Chapter 8

- Pros and Cons of Propositional Logic:
  - Pros:
    - A **declarative** language that allows to represent **partial**, conjunctive, disjunctive, and negated knowledge.
    - Meaning is **context-independent** – unlike natural language.
    - Has a **sound**, **complete** inference procedures.
  - Cons:
    - **Very limited expressive power** – unlike natural language.
    - **Lack of variables** prevents stating more **general rules**.
    - Changing of the knowledge base over time is difficult to represent.
- First-Order Logic (FOL):
  - Also known as **Predicate Logic** or **Predicate Calculus**.
  - A declarative language (like the propositional logic) and its semantics is based on a truth relation.
  - Greater expressive power than propositional logic as it can represent general laws or rules.
  - It can also express facts about some or all the objects in the universe.
  - Like natural language assumes the world contains:
    - Objects: it corresponds to English **nouns**.
    - Relations or Predicates
      - It corresponds to English **verbs**.
      - Can be **unary** relations or properties.
    - Functions: Arguments of each are **objects**; the return of each is only one **object**.
- Constants, Functions and Predicates:
  - **Constant** (starts with capital letter) symbols represent **objects** in the world.
  - **Functions** symbols stand for functions (maps a tuple of objects to an **object**)
  - **Predicate** symbols stand for relations (maps a tuple of objects to a **truth-value**)
  - Each function and predicate symbol come with an **arity** that fixes the number of its arguments.
  - Every model consists of a set of objects and an **interpretation** to determine if any given sentence is true or false.
  - Because number of possible models is **unbounded**, checking entailment by enumeration of all possible models is not **feasible** – unlike propositional logic.
- Components of First-Order Logic:
  - Term:
    - A logical expression that refers to an **object** (real individual).
    - Can be a constant symbol, a variable symbol, or an n-place function of n terms.

        - A term with no variables is called a **ground term**.
    - Atomic Sentence:
        - An n-place **predicate** of n ground terms – without variables.
    - Complex Sentence:
        - Atomic sentences + logical connectives (¬,∧,v,⇨,⇔).
    - Quantifiers:
        - Universal quantifier (∀) “for all”
            - The expression is true for **every** possible value of the variable.
            - ∀x P(x) means that P is true for all values of x in the domain associated with that variable.
        - Existential quantifier (∃) “there exists.”
            - The expression is true for at least **one** value of the variable.
            - ∃x P(x) means that P is true for some value of x in the domain associated with that variable.
    - Nested quantifiers:
        - Switching the order of the **different** quantifiers **does** change the meaning.
            - Everyone likes some kind of food: ∀x ∃y food(x) ∧ likes(x, y)
            - There is a kind of food that everyone likes: ∃y ∀x food(x) ∧ likes(x, y)
        - Always use different variable names with nested quantifiers.
    - Connections between quantifiers:
        - The two quantifiers are actually closely connected with each other, through negation using the De Morgan rules
            - ∀x ¬P ≡ ¬∃x P
            - ¬∀x P ≡ ∃x ¬P
            - ∀x P ≡ ¬∃x ¬P
            - ¬∀x ¬P ≡ ∃x P
        - ∀ is really a **conjunction** over the universe of objects and ∃ is a **disjunction**.
    - Equality:
        - Can be used to state facts about a given function:
            - Father(John) = Henry
        - Can also be used with negation to insist that two terms are not the same object:
            - ∃x, y Brother(x, Richard) ∧ Brother(y, Richard) ∧ **¬(x = y)**
- **Important examples from slide 16 to slide 27.**
- <u>Using of First-Order Logic:</u>
    - Interacting with first-order knowledge bases:
        - Sentences (called assertions) are added to a knowledge base using **TELL**.
        - Can ask questions (queries) of the knowledge base using **ASK**.

- Types of Answers:
  - Fact is in the KB.
  - Fact is not in the KB.
  - Fact contains variables.
    - **Substitution** or **binding** list for which the fact can be proven.
- ∀ always use the implies (⇨) symbol.
- ∃ always use the or (∧) symbol.

Chapter 9

- Recall 3 cases:
  - Direct matching.
  - Finding a proof (**inference**).
  - Finding a set of bindings (**unification**).
- Universal Instantiation (UI):
  - Every instantiation of a universally quantified sentence is inferred by:

$$\frac{\forall v \quad \alpha}{\textsc{Subst}(\{v/g\}, \alpha)}$$

  - Infer any sentence **a** by substituting any variable **v** by a **ground term g**.
    - Each **ground term** is a term without variables.
    - UI can be applied several times to **add** many new consequence sentences. The new KB is logically equivalent to the old.
- Existential Instantiation (EI):
  - Every instantiation of an existentially quantified sentence is inferred by:

$$\frac{\exists v \quad \alpha}{\textsc{Subst}(\{v/k\}, \alpha)}$$

  - Infer any sentence **a** by substituting any variable **v** by a **constant k** that does not appear elsewhere in the KB. “Skolem Constant”
  - EI can be applied once to **replace** the existential sentence. The new KB is not logically equivalent to the old.
- Unification:
  - It is the process of finding all legal substitutions that make different logical expressions look identical.
  - A key component of all first-order inference algorithms.
  - The **UNIFY** algorithm takes two sentences and returns a unifier for them if one exists:
    - UNIFY(p, q) = θ where SUBST(θ, p) = SUBST(θ, q)

- Generalized Modus Ponens:
    - A general version of modus ponens inference rule for first-order logic that does not require instantiation:

$$\frac{p_1{}',\ \ p_2{}',\ \ \ldots,\ \ p_n{}',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

- First-order Definite Clauses:
    - Either an atomic or is an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal.
- Forward Chaining:
    - Data driven.
    - Main idea:
        - Start with atomic sentences (facts) in the knowledge base.
        - Apply Modus Ponens in the forward direction, by triggering all rules whose premises are satisfied.
        - Adding conclusions of the satisfied rules to the known facts.
        - Repeat the process until the query is answered (assuming that just one answer is required) or no new facts are added.
    - Applied efficiently to **first-order definite clauses**.
    - Properties:
        - **Sound** because every inference is just done by applying Generalized Modus Ponens.
        - **Complete** for first-order definite clauses.
        - **Terminate** for Datalog KB in finite number of iterations.
            - Datalog = first-order definite clauses + without functions.
- Backward Chaining:
    - Goal driven.
    - Main idea:
        - Consider the item to be proven as a goal.
        - Find a rule whose its head is the goal and bindings with it.
        - Apply bindings to the body of that rule and try to prove these body as subgoals in turn.
        - If you prove all the subgoals and increasing the binding set as you go, then you will prove the goal item.
    - Properties:
        - Depth-first recursive proof search: space is linear in size of proof.
        - Incomplete due to infinite loops.
        - Inefficient due to repeated subgoals.
    - Widely used with logic programming (Prolog).