Chapter 3

First Order Logic

3.1 Introduction

Propositional Logic has a number of benefits (over natural lnguages for example) a representation language:

- Propositional logic is declarative: pieces of syntax correspond to facts
- Propositional logic allows partial/disjunctive/negated information (unlike most data structures and databases)
- Propositional logic is compositional: meaning of$B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$
- Meaning in propositional logic is context-independent (unlike natural language, where meaning depends on context)

PL has however one disadvantage : It does not have enough expressive power: it cannot express **pits cause breezes in adjacent squares** except by writing a sentence for each square. First Order Logic is an extension of PL that has more expressive power.

3.2 Syntax and Semanics

Whereas propositional logic assumes world contains facts,First-order logic (like natural language) assumes the world contains

- Objects: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries : : :
- Relations: red, round, bogus, prime, multistoried : : :, is the brother of, is bigger than, is inside, is part of, has color, occurred after, owns, comes between, : : :
- Functions: father of, best friend, third inning of, one more than, end of

3.2.1 Syntax of FOL

The basic elements of FOL syntax are :

- Constant symbols : KingJohn; 2; UniversityofMaryland; : : :
- Predicate symbols : Brother;>. ...
- Function symbols :Sqrt; LeftLegOf ;...
- Variable symbols x; y; a; b; : : :
- Connectives: $\wedge$,$\vee$,$\neg$,$\Rightarrow$, $\Leftrightarrow$,
- Equality =
- Quantifiers $\forall$, $\exists$
- Punctuation ( ) ,

Aomic sentences follow the rule :

```
Atomic sentence = predicate(term1; : : : ; termn)
or term1 = term2
Term = function(term1; : : : ; termn)
or constant or variable
```

Complex sentences are made from atomic sentences using connectives. If S ,S and S2 are atomic sentences , each of the following is also a sentence:

- $\neg S$
- $S1 \wedge S2$

- $S1 \vee S2$
- $S1 \Rightarrow S2$
- $S1 \Leftrightarrow S2$

3.2.2 Semantics

In FOL a model is a pair $M = (D, I)$ where D is a domain and I is an interpretation. D contains objects (elements of the domain) and relations between them. I specifies referents for

- constant symbols : objects o the domain
- predicate symbols : relations over objects of the domain
- function symbols :functional relations over objects of the domain.

Like in PL , we say that M is a model of α if α is true in model M. In most of the case ,enumerating all the possible models in FOL is not possible. Computing entailment by model checking is not possible in FOL.

3.3 Knowledge Engineering

The general process of knowledge base construction is called **knowledge engineering**. A knowledge enginner is somone who investigates a particular domain, learns what concepts are important in that domain and creates a formal representation of the objects and relations in the domain. Knowledge engineering projects vary in content, scope and difficulty, but all such projects include the following steps:

1. *Identify the task.* The knowledge engineer must delineate the range of questions that the knowledge base will support and the kinds of facts that will be availble for each specific problem instance.
2. *Assemble the relevant knowledge* : The knowledge engineer might already be ana expert in the domain, or might need to work with domain experts to extract what they know. this process is called **knowledge acquisition**. At this stage, the knowledge is not represented formally. The idea is to understand the scope of the knowledge base , as determined by the task and to understand how the domain works.

3. *Decide on a vocabulary of predicates, functions and constants*: the is translate the important domain -level concepts into logic -level names. This involves many questions of knowledge engineering style .Once the choices have been made, the result is a vocabulary that is known as the **ontology** of the domain. The ontology determines what kinds of things exist but does not determine their specific properties and interrelationships.

4. *Encode general knowledge about the domain.* The knowledge engineer writes down the axioms for all the vocabulary terms. This pins down (to the extent possible) the meaning of the terms, enabling the expert to check the content. Often this step reveals misconceptions or gaps in the vocabulary that must be fixed by returning to step 3 and iterating through the process.

5. *Encode a description of the specific problem instance.* If the ontology is well thought out, this step will be easy. It will involve writing simple atomic sentences about instances of concepts that are already part of the ontology. For a logical agent , problem instances are supplied by the sensors, whereas a knowledge base is supplied with additional sentences in the same way that traditional programs are supplied with input data.

6. *Pose queries to the inference procedure and get answers.* This is where the reward is : we can let the inference procedure operate on the axioms and problem -specific facts to derive the facts we are interested in knowing.

