Unit - 4

First-Order Logic in Artificial intelligence

In the topic of Propositional logic, we have seen that how to represent statements using propositional logic. But unfortunately, in propositional logic, we can only represent the facts, which are either true or false. PL is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power. Consider the following sentence, which we cannot represent using PL logic.

- **"Some humans are intelligent", or**
- **"Sachin likes cricket."**

To represent the above statements, PL logic is not sufficient, so we required some more powerful logic, such as first-order logic.

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic (like natural language) does not only assume that the world contains facts like propositional logic but also assumes the following things in the world:
- **Objects:** A, B, people, numbers, colors, wars, theories, squares, pits, wumpus,
- **Relations: It can be unary relation such as:** red, round, is adjacent, **or n- any relation such as:** the sister of, brother of, has color, comes between
- **Function:** Father of, best friend, third inning of, end of, ...
- As a natural language, first-order logic also has two main parts:
    - **Syntax**
    - **Semantics**

Syntax of First-Order logic:

The syntax of FOL determines which collection of symbols is a logical expression in first- order logic. The basic syntactic elements of first-order logic are symbols. We write statements in short-hand notation in FOL.

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. These sentences are formed from a predicate symbol followed by a parenthesis with a sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2, , term n)**.

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).

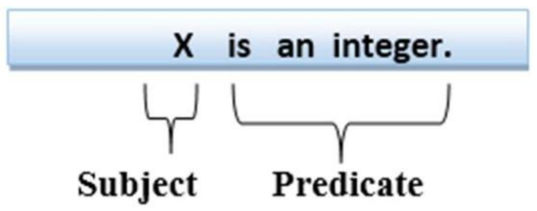
Chinky is a cat: => cat (Chinky).

Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.**Consider the statement: "x is an integer."**, it consists of two parts, the first part x is the subject of the statement and second part "is an integer," is known as a predicate.



Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
- **Universal Quantifier, (for all, everyone, everything)**
- **Existential quantifier, (for some, at least one).**

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

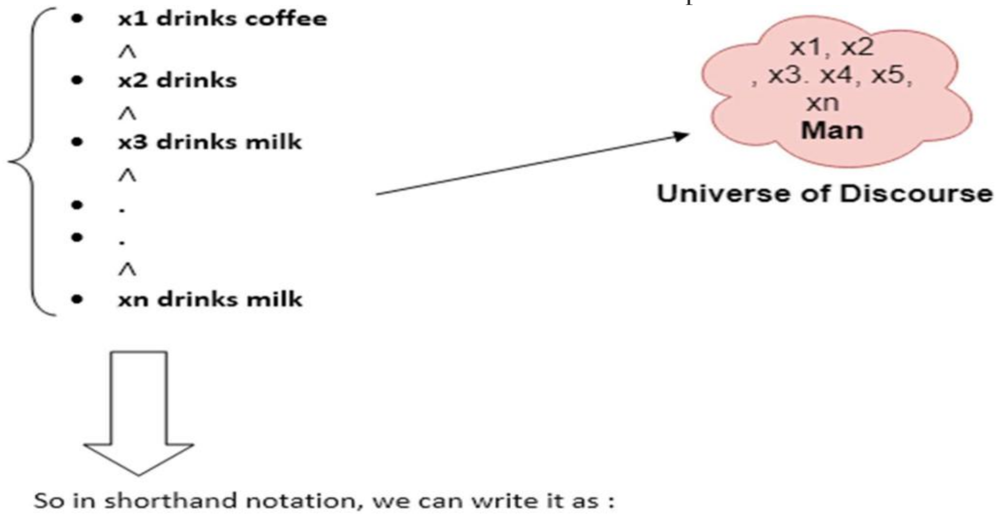
If x is a variable, then ∀x is read as:

- **For all x**
- **For each x**
- **For every x.**

Example:

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:



∀x man(x) → drink (x, coffee).

It will be read as: There are all x where x is a man who drink coffee.

