Chapter 0: Math Notation

- (p4). Sets
  - **Sets**, multi-sets, sequences, and **tuples**
    - Objects are members or elements
    - **Membership**: x ε S
    - Set notation: comma-separated list in "{}"
    - **Set notation**: {x| x ε S, has some property}
    - ∀ for all. "∃" there exists
    - **Multiset**: members can be duplicated
    - **Infinite set**: set has infinite # of members
    - **N** = set of natural numbers {1, 2, …}
    - **Z** = set of integers {…, -2, -1, 0, 1, 2, …}
    - The **Empty Set**: Φ has no members (arity = 0)
  - **Sequence** or **tuple** notation: comma-separate list in "()"
    - Number of elements in each tuple: its **arity**
    - **k-tuple** has k elements; 2-tuple = **Ordered Pair** = **Pair**
    - Elements may be repeated
  - Relationships between sets:
    - **Equal, disjoint, subset, proper subset**
  - Set operations: compute new set from 2 or more sets
    - **Union** AUB**, intersection** A∩B**, complementation** A\B
    - **Cartesian/cross product** AxB = {(a,b)|a ε A. b ε B}
    - **Power set** of set A: set of all subsets of A
  - p.5 Venn diagrams

- (p9). **Relation** R over $A_1,..A_n$ is some subset of $A_1 \times ... \times A_n$
  - Also called a **predicate**
  - Write "R(x,y,z)" if tuple (x,y,z) ε R
    - One-place relations called **properties**
      - Positives = {x|xεZ, x>0}
      - Human = {x|x an object, x is human}
      - Successor = {(x,x+1)}
      - > = {(x>y)}
      - AdditionFact = {(x,y,z)|z=x+y}
- Binary relations from a Power Set:
    - ParentOf = {(x,y)|x and y human and x is parent of y}
    - Properties: Assume R from AxA = A^2
      - **Reflexive**: (a,a) in R
      - **Symmetric**: if R(a,b) then R(b,a)
      - **Transitive**: if R(a,b) and R(b,c) then R(a,c)
    - If all 3, then **Equivalence Relation**
      - Two object are "equivalent" in some sense)
      - A = P_1 U P_2 U …P_n where
        - P_i called an **Equivalence Class**
        - P_i and P_j all disjoint
        - P_i = set of all elements x, y such that R(x,y)
      - E.g. A=Z and R = {(x,y)|x mod 3 = y mod 3}
        - P_0 = {0, 3, 6 , 9, 12, …}
        - P_1 = {1, 4, 7, 10, 13, …}

- P_2 = {2, 5, 8, 11, 14, …}

- **Transitive closure**: computation of equivalence class
  - Start with some element x in class
  - Add in all elements y such that R(x,y)
  - Repeat until exhausted
- **Function f**: related to binary relation F over AxB where
  - for all a in A there is exactly 1 b in B such that F(a,b)
  - Set A called **Domain** and set B called **Range**
  - Written f: A → B
  - Considered a **mapping** from **argument** a to **result** b
  - Notation: f(a) "stands for" object b such that F(a,b) is true
  - Argument and/or result may be tuples
  - Examples page 8 &9
- **Computation**: given an a, find f(a)
  - Also called **function evaluation** or **application**
- Types of functions:
  - **Total**: for each a, there is some b such that F(a,b) or f(a)=b
  - **Partial**: there is some a with no b such that F(a,b) or f(a)=b
  - **Injective or one-to-one**: f(a) = f(b) iff a = b
  - **Surjective or onto**: for each b there is some a where f(a) = b
  - **Bijective**: both above
  - If A and B overlap, a is a **fixed point** if f(a) = a
  - f and g **composable** if f:A→B and g:B→C.
    - Can write g(f(a))

- Since functions are sets, we can define functions that have domains and ranges of functions
  - Functions are first class objects
  - Define **composition function** ∘: (A→B)x(B→C) → (A→C)
    - ∘(g, f) = h, where h:A→C and h(a) = g(f(a))
- Notation for **binary functions** (argument is 2-tuple)
  - **Prefix** f(a,b), **infix** a f b, **postfix** a b f
  - **Commutativity**: f(a, b) = f(b, a)
  - **Associativity**: f(a,f(b,c)) = f(f(a,b),c)
  - i is **identity element** if f(i,x) = f(x,i) = x
- **Predicate**: function whose range is {true, false}
  - Equivalent to relation over domain
- **Curry function** ‘: ((A_1xA_2x…A_n)→B) → ((A_2x…A_n)→B)
  - Where ((‘f)(a_1)) = g_{a1} where g_{a1}(a_2, …a_n) = f(a_1, a_2, …a_n)

- (p.10). **Graphs**
  - **Vertices** and **edges** as sets
  - **Degree**
  - Labelled graph
  - **Subgraph**
  - **Path, simple path, cycle, simple cycle**
  - **Connected graph**
  - **Tree**
  - **Directed graph**
    - **in-degree, out-degree**
    - **Directed path**
    - **Strongly connected**
  - Graph = binary relation
- (p. 14): **Boolean Logic**
  - Functions with domains and ranges from {0, 1}
  - And, or, exclusive or, equality, implication