7. *Debug the knowledge base.* As the knowledge base is used, missing axioms or weak axioms will be identified and rectified in order for the knowledge base to work as expected by the user.

3.4 Inference in FOL

3.4.1 Universal Instantiation

If $\forall x P(x)$ is true, then P(c) is true, where c is a constant in the domain of x. For example, from $\forall x \quad eats(Ziggy, x)$ we can infer eats(Ziggy, IceCream). The variable symbol can be replaced by any ground term, i.e., any constant symbol or function symbol applied to ground terms only.

3.4.2 Existential Instantiation

From $\exists x P(x)$ infer P(c). For example, from $\exists x \quad eats(Ziggy, x)$ infer eats(Ziggy, Cheese). Note that the variable is replaced by a brand new constant that does not occur in this or any other sentence in the Knowledge Base. In other words, we don't want to accidentally draw other inferences about it by introducing the constant. All we know is there must be some constant that makes this true, so we can introduce a brand new one to stand in for that (unknown) constant.

3.4.3 Generalized Modus Ponens

Generalized Modus Ponens combines And-Introduction, Universal-Elimination, and Modus Ponens Example: from P(c), Q(c), and $\forall x(P(x)^Q(x)) \Rightarrow R(x)$, derive R(c) In general, given atomic sentences P1, P2, ..., PN, and implication sentence $(Q1 \wedge Q2 \wedge \cdots \wedge QN) \Rightarrow R$, where Q1, ..., QN and R are atomic sentences, and subst(Theta, Pi) = subst(Theta, Qi) for i=1,...,N, derive new sentence: subst(Theta, R) subst(Theta, alpha) denotes the result of applying a set of substitutions defined by Theta to the sentence alpha A substitution list Theta = v1/t1, v2/t2, ..., vn/tn means to replace all occurrences of variable symbol vi by term ti. Substitutions are made in left-to-right order in the list. Example: subst(x/IceCream, y/Ziggy, eats(y,x)) = eats(Ziggy, IceCream). In general, Generalized Modus Ponens is not complete in FOL , however, for KB containing only **Horn clauses**, GMP is complete. A Horn clause is a sentence of the form: $(\forall \quad x)(P1(x)^P 2(x) \cdot .. ^P n(x)) \Rightarrow Q(x)$ where there are 0 or more Pi's, and the Pi's and Q are positive (i.e., un-negated) literals Horn clauses represent a subset of the set of sentences representable in FOL. For example, $P(a) \vee Q(a$ is a sentence in FOL but is not a Horn clause. Natural deduction using GMP is complete for KBs containing only Horn clauses. Proofs start with the given axioms/premises in KB, deriving new sentences using GMP until the goal/query sentence is derived. This defines a **forward chaining** inference procedure because it moves "forward" from the KB to the goal.

Example: KB = All cats like fish, cats eat everything they like, and Ziggy is a cat. In FOL, KB =

1. $(\forall x) \quad cat(x) \Rightarrow likes(x, Fish)$
2. $(\forall x)(\forall y)(cat(x) \wedge likes(x, y)) \Rightarrow eats(x, y)$

3. cat(Ziggy)

Goal query: Does Ziggy eat fish?

Proof:

1. Use GMP with (1) and (3) to derive: 4. likes(Ziggy, Fish)
2. Use GMP with (3), (4) and (2) to derive eats(Ziggy, Fish)
3. So, Yes, Ziggy eats fish.

Backward-chaining deduction using GMP is complete for KBs containing only Horn clauses. Proofs start with the goal query, find implications that would allow you to prove it, and then prove each of the antecedents in the implication, continuing to work "backwards" until we get to the axioms, which we know are true.

Example: Does Ziggy eat fish?

To prove eats(Ziggy, Fish), first see if this is known from one of the axioms directly. Here it is not known, so see if there is a Horn clause that has the consequent (i.e., right-hand side) of the implication matching the goal. Here,
Proof:

1. Goal matches RHS of Horn clause (2), so try and prove new sub-goals cat(Ziggy) and likes(Ziggy, Fish) that correspond to the LHS of (2)
2. cat(Ziggy) matches axiom (3), so we've "solved" that sub-goal
3. likes(Ziggy, Fish) matches the RHS of (1), so try and prove cat(Ziggy)
4. cat(Ziggy) matches (as it did earlier) axiom (3), so we've solved this sub-goal
5. There are no unsolved sub-goals, so we're done. Yes, Ziggy eats fish.

