UNIT III

3. KNOWLEDGE REPRESENTATION

First Order Logic, Representation Revisited, Syntax and Semantics for First Order Logic - Using First Order Logic, Knowledge Engineering in First Order Logic, Inference in First Order Logic, Prepositional Versus First Order Logic, Lifting and Unification, Forward Chaining and Backward Chaining, Resolution, Knowledge Representation, Ontological Engineering, Categories and Objects, Actions, Simulation and Events, Mental Events and Mental Objects

3.1. First Order Logic(FOL)

3.2. Representation Revisited

FOL or **First Order Predicate Calculus** (FOPC), which creates a powerful set of ontological commitments i.e. properties, objects, functions and relations of the world belongings to be presented.

- **Properties** : It is used to differentiate one object with another object. (E.g. big, small, round etc.)
- **Objects** : Items with separate identities. (E.g. home, people, college, colors etc.)
- **Functions** : One type of relation, which has just a single value from a specified entry. (E.g. mother of, greatest brother etc.)
- **Relations** : relation exists between objects. (E.g. owns, bigger than etc.)

The initial variation between FOL logic and propositional lies in the existential obligation prepared by all languages – which are, what it suppose regarding the scenery of actuality.

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Figure: Formal Languages with their Epistemological and Ontological Commitments

3.3. Syntax and Semantics for FOL

FOL has sentences, and also has terms, which signify objects. Variables, functions and Constant symbols are utilized to create terms, predicate symbols and quantifiers are utilized to create sentences.

Models for FOL

The domain of the representation is a group of objects it have; these objects are occasionally known as domain elements. Figure: displays a pattern with 5 objects.

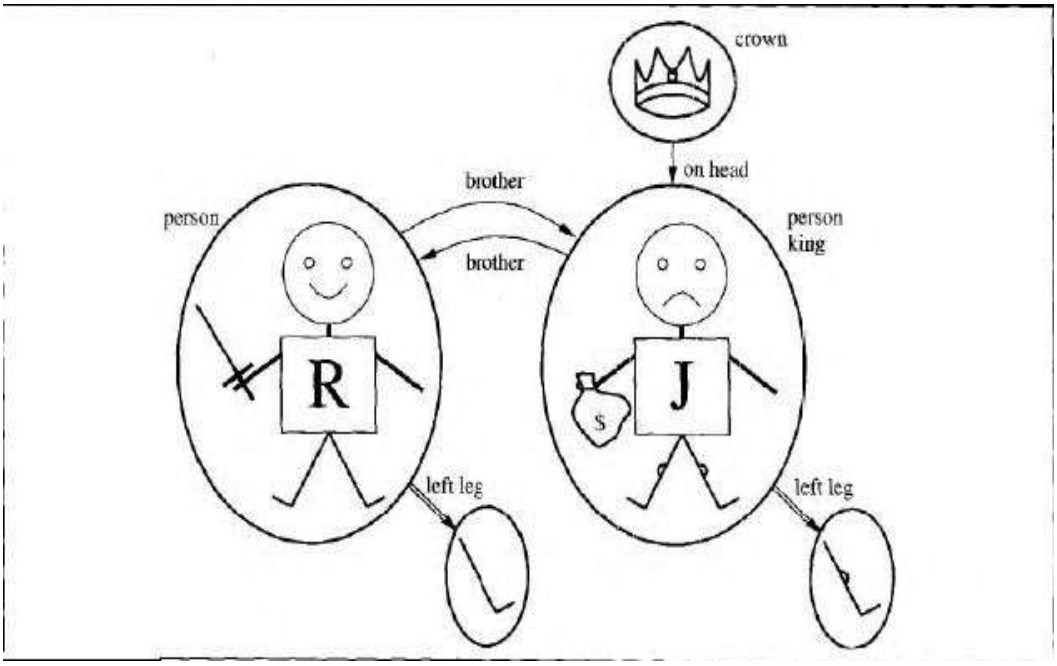


Figure: A Model Having 5 Objects Three Unary Relations, Two Binary Relations and One Unary Function

