**Logic Language Concepts**

* Describe what we are looking for and have language find it out
* Program attempts to imply that the goal is true after given a goal
* Define what something is … The GCD of A and B is G
  + It is true that G is the GCF if …
  + Define what is true about something
* Technically, no input/output in these languages



* Going down line by line to see if statement is true and running the operation if it is

**Horn Clause**

* Have a head and a body
* h <- B1, B2, …, Bn [H, if B1, and B2 , and … Bn are true]
* Resolution: cancel like terms to combine existing statements to simplify
* Unification: taking variables and give them values that make the statement true

**Prolog**

* It is an interpretive language that runs on a database of clauses
* Terms: constants, variables, structures
* Constant: atoms or number
* Structure: logical predicate or data structure
* Atom: lowercase (treated as a literal value)
* Variable: uppercase (treated as a variable)
  + Takes the value that makes statement true
  + Scope of a variable is limited to the clause it is defined in
* Uses dynamic type checking to be what is needed for the statement to be true
* Structures: functor and list of arguments
* Clauses end with period (.)

**Prolog Symbols**

* :- means implication
* , means and
* ?- means query – asks for a value or to check if statement is true
* ; means next solution – to get more answers
* \+ means not

**Prolog Notes**

* Variables in head of horn clause are universally quantified
* Variables not in head or horn clause (right of :-) are existentially quantified
* Empty left side; query or goal
* Ordering matters (quicker you can say no speeds up process)

**Resolution/Unification**

* If C1 and C2 are horn clauses, if head of C1 matches a term in C2, C1’s body can be placed in C2’s body
* C1 = x > 0, x < 10 and C2 = C1, 0 is mod (x,2)
* C2 = x> 0 , x < 10 , 0 is mod (x,2)



* + takes means person on left takes class on right
  + want to see if x and y are classmates
  + if x takes z and y takes z, we can conclude x and y are classmates
  + only need to find one z that is taken by both x and y
  + To load and run: `pl` `[ex1108].` `classmates(jane\_doe,ajit\_chandra).`
  + `classmates(jane\_doe,Classmate).` returns all the classmates in the variable Classmate that make this statement true.
    - Need to use semicolon to ask if there are more options
* Unification: pattern matching
* Instantiated variables: variables given values after unification have something to compare against
  + Constants unify with itself
  + Structures unify iff everything agrees
  + Variables unify with anything if they don’t already have a value
* Equality if unifiability



* Arithmetic is a predicate, not a function
* When comparing numerically, use ‘is’
  + A = 1+1, B=2, A=B. # false because A is an expression
  + A is 1, B = 2, A = B. # true

**Search and Execution Order**

* Forward chaining: starting with what you know and hopefully finish what you hope to know
* Backward chaining: start with what you hope to know and work backwards to get back to what you knew
* Many more rules than facts: forward chaining is more efficient
  + only so many ways to go if tightly constrained to rules
* Prolog uses backwards chaining enables to explore real possibilities more quickly
* Terms on right are unified with heads of other clauses and uses depth-first search
* If search can’t continue but cannot move forward after removing all possibilities, then need to backtrack and undo last step and find alternative
* This is implemented as a single stack
* Achieved subgoals kept on stack (in case of backtracking)
* consider clauses from first-to-last (top to bottom)
  + Order quicker things to top



**Control Flow**

* Cut(!): Cut off from multiple evaluation
  + When you want to lock in a value of a variable to never find another alternative, cut
  + between(1,10,X),0 is mod(X,2). vs between(1,10,X),0 is mod(X,2),!.
    - only finds the first value that makes it true (2)
* Stay away from loops