- (p. 13). **Strings and Languages**
  - **Alphabet** = set of **symbols** typically written as ∑
  - **String** over an alphabet: sequence of symbols
    - **Length**: # of symbols in string
    - **Empty string ε:** string of no symbols
  - **Reverse** of a string = string with symbols in reverse order
  - **Substring of string w:** string that appears within string w
  - **Concatenate(x,y):** string x followed by string y, written xy
  - w^k = concatenation of string w with itself k times
  - **Kleene operators**: unary operators on a string or set of strings
    - **Kleene Star**: w* = { ε, w, ww, www, wwww, …..}
      - If W is a set $\{w_1, w_2, ….\}$, W* = set of all 0 or more concatenations of strings from W
    - **Kleene Plus**: w^+ or W^+ - same as * but 1 or more times
  - x is a **prefix** of y if y = xz for some z
    - **proper prefix**: z not ε
  - **string order**
    - **Lexicographic**: familiar dictionary order
    - **Shortlex or string order**: same as above but short strings first
  - **Language**: set of strings formed in a particular way
    - **Grammar**: set of rules defining the valid strings
    - **Prefix free**: no member is proper prefix of another

- (p.102) **BNF** (Backus Normal Form)
  - Language for describing common grammar rules
  - Set of **substitution rules** (or **productions**)
  - **Nonterminal**: name for a subset of strings that have some particular structure
    - Written as “<” name of nonterminal class “>”
    - E.g. <number>
  - Each **rule** of form “LHS -> RHS”
    - LHS = “left hand side” = name of a nonterminal
    - RHS = “right hand side” = expression on how to concatenate strings in a valid fashion
    - Meaning: if you see a string as defined on right, you can call it a string of type named on left
    - Multiple rules can have same LHS
  - RHS may be > one string expressions separated by “|”
    - Meaning: any of the expressions works
  - A single RHS string expression
    - Concatenation of symbols from alphabet or nonterminals
    - May use Kleene operators * or $^+$
      - Applied to either a string or a nonterminal
    - May be recursive, i.e. may use nonterminal from LHS
  - Example simple sentences: page 103
  - Example simple expressions: page 105

- (p. 17): **Definitions, Theorems, Proofs**
  - **Definition**: description of object or set of objects
  - **Mathematical Statement**: expresses that some objects have certain properties
  - **Proof**: logical argument that a statement is true
  - **Theorem**: statement that has been proven true
    - **Lemma**: proved statement used in bigger proof
    - **Corollary**: statement that can be proven easily once some other statement is proven
  - (p. 18): composition of statements
    - **Implication**: if P then Q, or “Q if P”, written P => Q
    - **Equivalence**: P iff Q, written P ⇔ Q
  - **Inferences**: showing that some statement is true from some others
    - **Forward Inference**: given that statement P=>Q is true
      - If you can prove statement P is true
      - Then you can say Q is true
    - **Backwards Inference**: given statement P=>Q
      - If you can prove Q is false
      - Then you can say P must be false
  - Examples: p. 18 & p. 20

- P.21. **Proof Types**
  - **By construction**: useful in "for all x ∃y P(x,y)"
    - Demonstrate for any x how to construct the object y
    - Example p. 21, Theorem 0.22
  - **By Contradiction**: Want to prove some statement Q is true
    - Assume <u>opposite</u> of desired statement is false and show that this leads to a contradiction
      - And thus assumption that Q is false must be false
        - i.e. Q must be true
    - Also known as indirect proof
    - (p.22) prove that sqrt(2) is irrational
      - Assume opposite, i.e. sqrt(2) is rational = m/n
        - m and n have no common multiples
        - <u>either</u> m <u>or</u> n must be odd
      - Then n*sqrt(2) = m
      - Then $n^2 2 = m^2$
      - Thus m^2 is even
      - Thus m must be even (square of odd always odd)
      - Thus m = 2k, or $n^2 2 = (2k)^2 = 4k^2$
      - Thus $n^2 = 2k^2$
      - Thus n must also be even
      - But then both m and n must be even! Contradiction!
      - Thus sqrt(2) cannot be rational

- **By Induction**: useful to show for all x in set X, P(x) is true, and elements of X can be placed in some order x_1, …x_k, …
  - 3 step process
    - **Basis Step**: prove $P(x_1)$ is true
    - **State the Induction Hypothesis**: $P(x_k) \Rightarrow P(x_{k+1})$ for all k
      - i.e. what we are trying to prove is that if we assume $P(x_k)$ is true, then $P(x_{k+1})$ must also be true
    - **Induction Step**: Prove Induction Hypothesis
      - Typically by assuming $P(x_k)$ is true
  - If induction step is proven true
    - And we prove $P(x_1)$ is true
    - Then $P(x_2)$ is true because $P(x_1)$ is
    - Then $P(x_3)$ is true because $P(x_4)$ is
    - Then …
  - Example 1+2+3+ … n = n(n+1)/2
  - (p. 24) example of mortgage calculation where
    - P = original principal
    - t = number of months of loan
    - P_t = loan remaining after t months
    - M = monthly interest rate percentage + 1
    - Y = monthly mortgage payment