Existential Quantifier:

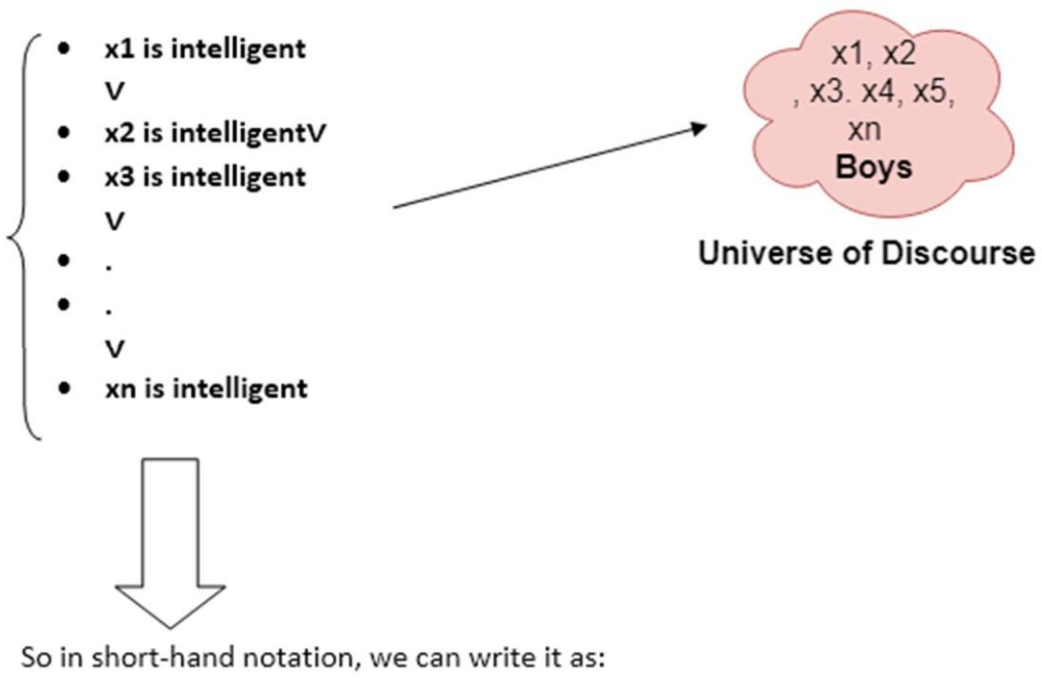
Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something.

It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier.
If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as:

- **There exists a 'x.'**
- **For some 'x.'**
- **For at least one 'x.'**

Example:

Some boys are intelligent.



∃x: boys(x) ∧ intelligent(x)

It will be read as: There are some x where x is a boy who is intelligent.

Points to remember:

- The main connective for universal quantifier ∀ is implication →.

- The main connective for existential quantifier **∃** is and ∧.

Properties of Quantifiers:

- In universal quantifier, ∀x∀y is similar to ∀y∀x.
- In Existential quantifier, ∃x∃y is similar to ∃y∃x.
- ∃x∀y is not similar to ∀y∃x.

Some Examples of FOL using quantifier:

1. **All birds fly.**

In this question the predicate is "**fly(bird)**."

And since there are all birds who fly so it will be represented as follows.

∀x bird(x) →fly(x).

2. **Every man respects his parent.**

In this question, the predicate is "**respect(x, y)," where x=man, and y= parent**.

Since there is every man so will use ∀, and it will be represented as follows:

∀x man(x) → respects (x, parent).

3. **Some boys play cricket.**

In this question, the predicate is "**play(x, y)**," where x= boys, and y= game. Since there are some boys so we will use **∃, and it will be represented as**:

∃x boys(x) → play(x, cricket).

4. **Not all students like both Mathematics and Science.**

In this question, the predicate is "**like(x, y)," where x= student, and y= subject**. Since there are not all students, so we will use ∀ **with negation, so** following representation for this:

¬∀ (x) [student(x) → like(x, Mathematics) ∧ like(x, Science)].

5. **Only one student failed in Mathematics.**

In this question, the predicate is "**failed(x, y)," where x= student, and y= subject**.