Symbols and Interpretations

Figure shows the formal grammar of FOL**.**

$$
\begin{aligned}
Sentence &\rightarrow AtomicSentence \\
&\mid (\ Sentence\ Connective\ Sentence\ ) \\
&\mid Quantifier\ Variable,\ldots\ Sentence \\
&\mid \neg\ Sentence
\end{aligned}
$$

$$
AtomicSentence \rightarrow Predicate(Term,\ldots) \mid Term = Term
$$

$$
\begin{aligned}
Term &\rightarrow Function(Term,\ldots) \\
&\mid Constant \\
&\mid Variable
\end{aligned}
$$

$$
\begin{aligned}
Connective &\rightarrow \Rightarrow \mid \wedge \mid \vee \mid \Leftrightarrow \\
Quantifier &\rightarrow \forall \mid \exists \\
Constant &\rightarrow A \mid X_1 \mid John \mid \ldots \\
Variable &\rightarrow a \mid x \mid s \mid \ldots \\
Predicate &\rightarrow Before \mid HasColor \mid Raining \mid \ldots \\
Function &\rightarrow Mother \mid LeftLeg \mid \ldots
\end{aligned}
$$

Figure: The Syntax of FOL with Equality, Specified in BNF

The basic semantic components of the FOL are the signs that apply for functions, relations and objects. The signs arrive in 3 types: **function symbols** apply for functions; **constant symbols** apply for objects; and **predicate symbols** apply for relations.

Terms are a logical statement, which indicates to an object.

An **atomic statement** is created from a predicate symbol pursued by a parenthesized listing of expressions.

The **atomic statement** is correct in a specified model, beneath a specified clarification, if the relationship indicated by the predicate symbol keeps amongst the objects indicated by the arguments.

Quantifiers: Theses are used to state properties of total collection of objects, than presenting the objects by the name.

First order logic (FOL) contains 2 quantifiers:

1. Existential quantifiers
2. Universal quantifiers

Equality

FOL consists of one additional way to build atomic sentences, excluding a terms and predicate as described in advance. We may apply the **equality sign** to create sentences to work that 2 expressions indicate to the similar object.

3.4. Knowledge Engineering in FOL

Knowledge Engineering (KE): The common procedure of knowledge base planning procedure is called knowledge engineering (KE).

The Knowledge Engineering Method

KE projects differ usually in difficulty, scope and content, but suchlike projects contain the bellow steps:

a. Discover the task.
b. Collect the appropriate knowledge.
c. Choose on expressions of constants, functions and predicates.
d. Encode broad awareness regarding the domain.
e. Encode an explanation of the detailed problem instance.
f. Pose questions to the deduction process and obtain response.
g. Debug the intelligence base.

Electronic Circuits' Domain

We can establish a knowledge base and ontology that permit us to cause regarding digital circuits of the type displayed in the bellow Figure. We follow the 7 step procedure for information engineering.

a. Discover the job.
b. Collect the appropriate knowledge.
c. Choose on expressions of constants, predicates and functions.
d. Encode broad awareness regarding the domain.
e. Encode an explanation of the detailed problem instance.

f. Pose questions to the deduction process and obtain response.

g. Debug the intelligence base.

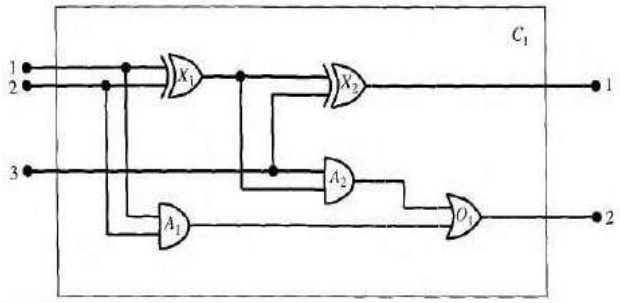


Figure: A digital-circuit 'C1', purporting to be a 1-bit. Full adder. The primary 2 inputs are the 2 bits to be inserted and the 3rd input is a carry bit. The 1st output is the addition and the 2nd output is a carry bit for the subsequent adder. The circuit has 2 XOR gates, 2 AND gates and 1 OR gate.

