Knowledge Representation

Daniel Gigliotti

1 Knowledge base and inference engine

A **knowledge base** (KB) is a repository of information and facts. The knowledge in a knowledge base is typically represented using some formal language or notation (Propositional Logic, First Order Logic, ...) that allows a system to understand and manipulate the information. Once the system has a knowledge base it can use an **inference engine** to derive new information and make decisions. The inference engine uses various inference rules and logical operations to process the input data and generate new insights. These two components allow us to **tell** the system what it needs to know: it is a **declarative** approach to building an agent.

2 Logics

Logics are formal languages for **representing information** such that **conclusions can be drawn**. **Syntax** defines the legal sentences in the language, **semantics** define the meaning of the sentences, their **truth in a particular world**.

Example: the language of arithmetic.

$$x + 2 >= y \text{ is a sentence.}$$
$$x2 + y >= \text{ is not a sentence.}$$
$$x + 2 >= y \text{ is true in a world where } x = 7, y = 1.$$
$$x + 2 >= y \text{ is false in a world where } x = 0, y = 6.$$

We can extract information from the knowledge base with entailment. **Entailment** means that one sentence follows from another:

$$KB \models \alpha$$

α is true in all worlds where KB is true.

3 Model Checking

The knowledge that is entailed by a KB can be computed by:

$$KB \models \alpha$$

and can be derived by building models and checking whether $M(KB) \subseteq M(\alpha)$. This approach is referred to as **model checking** and it is sort of a proof by exhaustion (enumeration), therefore not always applicable (exponential in n).

4 Inference (Reasoning)

Inference is a broad term that refers to a procedure to derive new knowledge from a knowledge base. We have deductive inference and inductive inference.

5 Deduction

Deduction is a type of inference where we follow strict rules to reach a guaranteed conclusion. For example, if you know that all cats have tails, and you also know that if something has a tail then it is an animal, then you can deduce that cats are animals. **A deduction procedure is a way of computing the knowledge entailed by a KB following strict rules**:

$$KB \vdash_i \alpha$$

denotes that α can be derived from KB by procedure i. Deduction works on formulae by applying **inference rules**.

We use two properties to describe deduction procedures:

- **Soundness**: i is sound if whenever $KB \vdash_i \alpha$, it is also true that $KB \models \alpha$.
- **Completeness**: i is complete if whenever $KB \models \alpha$, it is also true that $KB \vdash_i \alpha$.

6 Induction

Induction is a type of reasoning that involves making generalized conclusions based on specific observations. It's the process of inferring a general pattern from a limited set of examples. Inductive reasoning is used when we want to make educated guesses about things that we haven't directly observed, based on what we have observed.

7 Propositional Logic

Propositional logic is a branch of formal logic that deals with propositions and their relationships using a symbolic notation. Propositions are statements that are either true or false, and propositional logic focuses on studying how these propositions can be combined and manipulated to form more complex statements.

In propositional logic, propositions are represented using variables, such as p, q, or r, and logical operators, such as and ($\wedge$), or ($\vee$), not ($\neg$), implies ($\implies$), and iff ($\iff$). These operators allow us to construct compound propositions by combining simpler propositions.

- $True(\top)$ and $False(\bot)$ are propositional symbols;
- If S is a propositional symbol, S is a sentence;
- If S is a sentence, $\neg S$ is a sentence (negation);
- If S_1 and S_2 are sentences, $S_1 \wedge S_2$ is a sentence (conjunction);
- If S_1 and S_2 are sentences, $S_1 \vee S_2$ is a sentence (disjunction);
- If S_1 and S_2 are sentences, $S_1 \implies S_2$ is a sentence (implication);
- If S_1 and S_2 are sentences, $S_1 \iff S_2$ is a sentence (biconditional);

In propositional logic:

- We have a **knowledge base** that is a set of sentences expressed in a particular language, subjected to syntactic rules.
- Each sentence is formed by logical variables, constants, and operators.
- We can assign truth values to the variables and define how the logical operators and relations function. This is called **interpretation**. Possible worlds correspond to different interpretations.

$$I_1 = \{\alpha = \top, \beta = \bot, \gamma = \bot\}$$
$$I_2 = \{\alpha = \bot, \beta = \bot, \gamma = \top\}$$