Since there is only one student who failed in Mathematics, so we will use following representation for this:

∃(x) [student(x) → failed (x, Mathematics) ∧∀ (y) [¬(x==y) ∧ student(y) →

¬failed (x, Mathematics)].

Free and Bound Variables:

The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:

Free Variable: A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: $\forall x \exists(y)[P(x, y, z)]$, where z is a free variable.

Bound Variable: A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier

Example: $\forall x [A(x) B(y)]$, here x and y are the bound variables.

Knowledge Engineering in First-order logic

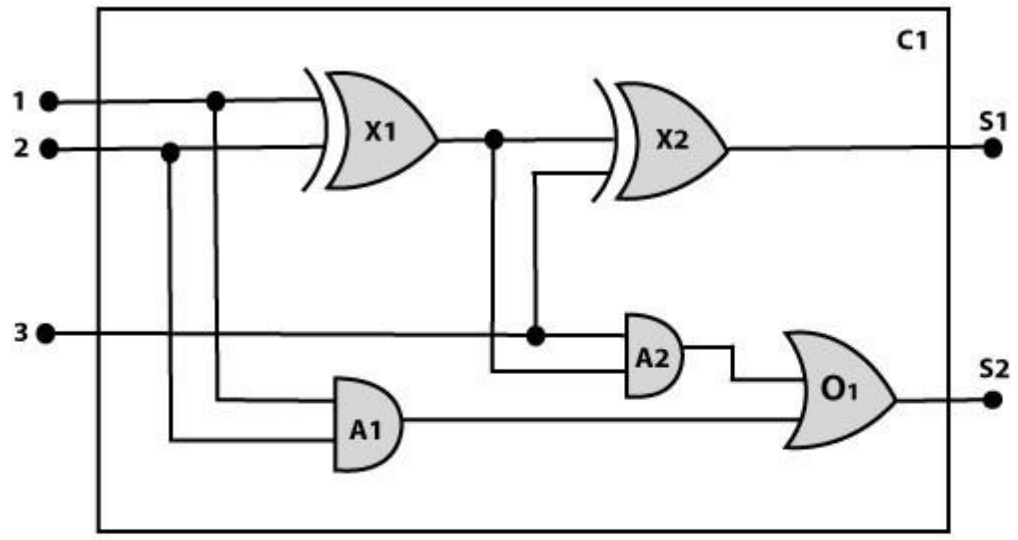
What is knowledge-engineering?

The process of constructing a knowledge-base in first-order logic is called as knowledge- engineering. In **knowledge-engineering**, someone who investigates a particular domain, learns important concept of that domain, and generates a formal representation of the objects, is known as **knowledge engineer**.

In this topic, we will understand the Knowledge engineering process in an electronic circuit domain, which is already familiar. This approach is mainly suitable for creating **special- purpose knowledge base**.

The knowledge-engineering process:

Following are some main steps of the knowledge-engineering process. Using these steps, we will develop a knowledge base which will allow us to reason about digital circuit (**One- bit full adder**) which is given below



1. **Identify the task:**

The first step of the process is to identify the task, and for the digital circuit, there are various reasoning tasks.

At the first level or highest level, we will examine the functionality of the circuit:

- **Does the circuit add properly?**
- **What will be the output of gate A2, if all the inputs are high?**

At the second level, we will examine the circuit structure details such as:

- **Which gate is connected to the first input terminal?**
- **Does the circuit have feedback loops?**

2. **Assemble the relevant knowledge:**

In the second step, we will assemble the relevant knowledge which is required for digital circuits. So for digital circuits, we have the following required knowledge:

- Logic circuits are made up of wires and gates.
- Signal flows through wires to the input terminal of the gate, and each gate produces the corresponding output which flows further.

- In this logic circuit, there are four types of gates used: **AND, OR, XOR, and NOT**.
- All these gates have one output terminal and two input terminals (except NOT gate, it has one input terminal).