3.5. Inference in FOL

FOL vs. Propositional

Inference Rules for Quantifiers

The rule of **Universal Instantiation**(UI) declares that, we may assume every statement gained by replacing a **ground-term** for the variable.

SUBST(*@a*, *)* indicate the outcome of using the replace *eight* to the statement '*a'*. Afterward the rule is drafted

$$\forall v\, a$$

$$\text{SUBST}(\{ V/\sim 4 \},$$

for every ground term '*g'* and variable 'v'.

The subsequent **Existential-Instantiation** rule: the existential quantifier is somewhat most complex. For every variable-v, constant symbol-*k* and sentence *'a',* which doesn't arrive somewhere in the intelligence base.

3.6. Lifting and Unification

A First Order Inference Rule

This inference procedure may be confined as a distinct inference rule, which we term General.

GENERALIZEPONEDN S **sized Modus-Ponens:** The atomic statement *q* and *p,,,, p,',*

Wherever there is a replacement *eight* such *Suesr(Q,pil)=Suesr(B,p,),* for all *i,*

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

Unification

Lifted-inference rules need discovering replacements, which create dissimilar reasonable statements.

UNIFICATION looks the same. This procedure is known as **unification** and it is a crucial element of every first-order UNIFIER inference algorithms.

The ***UNIFY*** **algorithm** obtain 2 statements and give back a **unifier** for them if one be present:

UNIFY(^,q)=**0** wherever *SUBST(O,= ~)SUBST(@ q,).*

The problem happens because the 2 sentences occur to utilize the similar name of a variable. The problem may be keep away by **regularizing separately** one of the 2 statements is combined that means changing the name of its APART variables to keep away name conflicts.

function UNIFY(x, y, θ) **returns** a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound
    y, a variable, constant, list, or compound
    θ, the substitution built up so far (optional, defaults to empty)

  if 0 = failure **then return** failure
  else if x = y **then return** θ
  else if VARIABLE?(x) **then return** UNIFY-VAR($x, y, 0$)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, 0)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
  else return failure

---

function UNIFY-VAR($var, x, 0$) **returns** a substitution
  inputs: var, a variable
    x, any expression
    0, the substitution built up so far

  if $\{var/val\} \in 6$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** failure
  else return add $\{varlx)$ to 6

Figure: The UNIFY Algorithm

Save and Retrieval

Primarily the SAY and INQUIRE functions are used to tell and investigate a information base are the additional primary SAVE and RETRIEVAL functions. SAVE 'S' saves a statement '*s*' into the information base and RETRIEVAL (^) gives back every unifiers such the question 'q' unifies with a few statement in the information base.

3.7. Forward Chaining

Forward chaining algorithm for hypothesis definitive sections was already specified. The proposal is easy: initiate with the atomic statements in the information base and perform ModusPonens in the frontward direction, inserting latest atomic statements, upto no more inferences may be created.

First Order Definite Sections

It directly resembles propositional definitive sections these are disunion of accurate of which *precisely 1 is positive. An* exact sections whichever is atomic or it's an inference whose predecessor is a combination of optimistic literals and who's consequential is an individual optimistic literal.

This data base contain no action symbols and therefore an example of the class "DATALOG" of **Data-log** databases, which is, the groups of first-order definitive sections without function symbols.