- Given an interpretation, the evaluation $I \models F$ means that F is true when the variables have the values of I.
- If $I \models F$, we say that I is a **model** of F.
- A knowledge base is said to be **consistent** if there is at least a model that satisfies it.
- $M(F)$ is the set of all models of F.
- The following statement holds:

$$KB \models \alpha \iff M(KB) \subseteq M(\alpha)$$

- Two sentences are **logically equivalent** iff true in the same models:

$$\alpha \equiv \beta \iff \alpha \models \beta \land \beta \models \alpha$$

Truth tables for connectives

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \implies Q$	$P \iff Q$
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Logical equivalence table

$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$	Commutativity of $\wedge$
$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$	Commutativity of $\vee$
$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$	Associativity of $\wedge$
$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$	Associativity of $\vee$
$\neg(\neg\alpha) \equiv \alpha$	Double-Negation Elimination
$\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$	Implication
$\alpha \Leftrightarrow \beta \equiv (\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$	Biconditional Elimination
$\neg(\alpha \wedge \beta) \equiv (\neg\alpha) \vee (\neg\beta)$	De Morgan on $\wedge$
$\neg(\alpha \vee \beta) \equiv (\neg\alpha) \wedge (\neg\beta)$	De Morgan on $\vee$
$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$	Distributivity of $\wedge$ over $\vee$
$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$	Distributivity of $\vee$ over $\wedge$

Validity and satisfiability

- A sentence is **valid** if it is **true in all interpretations** (it has infinite models).
$$A \vee \neg A, A \implies A, \top, ...$$
- A sentence is **satisfiable** if it is **true in some interpretations** (it has at least a model).
$$A \vee B$$
- A sentence is **unsatisfiable** (inconsistent) if it is **never true** (it has no model).
$$A \wedge \neg A$$
- α is valid if $\neg\alpha$ is unsatisfiable.
- α is satisfiable if $\neg\alpha$ is not valid.

8 Clauses

- A literal is a variable or the negation of a variable.
- **A clause is a disjunction of literals**.

$$L_1 \vee L_2 \vee ... \vee L_n$$

- A clause is said to be a **Definite clause** if it contains **exactly one positive literal**.

$$(\neg L_1 \vee \neg L_2 \vee L_3)$$

- A clause is said to be a **Horn clause** if it contains **at most one positive literal** ($n <= 1$).

$$(\neg L_1 \vee \neg L_2 \vee L_3)$$

but also:

$$(\neg L_1 \vee \neg L_2 \vee \neg L_3)$$

A KB is in Horn form if it is a conjunction of Horn clauses.

- A clause is said to be a **Goal clause** if it contains **no positive literal**.

$$(\neg L_1 \vee \neg L_2 \vee \neg L_3)$$

- Every formula in the KB can be rewritten in **conjunctive normal form** (CNF), that is the conjunction of clauses.

Knowledge base in conjunctive normal form:

$$KB = \{\{A, \neg B, \neg C\}, \{\neg A\}\} \iff (A \vee \neg B \vee C) \wedge (\neg A)$$

- We can use **alpha - beta formulas** to separate cunjunctions, disjunctions and implications into clauses:

$\wedge$ *based*		$\vee$ *based*	
$C = \{\alpha\}$	$C_1 = \{\alpha_1\}, C_2 = \{\alpha_2\}$	$C = \{\beta\}$	$C_1 = \{\beta_1, \beta_2\}$
$\alpha = A \wedge B$	$\alpha_1 = A, \alpha_2 = B$	$\beta = A \vee B$	$\beta_1 = A, \beta_2 = B$
$\alpha = \neg(A \vee B)$	$\alpha_1 = \neg A, \alpha_2 = \neg B$	$\beta = A \implies B$	$\beta_1 = \neg A, \beta_2 = B$
$\alpha = \neg(A \implies B)$	$\alpha_1 = A, \alpha_2 = \neg B$	$\beta = \neg(A \wedge B)$	$\beta_1 = \neg A, \beta_2 = \neg B$

- Example of CNF conversion without alpha-beta rules:

  1. $A \iff (B \vee C)$
  2. Use biconditional elimination and get:
     $(A \implies (B \vee C)) \wedge ((B \vee C) \implies A)$
  3. Use implication elimination ($\alpha \implies \beta \equiv \neg\alpha \vee \beta$) and get:
     $(\neg A \vee B \vee C) \wedge (\neg(B \vee C) \vee A)$
  4. Move $\neg$ inwards:
     $(\neg A \vee B \vee C) \wedge ((\neg B \wedge \neg C) \vee A)$
  5. Use the distributivity of $\vee$ over $\wedge$:
     $(\neg A \vee B \vee C) \wedge (\neg B \vee A) \wedge (\neg C \vee A)$
     We have a conjunctive normal form