3.4.4 Unification

Unification is a "pattern matching" procedure that takes two atomic sentences, called literals, as input, and returns "failure" if they do not match and a substitution list, Theta, if they do match. That is, unify(p,q) = Theta means subst(Theta, p) = subst(Theta, q) for two atomic sentences p and q. Theta is called the most general unifier (mgu). All variables in the given two literals are implicitly universally quantified. To make literals match, replace (universally-quantified) variables by terms. Unification algorithm:

```
procedure unify(p, q, theta)
  Scan p and q left-to-right and find the first corresponding
     terms where p and q "disagree"  ; where p and q not equal
  If there is no disagreement, return theta  ; success
  Let r and s be the terms in p and q, respectively,
     where disagreement first occurs
  If variable(r) then
     theta = union(theta, {r/s})
     unify(subst(theta, p), subst(theta, q), theta)
  else if variable(s) then
     theta = union(theta, {s/r})
     unify(subst(theta, p), subst(theta, q), theta)
  else return "failure"
end
```

Examples:

- Literal1 :parents(x, father(x), mother(Bill))
- Literal 2: parents(Bill, father(Bill), y)
- Result of Unify: $\{x/Bill, y/mother(Bill)\}$
- Literal 1 : parents(x, father(x), mother(Bill))
- Literal 2 : parents(Bill, father(y), z)
- Result of Unify :$\{x/Bill, y/Bill, z/mother(Bill)\}$
- Literal 1: parents(x, father(x), mother(Jane))
- Literal 2 : parents(Bill, father(y), mother(y))

- Result of Unify: Failure

Unify is a linear time algorithm that returns the most general unifier (mgu), i.e., a shortest length substitution list that makes the two literals match. (In general, there is not a unique minimum length substitution list, but unify returns one of those of minimum length.) A variable can never be replaced by a term containing that variable. For example, x/f(x) is illegal. This "occurs check" should be done in the above pseudo-code before making the recursive calls.

3.4.5 Resolution -Refutation

Resolution procedure is a sound and complete inference procedure for FOL Resolution procedure uses a single rule of inference: the Resolution Rule (RR), which is a generalization of the same rule used in PL. Resolution Rule for PL: From sentence P1 v P2 v ... v Pn and sentence $\neg P1$ v Q2 v ... v Qm derive resolvent sentence: P2 v ... v Pn v Q2 v ... v Qm

Examples:

- From P and $\neg P \vee Q$, derive Q (Modus Ponens)
- From$(\neg P \vee Q)$ and $(\neg Q \vee R)$, derive $\neg P \vee R$
- From P and $\neg P$, derive False
- From (P v Q) and $(\neg P \vee \neg Q)$, derive True

Resolution Rule for FOL: Given sentence P1 v ... v Pn and sentence Q1 v ... v Qm where each Pi and Qi is a literal, i.e., a positive or negated predicate symbol with its terms, if Pj and $\neg Qk$ unify with substitution list Theta, then derive the resolvent sentence: subst(Theta, P1 v ... v Pj-1 v Pj+1 v ... v Pn v Q1 v ... Qk-1 v Qk+1 v ... v Qm)
Example From clause P(x, f(a)) v P(x, f(y)) v Q(y) and clause $\neg P(z, f(a)) \vee \neg Q(z)$, derive resolvent clause $P(z, f(y)) v Q(y) \vee \neg Q(z)$ using $Theta = \{x/z\}$

Resolution -refutation is a Proof by contradiction method. Given a consistent set of axioms KB and goal sentence Q, we want to show that $KB \models Q$. This means that every interpretation I that satisfies KB, satisfies Q. But we know that any interpretation I satisfies either Q or $\neg Q$, but not both. Therefore if in fact $KB \models Q$, an interpretation that satisfies KB, satisfies Q and

does not satisfy $\neg Q$. Hence KB union $\{\neg Q\}$ is unsatisfiable, i.e., that it's false under all interpretations. In other words, $(KB \models Q) \Leftrightarrow (KB \wedge \neg Q \models False)$ What's the gain? If KB union $\neg Q$ is unsatisfiable, then some finite subset is unsatisfiable. Resolution procedure can be used to establish that a given sentence Q is entailed by KB; however, it cannot, in general, be used to generate all logical consequences of a set sentences. Also, the resolution procedure cannot be used to prove that Q is not entailed by KB. Resolution procedure won't always give an answer since entailment is only semidecidable. And you can't just run two proofs in parallel, one trying to prove Q and the other trying to prove $\neg Q$, since KB might not entail either one. Algorithm

```
procedure resolution-refutation(KB, Q)
  //KB is a set of consistent, true FOL sentences
 // Q is a goal sentence that we want to derive
// return success if KB |- Q, and failure otherwise

   KB = union(KB, ~Q)
   while false not in KB do
      pick 2 sentences, S1 and S2, in KB that contain
   literals that unify
      if none, return "failure"
      resolvent = resolution-rule(S1, S2)
      KB = union(KB, resolvent)
   return "success"
end
```

Example using PL Sentences

From the sentence **"Heads I win, tails you lose,"** prove that **"I win"**.