An Simplest Forward Chaining Algorithm

Initial Forward-Chaining algorithm can believe a simplest one, as displayed in the bellow Figure.

```
function FOL-FC-ASK(KB, α) returns a substitution or false
  inputs: KB, the knowledge base, a set of first-order definite clauses
          α, the query, an atomic sentence
  local variables: new, the new sentences inferred on each iteration

  repeat until new is empty
      new ← { }
      for each sentence r in KB do
          (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
          for each θ such that SUBST(θ, p1 ∧ ... ∧ pn) = SUBST(θ, p'1 ∧ ... ∧ p'n)
                      for some p'1, ..., p'n in KB
              q' ← SUBST(θ, q)
              if q' is not a renaming of some sentence already in KB or new then do
                  add q' to new
                  φ ← UNIFY(q', α)
                  if φ is not fail then return φ
      add new to KB
  return false
```

Figure: Efficient Forward Chaining

There are 3 available basis of difficulty. Initially, the inner loop of the algorithm includes discovering every available unifier such the basis of a unifies rule with an appropriate group of particulars in the data base. This is generally known as **pattern matching** and will be high costly. Second, this algorithm again checks each rule on each round to observe regardless if its locations are fulfilled, yet if little add-ons are completed to the data base on every round. Lastly, the algorithm could produce several details that are unnecessary to the target.

Matching Rules against Unknown Facts

We will convey each limited domain CSP as an individually best section jointly with a few related ground particulars.

Incremental Forward Chaining

Redundant rule matching will be keep away if we create the following studies: ***Each latest fact deduced on repetition 't' should be resulting from no less than one latest fact deduced on repetition 't–1'.***

3.8. Backward Chaining

Backward-Chaining Algorithm

The bellow figure explains an easy backward-chaining algorithm:

function FOL-BC-ASK(KB, *goals*, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    goals, a list of conjuncts forming a query (θ already applied)
    θ, the current substitution, initially the empty substitution $\{\}$
  local variables: *answers*, a set of substitutions, initially empty

  if *goals* is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST(*goals*))
  for each sentence r **in** KB where STANDARDIZE-APART(r) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
    new-goals $\leftarrow [p_1, \ldots, p_n | \text{REST}(goals)]$
    answers $\leftarrow$ FOL-BC-ASK(KB, *new-goals*, COMPOSE(θ', θ)) $\cup$ *answers*
  return *answers*

Figure A simple backward-chaining algorithm.

Logic Programming

$$Algorithm = Logic + Control$$

Prolog is by distant the almost extensively utilized logic programming language. Its clients count in the hundreds of thousands. It is utilized initially as a fast prototyping language and for sign modification work such as scripting compilers(1990, VanRoy) and interpret to natural language. Prolog-programs are groups of definitive sections drafted in a document somewhat dissimilar from normal first-order. Logic Program utilizes small letters for constants and capital letters for variables. Sections are drafted with the head earlier the body:- it is utilized for a period marks the ending of a statement, comma divide literals in the body and left implication:

The implementation of Prolog-programs is complete via depth first backward-chaining, where sections are attempted in the order, and then they are drafted in the knowledge base. Few portions of Prolog drop outside normal logical inference:

Proficient Execution of Logic Programs

The implementation of a Prolog-program may occur in 2 forms interpret and compiled. Interpretation fundamentally quantities to operating the FOL_ BC_ ASK algorithm from the bellow figure.

```
procedure APPEND(ax, y, ar, continuation)

    trail ← GLOBAL-TRAIL-POINTER()
    if ax = [] and UNIFY(y, az) then CALL(continuation)
    RESET-TRAIL(trail)
    a ← NEW-VARIABLE(); x ← NEW-VARIABLE(); z ← NEW-VARIABLE()
    if UNIFY(ax, [a | x]) and UNIFY(az, [a | z]) then APPEND(x, y, z, continuation)
```

Figure: APPEND Algorithm

3.9. Resolution

Conjunctive Normal Form (CNF) for First-order Logic

In the propose case, first-order decision needs that statements be in **CNF**, which is, a conjunction of sections, wherever every section is a disconnection of literal. Literals may have variables that are supposed to be globally evaluated.

e.g., the statement

V x American(x) A Weapon(y) A Sells(z, y , z) A Hostile(z) +- Criminal (x)

becomes, in CNF,

l A m e r i c a n (x) V -1 Weapon(y) V l S e l l s (x , y , z) V 1 Hostile(z) V Criminal (x) .