- Example of CNF conversion with alpha-beta rules:

  1. $(P \implies (Q \implies (S \vee T))) \implies (T \implies Q)$
  2. Use implication elimination and get:
     $\neg(P \implies (Q \implies (S \vee T))) \vee (T \implies Q)$
  3. Use β rule on disjunction:
     $C = \{\beta_1 = \neg(P \implies (Q \implies (S \vee T))), \beta_2 = (T \implies Q)\}$
  4. Use α rule on β_1 (negation of implication) and get:
     $C_1 = \{P, (T \implies Q)\}, C_2 = \{\neg(Q \implies (S \vee T)), (T \implies Q)\}$
  5. Use β rule on C_1:
     $C_1 = \{P, \neg T, Q\}, C_2 = \{\neg(Q \implies (S \vee T)), (T \implies Q)\}$
  6. Use β rule on C_2:
     $C_1 = \{P, \neg T, Q\}, C_2 = \{\neg(Q \implies (S \vee T)), \neg T, Q\}$
  7. Use α rule on C_2:
     $C_1 = \{P, \neg T, Q\}, C_2 = \{Q, \neg T, Q\}, C_3 = \{\neg(S \vee T), \neg T, Q\}$
  8. Use α rule on C_3:
     $C_1 = \{P, \neg T, Q\}, C_2 = \{Q, \neg T, Q\}, C_3 = \{\neg S, \neg T, Q\}, C_4 = \{\neg T, \neg T, Q\}$
  9. Remove double $\neg T$ from C_4:
     $C_1 = \{P, \neg T, Q\}, C_2 = \{Q, \neg T, Q\}, C_3 = \{\neg S, \neg T, Q\}, C_4 = \{\neg T, Q\}$

9 From validity and satisfiability to Inference

- **Deduction theorem**:

$$KB \models \alpha \iff (KB \implies \alpha) \; is \; valid$$

  thus, if $KB \implies \alpha$ is valid, then $KB \models \alpha$

- **Reductio ad absurdum**:

$$KB \models \alpha \iff (KB \wedge \neg\alpha) \; is \; unsatisfiable$$

  thus, if $KB \wedge \neg\alpha$ is unsatisfiable, then $KB \models \alpha$

- There are several ways to prove that a sentence is satisfiable (SAT problem).

10 Proving satisfiability of a sentence

SAT (Boolean Satisfiability Problem) involves determining whether a given Boolean Propositional Logic formula can be satisfied by assigning truth values (true or false) to its variables. The main goal of SAT solvers is to find a valid assignment of truth values to variables that makes the entire formula true. If such an assignment exists, the formula is satisfiable, otherwise, it's unsatisfiable. The formula is usually given in conjunctive normal form (CNF). Multiple solvers exist, among which DPLL and GSAT are the most notorious.

GSAT is a specific heuristic approach used to solve SAT problems. It's a local search algorithm that focuses on finding a satisfying assignment by iteratively flipping the truth values of variables in an attempt to improve the satisfaction of clauses. The algorithm starts with a random or initial assignment and then repeatedly changes the truth value of a variable to maximize the number of satisfied clauses. In GSAT, at each step, the algorithm selects the variable whose change will result in the maximum increase in satisfied clauses. This greedy approach aims to quickly move towards a satisfying assignment by making locally optimal changes. GSAT is incomplete meaning it may fail.

11 Deduction in Propositional Logic

We can use inference rules to derive a proof of the interpretation truthfulness. The idea of finding a proof rather than using model checking is that the proof can ignore irrelevant propositions, no matter how many of them there are.

Deduction theorem: Let KB be a set of formulae. A is derived from KB ($KB \vdash \alpha$) if there exists a sequence of formulae A_1, ..., A_n such that:

- α is A_n;
- for every i between 1 and n, either $A_i \in KB$ or A_i is a **direct derivation** of the formulae in A_1, ..., A_{i-1} resulting from the application of an inference rule.

The sequence A_1, ..., A_n is a proof of α from KB.

Inference rules:

- **Modus Ponens**
$$\frac{\alpha \implies \beta, \alpha}{\beta}$$
- **And Elimination**
$$\frac{\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n}{\alpha_i}$$
- equivalences from the logical equivalence table

12 Forward and Backward Chaining

Forward chaining and backward chaining are two common approaches used in artificial intelligence and knowledge representation to draw conclusions from a knowledge base. Basically they are algorithms that iteratively apply modus ponens to find the goal.

