S, let be the relation where if and only if )
  + NB: sufficient to define as it’s possible to obtain from by adding pairs (s,s) for all s in S
    - i.e. adding the diagonal relation
    - follows since a partial order is reflexive
* For product spaces:
  + if or there exists i>0 such that for all and
* For strings (of different lengths):
  + let if:
    - either where t = min(m,n)
    - or m < n and
      * i.e. first string is shorter and is a prefix of the second

Hasse diagrams:



* A partially ordered set can be drawn as a digraph
  + has loops at every vertex (reflexive)
  + has transitive edges (but no cycles)
  + has directed (asymmetric) edges
* Hasse diagrams:
  + loops removed
  + transitive edges removed
  + all directions removed (draw pointing upwards)
* Elements (for a partially ordered set (S,):
  + s is a **maximal** element if
    - at the top of the diagram
  + s is a **minimal** element if
    - at the bottom of the diagram
  + Possible existence:
    - s is the **greatest** element if
    - s is the **least** element if
  + A **lattice**: a partially ordered set such that every pair of elements has both a **least upper bound** (lub) and a **greatest lower bound** (glb)
    - called the “join” (s or t) and the “meet” (s and t)