Each statement of first-order logic may be transformed into a priory similar CNF statement.

We can demonstrate the process by interpreting the sentence "Every person who feels affection for all animals is loved by everyone,"

Or

'dx ['d y Animal(y) J Loves(x, y)] + [3 y Loves(y, x)].

The steps are as follows:

1. Eliminate implications:

ti x [i ' d y 1 Animal (y) V Loves(x, y j] v [3 y Loves(y, xj]

2. Shift *for* inmost: Adding to the common rules for invalid combinative, we require rules for invalid quantifiers. Therefore, we contain *l'dxp* turn into *3x1p*73*xp* turn into *'dx1p*.

Our sentence moves between the following conversions:

'd x [3 y l(lAnima1 (Y) V Loves(x, y))] V [3 y Loves(y, x)] .

'dx[3 y iiAnimal(y) A lLoves(x, y)] V [3 y Loves(y, x)] .

t i x [3 y Animal(y) A lLoves(x, y)] v [3 y Loves(y, x)] .

Normalize Variables

Distribute V over A

Drop global quantifiers

Solemnize

Completeness of Resolution

Resolution is **refusal-complete** that means, which is a group of statements is not classifiable, and then resolution can forever be capable to obtain a challenge. Resolution may not be utilized to produce all rational outcomes of a group of statements, but it may be utilized to create that a specified statement is involved by the group of statements.

Our target is to demonstrate the following: if 'S' ***is an unsatisfied group of sections next the request of limited count of resolution steps to 'S' can give up a negation.***

The fundamental structure of the evidence is exposed in the bellow Figure.

It continues as follows:

1. Initially, we examine that if 'S' is unclassified, next there continues an appropriate group of ***ground_instances*** of the sections of 'S' thus, this group is unclassified (Her brand's principle).
2. We request to the **ground resolution theorem** that situations that proposal decision is finished for ground statements.
3. We utilize a **lifting lemma** to demonstrate that, for every proposal decision evidence utilizing the group of ground statements, there is matching first-order decision evidence utilizing the first-order statements from which the ground statements are gained.