- **forward chaining**, also known as data-driven reasoning, starts with the available facts and uses them to derive new conclusions or make decisions. It works by applying production rules (if-then statements) to the known facts and inferring new facts from them. The process continues iteratively, adding newly derived facts to the list of known facts, and using those facts to derive even more conclusions.

  Think of forward chaining as building a chain of reasoning from the ground up. It's like starting with pieces of information you already have and gradually building a bigger picture by following the rules.

- **backward chaining**, also known as goal-driven reasoning, starts with a specific goal or conclusion that needs to be proven or derived. It works by examining the available rules in reverse, trying to find a sequence of rules that lead from the goal to the known facts. This approach aims to find the necessary conditions for the goal to be true.

  Backward chaining is like working backward from a goal to see what needs to be true in order to achieve it. It's like starting with a question and figuring out the answers step by step.

13 Resolution

Resolution is a powerful inference rule used in propositional logic to derive new logical statements from existing ones. It's a fundamental technique used in automated theorem proving, logic programming, and other areas of artificial intelligence and logic.

The key idea of the propositional resolution is that if we have two clauses C_1, C_2 and one literal L that appears with different sign in both $L \in C_1, \neg L \in C_2$, then we can generate a new clause by joining $C_1 \vee C_2$ and removing $L, \neg L$ from them. This because if a symbol appears with both signs in a clause (for example $C = L_1 \vee L_2 \vee \neg L_1$), then every interpretation given to L_1 is a model. Iterating this procedure we can simplify the knowledge base and derive new information. **Unit resolution is a special case of Resolution**.

Let there be two clauses:

$$L = L_1 \vee L_2 \vee ... \vee L_n = \{L_1, L_2, ..., L_n\}$$

$$P = P_1 \vee P_2 \vee ... \vee P_m = \{P_1, P_2, ..., P_m\}$$

for any n, m.
Let there be a set of two literals, $O = \{O_l, O_p\}$, which appears in both L and P but with opposite sign:

$$O_1 \in L, \neg O_l = O_p \in P$$

We can use resolution to join P and L and remove O from the union:

$$\frac{L \;\; P}{(L \cup P) \setminus O}$$

which will result in this:

$$(L_1 \vee L_2 \vee ... \vee L_n \vee P_1 \vee P_2 \vee ... \vee P_m) \setminus (O_l \vee O_p)$$

Resolution example:

Given the following KB:

$$KB = \{C_1 = \{\neg P, Q\, \neg P\}, C_2 = \{P, \neg L\}\}$$

and the formula $\alpha = \{Q, \neg L\}$, we want to know if $KB \implies \alpha$, that is $(KB \vee \neg\alpha)$ is unsatisfiable, that is $(KB \vee \neg\alpha) \vdash_R \{\}$.

1. Apply refactoring to C_1 to remove double $\neg P$:
   $KB = \{C_1 = \{\neg P, Q\}, C_2 = \{P, \neg L\}\}$

2. Add $\neg\alpha$ to the knowledge base and apply resolution.
   $(KB \vee \neg\alpha) = \{C_1 = \{\neg P, Q\}, C_2 = \{P, \neg L\}, C_3 = \{\neg Q\}, C_4 = \{L\}\}$
   C_3 and C_4 are the result of $\neg\alpha$.

3. Resolve C_1 with C_2:

   $$C_5 = \frac{\{\neg P, Q\}\ \{P, \neg L\}}{\{\neg L, Q\}} = \{\neg L, Q\}$$

   We end up with:
   $C_1 = \{\neg P, Q\}, C_2 = \{P, \neg L\}, C_3 = \{\neg Q\}, C_4 = \{L\}, C_5 = \{\neg L, Q\}$

4. Resolve C_3 and C_5:

   $$C_6 = \frac{\{\neg Q\}\ \{\neg L, Q\}}{\{\neg L\}} = \{\neg L\}$$

   We end up with:
   $C_1 = \{\neg P, Q\}, C_2 = \{P, \neg L\}, C_3 = \{\neg Q\}, C_4 = \{L\}, C_5 = \{\neg L, Q\}, C_6 = \{\neg L\}$

5. Resolve C_4 and C_6:

   $$C_6 = \frac{\{L\}\ \{\neg L\}}{\{\}} = \{\}$$

   We obtain the empty clause $\{\}$. We just demonstrated that $(KB \vee \neg\alpha) \vdash_R \{\}$ and thus $(KB \models \alpha)$ since $(KB \vee \neg\alpha)$ is unsatisfiable.