3. Decide on vocabulary:

The next step of the process is to select functions, predicate, and constants to represent the circuits, terminals, signals, and gates.
Firstly we will distinguish the gates from each other and from other objects. Each gate is represented as an object which is named by a constant, such as, **Gate(X1)**. The functionality of each gate is determined by its type, which is taken as constants such as **AND, OR, XOR, or NOT**. Circuits will be identified by a predicate: **Circuit (C1)**.

For the terminal, we will use predicate: **Terminal(x)**.

For gate input, we will use the function **In(1, X1)** for denoting the first input terminal of the gate, and for output terminal we will use **Out (1, X1)**.

The function **Arity(c, i, j)** is used to denote that circuit c has i input, j output.

The connectivity between gates can be represented by predicate **Connect(Out(1, X1), In(1, X1))**.

We use a unary predicate **On (t)**, which is true if the signal at a terminal is on.

4. Encode general knowledge about the domain:

To encode the general knowledge about the logic circuit, we need some following rules:

- If two terminals are connected then they have the same input signal, it can be represented as:

1. ∀ t1, t2 Terminal (t1) ∧ Terminal (t2) ∧ Connect (t1, t2) → Signal (t1) = Signal (2).

- Signal at every terminal will have either value 0 or 1, it will be represented as:

1. ∀ t Terminal (t) →Signal (t) = 1 ∨Signal (t) = 0.

- Connect predicates are commutative:

1. ∀ t1, t2 Connect(t1, t2) → Connect (t2, t1).

- Representation of types of gates:

1. $\forall$ g Gate(g) $\land$ r = Type(g) $\rightarrow$ r = OR $\lor$r = AND $\lor$r = XOR $\lor$r = NOT.
    - Output of AND gate will be zero if and only if any of its input is zero

$\forall$ g Gate(g) $\land$ Type(g) = AND $\rightarrow$Signal (Out(1, g))= 0 $\Leftrightarrow$ $\exists$n Signal (In(n, g))= 0.

- Output of OR gate is 1 if and only if any of its input

1. $\forall$ g Gate(g) $\land$ Type(g) = OR $\rightarrow$ Signal (Out(1, g))= 1 $\Leftrightarrow$ $\exists$n Signal (In(n, g))= 1
    - Output of XOR gate is 1 if and only if its inputs are different:

1. $\forall$ g Gate(g) $\land$ Type(g) = XOR $\rightarrow$ Signal (Out(1, g)) = 1 $\Leftrightarrow$ Signal (In(1, g)) $\neq$ Signal (In(2, g)).

    - Output of NOT gate is invert of its input:

1. $\forall$ g Gate(g) $\land$ Type(g) = NOT $\rightarrow$ Signal (In(1, g)) $\neq$ Signal (Out(1, g)).
    - All the gates in the above circuit have two inputs and one output (except NOT gate).

1. $\forall$ g Gate(g) $\land$ Type(g) = NOT $\rightarrow$ Arity(g, 1, 1)
2. $\forall$ g Gate(g) $\land$ r =Type(g) $\land$ (r= AND $\lor$r= OR $\lor$r= XOR) $\rightarrow$ Arity (g, 2, 1).
    - All gates are logic circuits:

1. $\forall$ g Gate(g) $\rightarrow$ Circuit (g).

5. Encode a description of the problem instance:

Now we encode problem of circuit C1, firstly we categorize the circuit and its gate components. This step is easy if ontology about the problem is already thought.

This step involves the writing simple atomics sentences of instances of concepts, which is known as ontology.

For the given circuit C1, we can encode the problem instance in atomic sentences as below:

Since in the circuit there are two XOR, two AND, and one OR gate so atomic sentences for these gates will be:

1. For XOR gate: Type(x1)= XOR, Type(X2) = XOR
2. For AND gate: Type(A1) = AND, Type(A2)= AND
3. For OR gate: Type (O1) = OR. And then represent the connections between all the gates.

6. Pose queries to the inference procedure and get answers:

In this step, we will find all the possible set of values of all the terminal for the adder circuit. The first query will be:

What should be the combination of input which would generate the first output of circuit C1, as 0 and a second output to be 1?