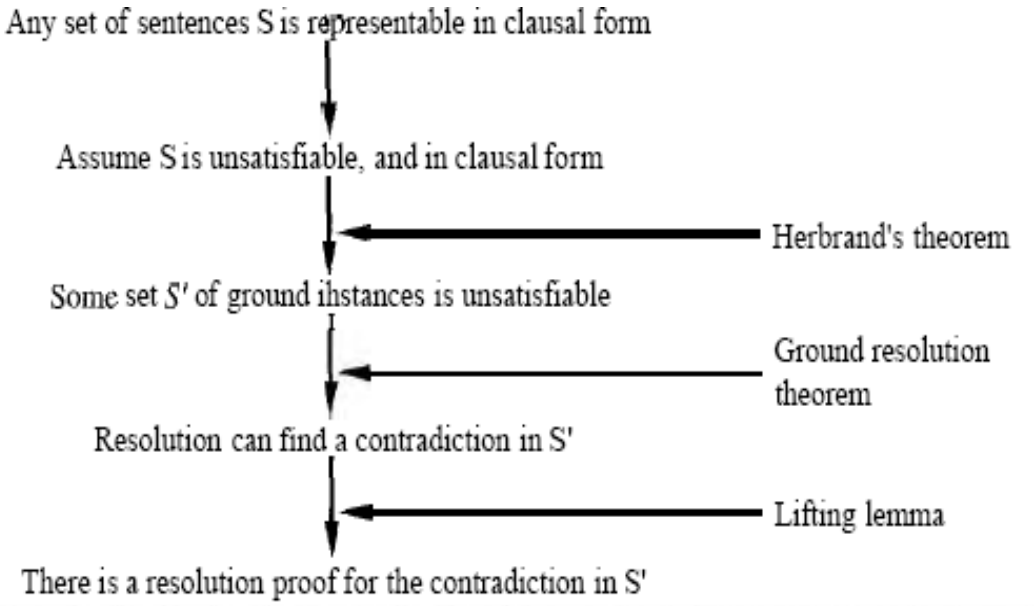


Figure: Structure of a Completeness Proof for Resolution

```
procedure OTTER(sos, usable)
  inputs: sos, a set of support—clauses defining the problem (a global variable)
          usable, background knowledge potentially relevant to the problem

  repeat
      clause ← the lightest member of sos
      move clause from sos to usable
      PROCESS(INFER(clause, usable), sos)
  until sos = [] or a refutation has been found
```

```
function INFER(clause, usable) returns clauses

  resolve clause with each member of usable
  return the resulting clauses after applying FILTER
```

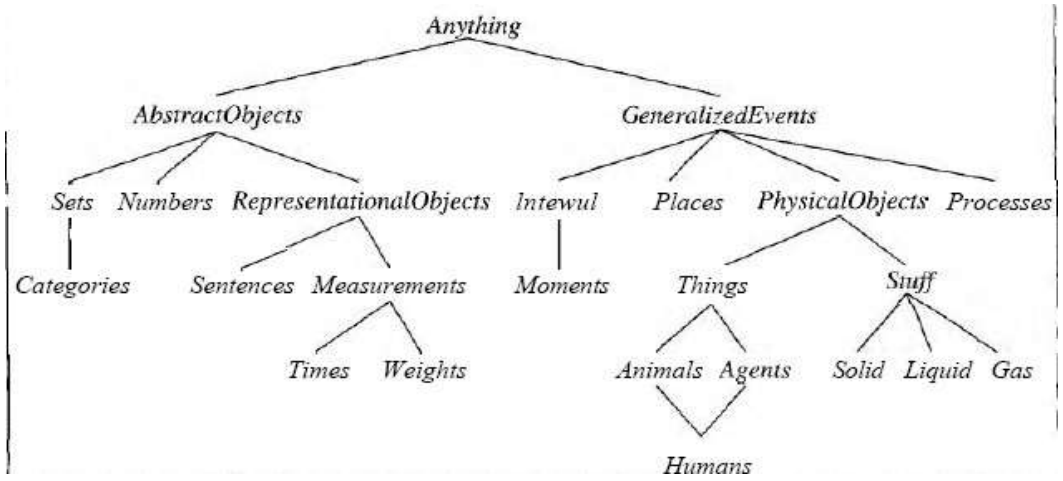
```
procedure PROCESS(clauses, sos)

  for each clause in clauses do
      clause ← SIMPLIFY(clause)
      merge identical literals
      discard clause if it is a tautology
      sos ← [clause | sos]
      if clause has no literals then a refutation has been found
      if clause has one literal then look for unit refutation
```

Figure : Drawing of the OTTER theorem prover. Heuristic-control is executed in the choice of the "lightest" section and in the FILTER function that deletes not interesting sections from deliberation.

3.10. Knowledge Representation

3.11. Ontological Engineering

This chapter demonstrates how to make these presentations, focusing on common ideas, such as Time, *Actions,* Beliefs-that, *and Physical Objects* occur in most dissimilar domains. Presenting these abstract ideas is occasionally known as ontological-engineering. It is linked to the data engineering procedure, but works on a grander-scale. The expectation of presenting *all* in the universe is frightening.



