**Predicate Logic**

* **Syllogism** – identified correct argument
* Predicate logic allows us to refer to values, a claim about a value, and relations between values
  + E.g. Billy is a child
    - Billy = value; child = attribute of Billy
  + E.g. Billy likes ice cream
    - Likes = relation (predicate)
    - Ice cream = value that Billy likes
* **Predicate** – a symbol denoting the meaning of an attribute of a value or of a relationship b/t two or more values
  + Returns true or false when applied to values (can be used as a prime proposition)
  + Unary predicate – takes a single value as argument
  + Binary predicate – takes two values as arguments
  + N-ary predicate – takes n values as arguments
  + E.g. Barbara plays the piano – plays(Barbara, piano)
    - Plays(x, y) means x plays y
  + E.g. John is happy if John visits Vancouver
    - Visits(John, Vancouver) ⇒ happy(John)
  + P(X) – “P is X”
  + Q(X, Y) – “X is in Q relation with Y”
* **Constant** – a symbol denoting a particular value
* **Variable** – a symbol where a value can be substituted
* **Quantifiers**
  + Universal quantification – ∀ − forall – “for all”, “every”
  + Existential quantification – ∃ − exists – “some”, “there exists” (at least one)
  + ⋅ − “such that”
  + E.g. every child likes Mickey Mouse
    - ∀x ⋅ child(x) ⇒ likes(x, Mickey Mouse)
  + Laws:
    - ¬(∀x ⋅ P(x)) ↔ ∃x ⋅ ¬P(x)
    - ¬(∃x ⋅ P(x)) ↔ ∀x ⋅ ¬P(x)
* **Function** – returns a value, not true/false
  + The value returned is unique
  + E.g. Mary’s age is less than 20 – Mary can only have one age
    - Age(Mary) < 20
  + E.g. Eunsuk was born north of Toronto – Eunsuk can only have one birthplace
    - NorthOf(birthplace(Eunsuk), Toronto)
* **Predicate logic – syntax**
* **Alphabet** – the syntax of predicate logic consists of:
  + Constants, variables, function & predicate symbols, logical connectives, quantifiers, punctuation, brackets
* **Terms**
  + Represent values
  + Every constant & variable is a term
  + A function of terms is a term
* **Well-formed formulas**
  + Represent truth values
  + Atomic formula – a predicate applied to terms
  + Formulas with logical connectives (¬P, P ∧ Q etc.) are formulas
  + If P is a formula and x is a variable, ∀x ⋅ P and ∃x ⋅ P are formulas
* E.g. everyone doesn’t like something
  + ∀x ⋅ ∃y ⋅ ¬likes(x, y) – x = a person, y = a thing
* E.g. no one likes everything
  + ¬(∃x ⋅ ∀y ⋅ likes(x, y))
* E.g. b(x) = x is a bicycle, g(x) = x is in a garage
  + All bicycles are in a garage
    - ∀x ⋅ b(x) ⇒ g(x)
  + All things are bicycles in a garage
    - ∀x ⋅ b(x) ∧ g(x)
  + Some bicycles are in a garage
    - ∃x ⋅ b(x) ∧ g(x)
  + Something in a garage is a bicycle
    - ∃x ⋅ b(x) ∧ g(x)
* ∀x ⋅ ∃y ⋅ P(x, y) is not the same as ∀y ⋅ ∃x ⋅ P(x, y)
* **Scope of quantifier** – assumed to extend to the right end of the formula
  + A scope is only stopped by a right bracket
  + A variable is bound to the closest quantifier to the left whose scope it’s in
  + All variable occurrences bound to the same quantifier represent the same value
  + A variable is free if it’s not within the scope of any quantifier
  + A formula is closed if it contains no free variables
  + E.g. ∀x ⋅ (∃y ⋅ p(x, y) ∧ ∃y ⋅ q(y)) ∧ r(x, y)
* Formalizing sentences
  + Look for: logical connectives → quantifiers → constants → functions (& the values they apply to) → predicates (& the values they apply to)
  + E.g. all students who like software engineering also like magic
    - ∀x ⋅ (student(x) ∧ likes(x, SE)) ⇒ likes(x, logic)
* Types
  + Type – a set of values describing the possible values of a variable
    - E.g. ∀x: R ⋅ ∃y: R ⋅ x ≤ y – where R is a type
  + ∀x : Q ⋅ P(x) = ∀x ⋅ Q(x) ⇒ P(x)
  + ∃x : Q ⋅ P(x) = ∃x ⋅ Q(x) ∧ P(x)
* Syntactic shortcuts
  + ∀x : Person ⋅ ∀y : Person = ∀x, y : Person
  + ∃x : Person ⋅ ∃y : Place = ∃x : Person, y : Place
* **Predicate – semantics**
* Domain of discourse (D) – set of values (other than Tr = {T, F})
* Terms → values in the domain
* Formulas → truth values
* Interpretation – describes the meaning of a predicate logic formula
  + A non-empty domain – a set of distinct values
  + Constants → (mapped to) values in domain
  + Functions → take arguments from the domain and return values in the domain
  + Predicates → take arguments from the domain and return values of Tr
  + Different constants/functions/predicates symbols in the formula can map to the same constants/functions/predicates (can have the same meanings)
  + Use different names for predicates/functions in the syntax and semantics
  + ∀ − formula must be true for all substitutions of a value in the domain
    - i.e. conjunction of formula applied to each value
  + ∃ − formula must be true for some substitutions of a value in the domain
    - i.e. disjunction of formula applied to each value
  + Ex: ∀x . b(x) ⇒ g(x)
    - Given an interpretation:
      * Domain = {A, B, C}
      * Mapping:
        + Syntax – b(.)
        + Meaning – bike(A) := T, bike(B) := T, bike(C) := F
        + Syntax – g(.)
        + Meaning – garage(A) := T, garage(B) := F, garage(C) := F
    - [∀x . b(x) ⇒ g(x)]
    - = (bike(A) IMP garage(A)) AND (bike(B) IMP garage (B)) AND (bike(C) IMP garage(C))
    - = (T IMP T) AND …
    - = F
  + Ex: ∀x . ∃y . p(x) ⇔ ¬p(y)
    - Interpretation:
      * D = {d1, d2}
      * Syntax – p(.)
      * Meaning – P(d1) := T, P(d2) := F
    - [∀x . ∃y . p(x) ⇔ ¬p(y)]
    - = [∃y . p(^d1) ⇔ ¬p(y)] AND [∃y . p(^d2) ⇔ ¬p(y)]
    - = ([p(^d1) ⇔ ¬p(^d1)] OR [p(^d1) ⇔ ¬p(^d2)]) AND ([p(^d2) ⇔ ¬p(^d1)] OR [p(^d2) ⇔ ¬p(^d2)])
    - = …
  + ^ − escape character
    - Used to substitute domain (semantics) values into syntax
  + Use “…” for infinite domain
    - E.g. () AND () AND …