1. ∃ i1, i2, i3 Signal (In(1, C1))=i1 ∧ Signal (In(2, C1))=i2 ∧ Signal (In(3, C1))= i3
2. ∧ Signal (Out(1, C1)) =0 ∧ Signal (Out(2, C1))=1

7. Debug the knowledge base:

Now we will debug the knowledge base, and this is the last step of the complete process. In this step, we will try to debug the issues of knowledge base.

In the knowledge base, we may have omitted assertions like $1 \neq 0$

Inference in First-Order Logic

Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences. Before understanding the FOL inference rule, let's understand some basic terminologies used in FOL.

Substitution:

Substitution is a fundamental operation performed on terms and formulas. It occurs in all

inference systems in first-order logic. The substitution is complex in the presence of quantifiers in FOL. If we write **F[a/x]**, so it refers to substitute a constant "**a**" in place of variable "**x**".

Equality:

First-Order logic does not only use predicate and terms for making atomic sentences but also uses another way, which is equality in FOL. For this, we can use **equality symbols** which specify that the two terms refer to the same object.

Example: Brother (John) = Smith.

As in the above example, the object referred by the **Brother (John)** is similar to the object referred by **Smith**. The equality symbol can also be used with negation to represent that two terms are not the same objects.

Example: ¬(x=y) which is equivalent to x≠y.

FOL inference rules for quantifier:

As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:

- **Universal Generalization**
- **Universal Instantiation**
- **Existential Instantiation**
- **Existential introduction**

Universal Generalization:

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).
- It can be represented as: $\frac{P(c)}{\forall x\, P(x)}$.
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.

- **Example:** Let's represent, P(c): "**A byte contains 8 bits**", so for ∀ **x P(x)** "**All bytes contain 8 bits**.", it will also be true.

Universal Instantiation:

- Universal instantiation is also called as universal elimination or UI is a valid inference rule. It can be applied multiple times to add new sentences.
- The new KB is logically equivalent to the previous KB.
- As per UI, **we can infer any sentence obtained by substituting a ground term for the variable**.
- The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from ∀ **x P(x) for any object in the universe of discourse**.
- It can be represented as: $\frac{\forall x\, P(x)}{P(c)}$.

Example:1.

IF "Every person like ice-cream"=> ∀x P(x) so we can infer that
"John likes ice-cream" => P(c)

Example: 2.

Let's take a famous example, "All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:

∀x king(x) ∧ greedy (x) → Evil (x),

So from this information, we can infer any of the following statements using Universal Instantiation:

- **King(John) ∧ Greedy (John) → Evil (John),**
- **King(Richard) ∧ Greedy (Richard) → Evil (Richard),**
- **King(Father(John)) ∧ Greedy (Father(John)) → Evil (Father(John)),**

Existential Instantiation:

- Existential instantiation is also called as Existential Elimination, which is a valid inference rule in first-order logic.
- It can be applied only once to replace the existential sentence.
- The new KB is not logically equivalent to old KB, but it will be satisfiable if old KB was satisfiable.
- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- The restriction with this rule is that c used in the rule must be a new term for which P(c) is true.
- It can be represented as: $\frac{\exists x\, P(x)}{P(c)}$

Example:

From the given sentence: **∃x Crown(x) ∧ OnHead(x, John),**

So we can infer: **Crown(K) ∧ OnHead(K, John),** as long as K does not appear in the knowledge base.

- The above used K is a constant symbol, which is called **Skolem constant**.
- The Existential instantiation is a special case of **Skolemization process**.

Existential introduction

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic.
- This rule states that if there is some element c in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the

property P.

- It can be represented as: $\dfrac{P(c)}{\exists x P(x)}$

Generalized Modus Ponens Rule:

- For the inference process in FOL, we have a single inference rule which is called Generalized Modus Ponens. It is lifted version of Modus ponens.

- Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."
- According to Modus Ponens, for atomic sentences **pi, pi', q**. Where there is a

$$\frac{p1', p2', \ldots, pn', (p1 \land p2 \land \ldots \land pn \Rightarrow q)}{\text{SUBST}(\theta, q)}$$

substitution θ such that SUBST **(θ, pi',) = SUBST(θ, pi)**, it can be represented as:

Example:

We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.

What is Unification?

- Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process.
- It takes two literals as input and makes them identical using substitution.
- Let Ψ1 and Ψ2 be two atomic sentences and σ be a unifier such that, **Ψ1 = Ψ2** , then it can be expressed as **UNIFY(Ψ1, Ψ2)**.

Example: Find the MGU for Unify{King(x), King(John)}

Let Ψ1 = King(x), Ψ2 = King(John),

Substitution θ = {John/x} is a unifier for these atoms and applying this substitution, and both expressions will be identical.

- The UNIFY algorithm is used for unification, which takes two atomic sentences and returns a unifier for those sentences (If any exist).
- Unification is a key component of all first-order inference algorithms.

- It returns fail if the expressions do not match with each other.
- The substitution variables are called Most General Unifier or MGU.

E.g. Let's say there are two different expressions, **P(x, y), and P(a, f(z))**.

In this example, we need to make both above statements identical to each other. For this, we will perform the substitution.

P(x, y) .... (i)
P(a, f(z))........ (ii)

- Substitute x with a, and y with f(z) in the first expression, and it will be represented as **a/x** and f(z)/y.
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: **[a/x, f(z)/y]**.

Conditions for Unification:

Following are some basic conditions for unification:

- Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.
- Number of Arguments in both expressions must be identical.
- Unification will fail if there are two similar variables present in the same expression.

Forward Chaining and backward chaining in AI

In artificial intelligence, forward and backward chaining is one of the important topics, but before understanding forward and backward chaining lets first understand that from where these two terms came.

Inference engine:

The inference engine is the component of the intelligent system in artificial intelligence, which applies logical rules to the knowledge base to infer new information from known facts. The first inference engine was part of the expert system. Inference engine commonly proceeds in two modes, which are:

a. **Forward chaining**
b. **Backward chaining**

Forward chaining

- Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine.

- Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.

- The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts.

- This process repeats until the problem is solved.

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Consider the following famous example which we will use in both approaches:

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."

Prove that **"Robert is criminal."**

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p)………(1)

- Country A has some missiles. **?p Owns(A, p) ∧ Missile(p)**. It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

  Owns(A,T1......... **(2)**
  Missile(T1) .... **(3)**

- All of the missiles were sold to country A by Robert.

  ?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) .... **(4)**

- Missiles are weapons.

  Missile(p) → Weapons (p)............................**(5)**

- Enemy of America is known as hostile.

  Enemy(p, America) →Hostile(p) ....**(6)**

- Country A is an enemy of America.

  Enemy (A, America) ....**(7)**

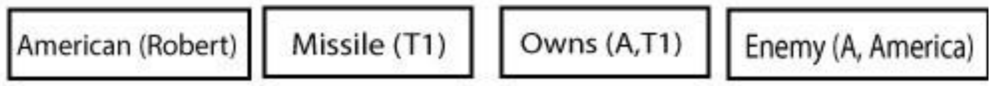
- Robert is American

  American(Robert).............................**(8)**

Forward chaining proof:

Step-1:

In the first step we will start with the known facts and will choose the sentences which do not have implications, such as: **American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1)**. All these facts will be represented as below.



Step-2:

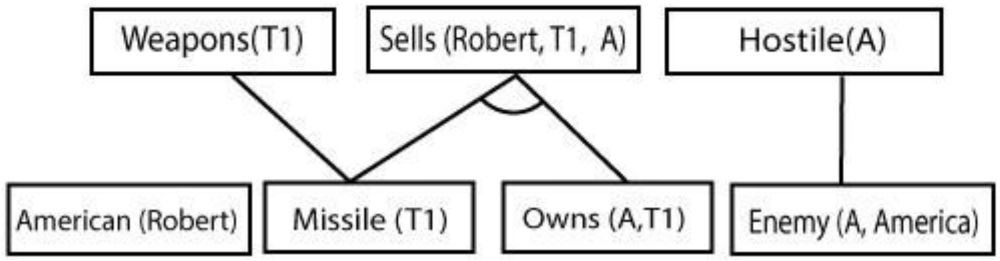
At the second step, we will see those facts which infer from available facts and with satisfied premises.