First, define the axioms in KB:

1. "Heads I win, tails you lose."
   $(Heads \Rightarrow IWin)$ or, equivalently, $(\neg Heads \vee IWin)$
   $(Tails \Rightarrow YouLose)$or, equivalently,$(\neg Tails \vee YouLose)$

2. Add some general knowledge axioms about coins, winning, and losing:
   $(Heads \vee Tails)$

$(YouLose \Rightarrow IWin)$ or, equivalently, $(\neg YouLose \vee IWin)$ $(IWin \Rightarrow YouLose)$ or, equivalently, $(\neg IWin \vee YouLose)$

Goal : **IWin**.
Proof :

Sentence 1	Sentence 2	Resolvent
$\neg IWin$	$(\neg Heads \vee IWin)$	$\neg Heads$
$\neg Heads$	$Heads \vee Tails$	Tails
Tails	$\neg Tails \vee YouLose$	YouLose
YouLose	$\neg YouLose \vee IWin$	IWin
IWin	$\neg IWin$	False

Resolution rule of inference is only applicable with sentences that are in the form P1 v P2 v ... v Pn, where each Pi is a negated or un-negated predicate and contains functions, constants, and universally quantified variables, so can we convert every FOL sentence into this form?

Steps to convert a sentence to clause form:

1. Eliminate all $\Leftrightarrow$ connectives by replacing each instance of the form $(P \Leftrightarrow Q)$by the equivalent expression $((P \Rightarrow Q) \wedge (Q \Rightarrow P))$

2. Eliminate all$\Rightarrow$ connectives by replacing each instance of the form $(P => Q)$by $(\neg P \vee Q)$

3. Reduce the scope of each negation symbol to a single predicate by applying equivalences such as converting $\neg\neg P$ to P;$\neg(P \vee Q)$ to $\neg P \wedge \neg Q$; $\neg(P \wedge Q)$ to $\neg P \vee \neg Q$; $\neg(\forall x)P$ to $(\exists x)\neg P$, and $\neg(\exists x)P$ to $(\forall x)\neg P$

4. Standardize variables: rename all variables so that each quantifier has its own unique variable name. For example, convert $(\forall x)P(x)$ to $(\forall y)P(y)$ if there is another place where variable x is already used.

5. Eliminate existential quantification by introducing Skolem functions. For example, convert $(\exists x)P(x)$ to P(c) where c is a brand new constant symbol that is not used in any other sentence. c is called a Skolem constant. More generally, if the existential quantifier is within the scope of a universal quantified variable, then introduce a Skolem function that depend on the universally quantified variable. For example, $(\forall x)(\exists y)P(x, y)$ is converted to $(\forall x)P(x, f(x))$. f is called a Skolem function, and must be a brand new function name that does not occur

in any other sentence in the entire KB. Example: $(\forall x)(\exists y)loves(x, y)$ is converted to $(\forall x)loves(x, f(x))$ where in this case f(x) specifies the person that x loves. (If we knew that everyone loved their mother, then f could stand for the mother-of function.

6. Remove universal quantification symbols by first moving them all to the left end and making the scope of each the entire sentence, and then just dropping the "prefix" part. For example, convert $(\forall x)P(x)$ to P(x).

7. Distribute "and" over "or" to get a conjunction of disjunctions called conjunctive normal form. Convert $(P \wedge Q) \vee R$ to$(P \vee R) \wedge (Q \vee R)$, and convert $P \vee Q) \vee R$ to $(P \vee Q \vee R)$.

8. Split each conjunct into a separate clause, which is just a disjunction ("or") of negated and un-negated predicates, called literals.