Exercise 1

Let A, B, C be propositional symbols. Given:

$$KB = \{A \implies C, B \implies C, A \vee B\}$$

tell whether the formula C can be derived from KB using Modus Ponens and Resolution.

- Modus Ponens:
  C cannot be derived using Modus Ponens as we only know that $A \vee B$ is true, but we don't know which one between A and B is true. We can neither apply Modus Ponens to A and $A \implies C$ nor to B and $B \implies C$.

- Resolution:

  $$\{\{\neg A \vee C\}_1, \{\neg B \vee C\}_2 \{A \vee B\}_3, \{\neg C\}_4\}$$

  with $\{\neg C\}_4$ being the negation of the thesis.

  From $\{\}_1$ and $\{\}_3$ we have $\{B \vee C\}_5$.
  From $\{\}_2$ and $\{\}_5$ we have $\{C\}_6$.
  From $\{\}_4$ and $\{\}_6$ we have $\{\}$.

Exercise 2

Consider the following set of sentences:

1. If it rains, then it is wet.
2. If it is wet, then it does not rain.
3. It rains.

(a) Write the corresponding propositional formulae.

(a) $R \implies W$
(b) $W \implies \neg R$
(c) R

(b) Prove, via resolution, that they are inconsistent.

$$\{\{\neg R \vee W\}_1, \{\neg W \vee \neg R\}_2 \{R\}_3\}$$

To prove inconsistency with resolution we just need not to add the negation of the thesis to the KB.

From $\{\}_1$ and $\{\}_2$ we have $\{\neg R\}_4$.
From $\{\}_3$ and $\{\}_4$ we have $\{\}$.

14 First Order Logic

Whereas Propositional Logic assumes the world contains facts, First Order Logic assumes the world contains:

- **Objects**: variables and constants. A set D called domain is the set containing all the objects.
- **Functions** are used to represent operations that map elements from one domain to another. A function assigns a unique output value to each input value.
- **Relations** (predicate symbols) are used to represent sets of ordered tuples of elements from one or more domains. Relations can be unary (relating one element to itself), binary (relating two elements), ternary (relating three elements), and so on. They are often used to express properties, connections, or comparisons between elements. For example, "less than," "equal to," "ancestor of," and "divides" are examples of relations.
- **Propositional connectives** ($\neg, \vee, \wedge$) and quantifiers ($\exists, \forall$).

Terms are built upon variables, constants and function symbols. Examples are $f(x, c),\ d,\ h(x),\ ....$ They result in an element of the domain. **Formulae** are built upon literals, which are predicates applied to terms. Examples are $P(x, f(c, d)),\ P(c) \wedge R(d),\ \exists x P(f(x, c)),\ ...$ They result in true/false.

Formally:

Language

A **structure** $\mathcal{U}$ for the language $\mathcal{L}$ is a pair $\mathcal{U} =< D, I >$ where:

- D is a non empty set called **domain** of $\mathcal{U}$
- I is a function that maps
  - every constant symbol c into an element $c^I \in D$;
  - every n-ary function symbol f into a function $f^I : D^n \to D$;
  - every n-ary predicate symbol p into a n-ary relation $p^n \subseteq D$;

Validity and satisfiability

A formula $A \in \mathcal{L}$ is **valid** iff it is true in every structure $\mathcal{U}$ of $\mathcal{L}$. This is denoted with $\models A$.
A set of formulae Γ is **satisfiable** if there exists a structure $\mathcal{U}$ such that for every $A \in \Gamma$, A is true in $\mathcal{U}$.

Entailment and equivalence

Let Γ be a set of formulae and A a closed formula. Γ logically entails A ($\Gamma \models A$) iff every model of Γ is also a model of A: for every structure $\mathcal{U}$ of the language $\mathcal{L}$ such that Γ is true in $\mathcal{U}$, then A is true in $\mathcal{U}$.

Two formulas P and Q are **semantically (or logically) equivalent** ($P \equiv Q$) if for every structure $\mathcal{U}$ we have that:

$$P \text{ is true in } \mathcal{U} \iff Q \text{ is true in } \mathcal{U}.$$

15 Unification

Unification is the process of making two different logical atomic expressions identical by finding a substitution. $Unify(P, Q)$ takes two atomic (i.e. single predicates) sentences P and Q and returns a substitution that makes P and Q identical.