For example, we can describe the information of dissimilar kinds of books, televisions, objects robots, or what should be filled afterward.

The common structure of ideas is known as **upper ontology,** for the reason that of the gathering of sketch charts with the common ideas at the top and the most precise ideas.

3.12. Objects and Categories

The management of objects into **categories** is a very important element of data presentation. Even if communications with the universe take position at the stage of single objects, *a lot analysis gets position at the stage of categories.*

There are 2 chances for presenting categories in first order logic: **objects** and **predicates.**

Measurements

The both commonsense and scientific theories of the universe, cost, mass, objects contain height, and so on. The values, which we allocate for the properties are known as **measures.** Normal quantitative-measures are really simple to present. We assume that the world contains abstract objects measure as the *length,* which is the length of this line-segment.

3.13. Events, Actions, and Situations

The Idea of Situation Calculation

One noticeable way to keep away from several duplicates of axioms is easily to count more time to declare, *"'Vt',* is the outcome at *t*+1 of performing the operation at '*t*'". Rather than association with accurate times similar to *t*+1, we can focus on this clause on ***situations*** that represent the resultant states from performing operations. This method is known as **situation calculation** and associates with the idea:

Activities are logical expressions such as ***frontward*** and ***TurnRight***. *C*urrently, we can imagine that the situation involves just a single agent. (If there is greater than one, an extra argument may be added to convey. Which, agent is performing the activity).

Situations are logical expressions containing the primary state (regularly known as *So)* and each situation that are created by using an activity to a state. The function *Result(a,s)* (occasionally known as *Do)* names the state that outcomes whenever activity *a* is applied in state *s.* The bellow Figure demonstrates this thought.

Fluent are tasks and declares that differ from one state to the other, the position of the agent or the attention of the wumpus. The dictionaries convey a fluent is a thing that flow similar to a fluid. It means flowing or shifting crossways states. By showing, the state is at all times the previous argument of a fluent. example, *l Holdzng (G 1,So)* coveys that the agent is not containing the gold-GI in the primary state '*So'. Age (Wumpus,So)* indicates to the wumpus's age in '*So'.*

A temporary or **continual** builds and performs are also permitted. Examples contain the predicate_Gold(GI) and the function *Left_Leg_Of* (Wumpus *).*