9. Standardize variables apart again so that each clause contains variable names that do not occur in any other clause.

Convert the sentence $(\forall x)(P(x) \Rightarrow ((\forall y)(P(y) \Rightarrow P(f(x, y))) \wedge \neg(\forall y)(Q(x, y) \Rightarrow P(y))))$

1. Eliminate $\Leftrightarrow$ Nothing to do here.

2. Eliminate $\Rightarrow$
$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x, y))) \wedge \neg(\forall y)(\neg Q(x, y) \vee P(y))))$

3. Reduce scope of negation $(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x, y))) \wedge (\exists y)(Q(x, y) \wedge \neg P(y))))$

4. Standardize variables $(\forall x)(\neg P(x) v((\forall y)(\neg P(y) \vee P(f(x, y))) \wedge (\exists z)(Q(x, z) \wedge \neg P(z))))$

5. Eliminate existential quantification $(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x, y))) \wedge (Q(x, g(x)) \wedge \neg P(g(x)))))$

6. Drop universal quantification symbols $(\neg P(x) \vee ((\neg P(y) \vee P(f(x, y))) \wedge (Q(x, g(x)) \wedge \neg P(g(x)))))$

7. Convert to conjunction of disjunctions $(\neg P(x) \vee \neg P(y) \vee P(f(x, y))) \wedge (\neg P(x) \vee Q(x, g(x))) \wedge (\neg P(x) \vee \neg P(g(x)))$

8. Create separate clauses
   - $\neg P(x) \vee \neg P(y) \vee P(f(x, y))$
   - $P(x) \vee Q(x, g(x))$
   - $P(x) \vee \neg P(g(x))$
9. Standardize variables
   - $\neg P(x) \vee \neg P(y) \vee P(f(x, y))$
   - $\neg P(z) \vee Q(z, g(z))$
   - $\neg P(w) v \neg P(g(w))$

Revised Resolution Refutation Procedure

```
procedure resolution-refutation(KB, Q)
  //KB is a set of consistent, true FOL sentences
   // Q is a goal sentence that we want to derive
  // return success if KB |- Q, and failure otherwise
   KB = union(KB, ~Q)
   KB = clause-form(KB)  ; convert sentences to clause form
   while false not in KB do
     pick 2 clauses, C1 and C2, that contain literals that unify
     if none, return "failure"
     resolvent = resolution-rule(C1, C2)
     KB = union(KB, resolvent)
   return "success"
end
```

Example: Hoofers Club

- **Problem Statement**:

  Tony, Shi-Kuo and Ellen belong to the Hoofers Club. Every member of the Hoofers Club is either a skier or a mountain climber or both. No mountain climber likes rain, and all skiers like snow. Ellen dislikes whatever Tony likes and likes whatever Tony dislikes. Tony likes rain and snow.

- **Query**:
  Is there a member of the Hoofers Club who is a mountain climber but not a skier?

- **Translation into FOL Sentences**

  Let S(x) mean x is a skier, M(x) mean x is a mountain climber, and L(x,y) mean x likes y, where the domain of the first variable is Hoofers Club members, and the domain of the second variable is snow and rain. We can now translate the above English sentences into the following FOL wffs:

  1. $(\forall x)S(x) \vee M(x)$
  2. $\neg(\exists x)M(x) \wedge L(x, Rain)$
  3. $(\forall x)S(x) \Rightarrow L(x, Snow)$
  4. $(\forall y)L(Ellen, y) \Leftrightarrow \neg L(Tony, y)$
  5. L(Tony, Rain)
  6. L(Tony, Snow)
  7. Query: $(\exists x)M(x) \wedge \neg S(x)$
  8. Negation of the Query: $\neg(\exists x)M(x) \wedge \neg S(x)$

- **Conversion to Clause Form**

  1. S(x1) v M(x1)
  2. $\neg M(x2) \vee \neg L(x2, Rain)$
  3. $\neg S(x3) \vee L(x3, Snow)$
  4. $\neg L(Tony, x4) \vee \neg L(Ellen, x4)$
  5. L(Tony, x5) v L(Ellen, x5)
  6. L(Tony, Rain)
  7. L(Tony, Snow)
  8. Negation of the Query: $\neg M(x7) \wedge S(x7)$

- **Resolution refutation proof**

Clause 1	Caluse 2	Resolvent	MGU(i.e Theta)
8	1	9 S(x1)	$\{x7/x1\}$
9	3	10. L(x1, Snow)	$\{x3/x1\}$
10	4	11.$\neg L(Tony, Snow)$	$\{x4/Snow, x1/Ellen\}$
11	7	False	$\{\}$