Rule-(1) does not satisfy premises, so it will not be added in the first iteration.

Rule-(2) and (3) are already added.

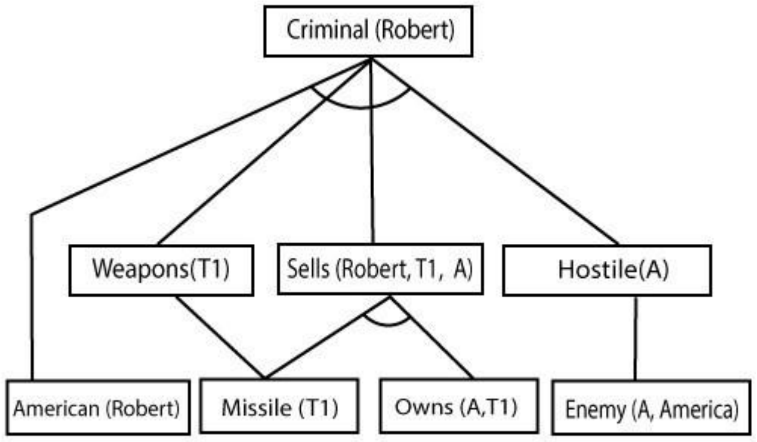
Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).

Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).



Step-3:

At step-3, as we can check Rule-(1) is satisfied with the substitution **{p/Robert, q/T1, r/A}, so we can add Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



Hence it is proved that Robert is Criminal using forward chaining approach.

Backward Chaining:

Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine. A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Example:

In backward-chaining, we will use the same above example, and will rewrite all the rules.

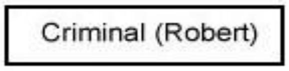
- **American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1) Owns(A, T1) (2)**
- **Missile(T1)**
- **?p Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A) ... (4)**
- **Missile(p) → Weapons (p) (5)**
- **Enemy(p, America) →Hostile(p) (6)**
- **Enemy (A, America) (7)**
- **American(Robert). ..(8)**

Backward-Chaining proof:

In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

Step-1:

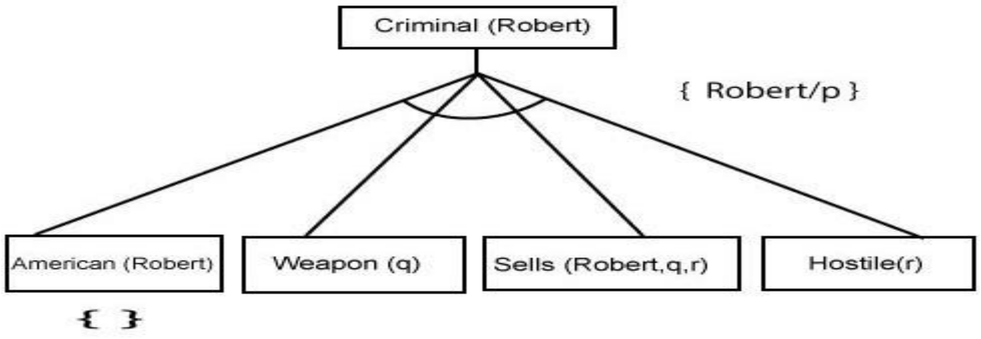
At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true. So our goal fact is "Robert is Criminal," so following is the predicate of it.