TODO

16 Inference in First Order Logic

First order inference can be done applying inference rules, among which we focus on:

- Modus Ponens;
- Resolution. We Apply resolution by converting the KB to propositional logic and using propositional inference on the result.

17 Resolution

Resolution is another inference rule. We Apply resolution by converting the KB to propositional logic and using propositional inference on the result. To achieve this we need to transform each sentence in the KB into Conjunctive Normal Form. Fortunately, every sentence in FOL can be converted into an inferentially equivalent sentence in CNF. The difference between propositional CNF and FOL CNF is the presence of the existential quantifier. Universal quantifiers are simply removed.

- **Universal instantiation** is used to remove universal quantifiers:

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

  for any variable v and ground term g.
  Example:

  $$\forall x, y\ Loves(x, y)$$

  We can remove the universal quantifier by substituting the variables with ground terms, like this:

  $$Subst(\{x/Giorgia, y/Daniel\},\ Loves(x, y))\ =\ Loves(Giorgia, Daniel)\ (hopefully)$$

- **Existential instantiation** is used to remove universal quantifiers by replacing the variable with a single constant symbol C that **do not appear elsewhere** in the KB. After applying the EI, the KB will not be logically equivalent to the old one, but will be satisfiable exactly when the original KB is satisfiable (EI preserves satisfiability).

  $$\exists x\ Game(x)\ \wedge\ Play(Daniel, x)$$

  We can substitute like this:

  $$Game(G_1)\ \wedge\ Play(Daniel, G_1)$$

  where G_1 is a generic constant symbol. By the way this is not true because i have to study AI for the exam.

18 Prenex Normal Form

Before proceding with the Skolem Normal Form, we must describe the Prenex Normal Form. As for the CNF that needs the negation symbols $\neg$ to be pushed inward, the PNF wants the quantifier symbols to be moved outwards to the left side of the formula.

$$[\forall x \phi(x) \wedge \forall y \psi(y)] \implies \exists z \rho(z)$$

We transform the above sentence in PNF just by pushing the quantifiers to the left:

$$\forall x \forall y \exists z [[\phi(x) \wedge \psi(y)] \implies \rho(z)]$$

Given a generic formula ϕ the steps for transforming it into PNF are the following:

- Build a formula ϕ' where only $\wedge, \vee, \exists, \forall$ occur, negation is pushed inwards and double negations are eliminated.
- Rename bound variables so that each quantifier uses a different variable (this technique is called **standardize apart**).
- Build the PNF formula moving all the quantifiers to the left.

19 Skolem Normal Form

We can't simply use EI since by doing so we assume there exists an element, which we don't know, that satisfy the existential property. Consider the following sentence:

"Everyone has a heart"

that is translated in FOL as:

$$\forall x[Person(x) \implies \exists y\ Heart(y) \ \wedge \ Has(x, y)]$$

If we use EI we obtain:

$$\forall x[Person(x) \implies Heart(H) \ \wedge \ Has(x, H)]$$

but the sentence becomes:

"Everyone has the heart H"

For this kind of problem we need a function which takes as input a person and returns the person's heart. Let us denote this function as $F(x)$, called **Skolem function**. The above sentence becomes:

$$\forall x[Person(x) \implies Heart(F(x)) \ \wedge \ Has(x, F(x))]$$

20 Conjunctive Normal Form

We obtain the Conjunctive Normal Form from the Skolem Normal Form by dropping the universal quantifier with UI and then proceeding with the usual Propositional Logic rules to simplify the formula.

Transform the following sentence in FOL:

Everyone who loves all animal is loved by someone.

$$\forall x[\forall y\ Animal(y) \implies Loves(x,y)] \implies [\exists y\ Loves(y,x)]$$

The steps for reducing the sentence into CNF are the following:

1. Eliminate implication:

$$\forall x[\neg\forall y \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\ Loves(y,x)]$$

2. Move $\neg$ inwards:

$$\forall x[\exists y \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y Loves(y,x)]$$

$$\forall x[\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]$$

3. Standardize variables (removing variables with the same name):

$$\forall x[\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ Loves(z,x)]$$

   The formula is now in **PNF**.

4. Skolemize to remove the existential quantifier, using two skolem functions $F(x), G(z)$ not present in the KB:

$$\forall x[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)]$$

   The formula is now in **SNF**.

5. Drop universal quantifier using UI:

$$[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$$

6. Distribution over $\wedge, \vee$ to obtain the SNF with two clauses:

$$[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$$

   The formula is now in **CNF**