Step-2:

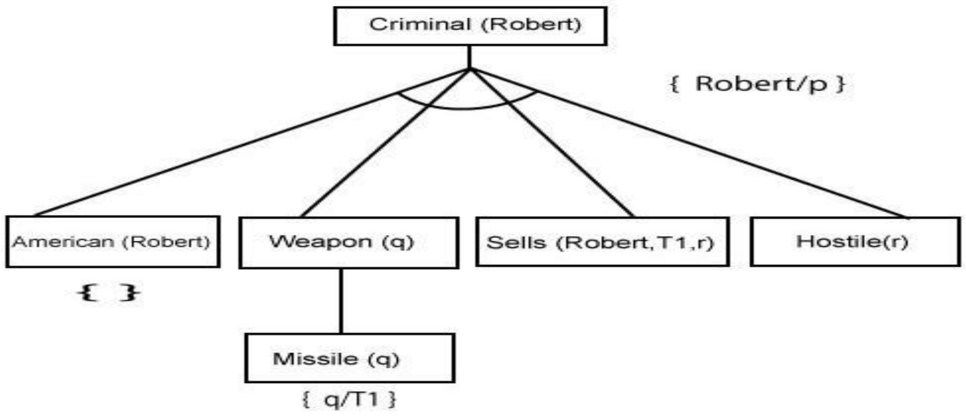
At the second step, we will infer other facts form goal fact which satisfies the rules. So as we can see in Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.

Here we can see American (Robert) is a fact, so it is proved here.

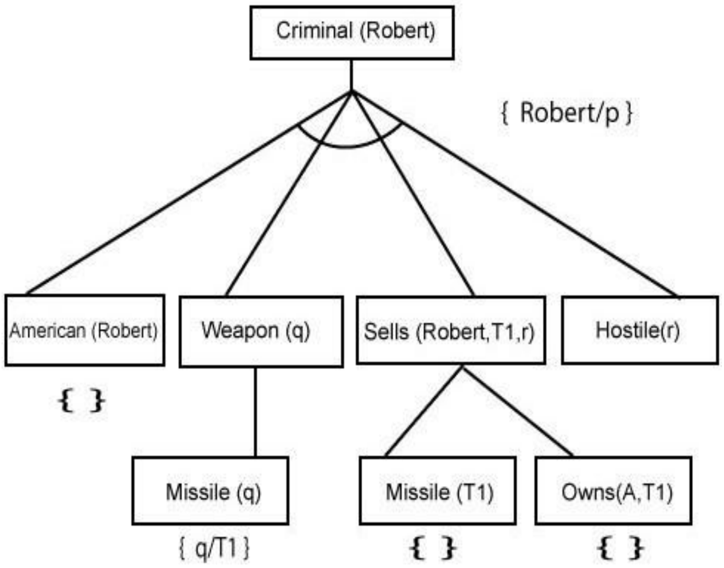


Step-3:t At step-3, we will extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.

Step-4:

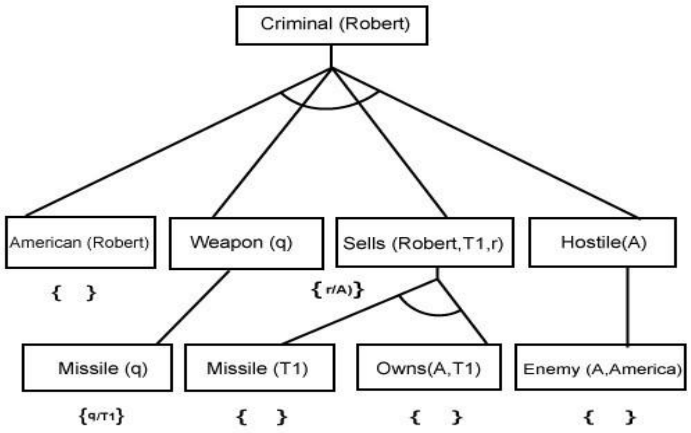


At step-4, we can infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the **Rule- 4**, with the substitution of A in place of r. So these two statements are proved here.



Step-5:

At step-5, we can infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule-6. And hence all the statements are proved true using backward chaining


Criminal (Robert)
American (Robert)
Weapon (q)
Sells (Robert,T1,r)
Hostile(A)
{ }
{r/A)}
Missile (q)
Missile (T1)
Owns(A,T1)
Enemy (A,America)
{q/T1}
{ }
{ }
{ }