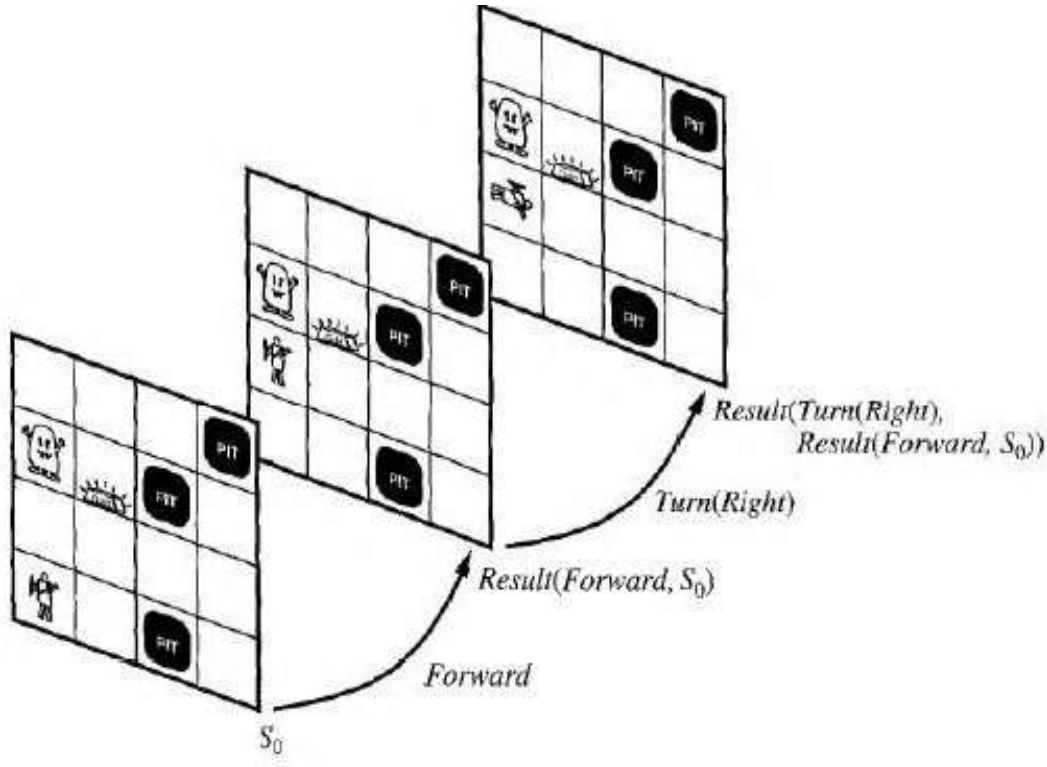


Figure: In Situation Calculation, Every Situation (excluding 'So') is the Outcome of an Action

A state calculation agent would be capable to deduct the result of a specified series of PROJECTION activity; this is the **projection** work, with a appropriate constructive_inference algorithm, it would also be capable to connect a series that attains a preferred outcome; it is the **planning** job.

Defining Actions in Situation Calculations

In the easiest edition of situation calculation, every action is defined by 2 axioms: a **possibility axiom,** which conveys when it is probable to perform the operation, and an **effect axiom,** which conveys EFFECT-AXIOM what occurs whenever an available action is performed.

The axioms contain the bellow method:

EFFECT_AXIOM: *Poss (a,s)+Modifications that outcome from doing action.*

POSSIBILITY_AXIOM: *Preconditions+Poss (a,s)*.

The problem is the effect-axioms conveys what modifications, but does not convey what continues the same.

Presenting all the things, which continue the similar, is known as the **frame problem.** We can discover an adequate result to the frame-problem for the reason that, the actual universe, approximately all continues the similar approximately everything at the time. Every operation changes just a small part of every fluent.

A method is to draft accurate **frame_axioms,** which conveys what continues the similar.

Resolving the Presentational Frame Problem

The result to the presentational frame problem contains only a small modification in point of view on how to draft the axioms. As an alternative of drafting out the outcome of every operation, we look at how every fluent predicate derives above the time. The axioms we utilize are known as **successor state axioms.**

It contains the bellow method:

AXIOM SUCCESSOR-STATE AXIOM:

Operation is available+(Fluent is true in solution state#Action 'S'
reaction completed it true. It was true previous to and operation is left).

The *Identical names-axiom* states are not qualifies for each pair of not changeable values in the knowledge base.

Resolving the Inferential-frame Problem

To resolve the inferential-frame problem, we contain 2 chances. Initially, we w*ould* reject situation calculation and discover a latest formality for drafting axioms. That has been completed with formalities such as *'1'* has *fluent-calculus.* Second, we would change the inference process to control frame axioms rnose precisely.

Time and Event Calculus

The starting and ending associations play a responsibility same to the outcome association in state calculation; Initiates(e,f,t) means, which the happening of event 'e' at time 't' source fluent 'f' to turn into true, while Terminates(e,f,t_1) means, which 'f' halts to be true. We utilize Happens(e,t) to denote that event 'e' occurs at time 't', and we utilize Clipped(f,t,t_2) to denote that 'f' is ended by few events for a time among 't_1' and 't_2'. Properly, the axiom is:

EVENT CALCULUS AXIOM:

$$T(f,t_2) \Leftrightarrow \exists e,t \;\; Happens(e,t) \wedge Initiates(e,f,t) \wedge (t < t_2) \wedge \neg Clipped(f,t,t_2)$$

$$Clipped(f,t,t_2) \Leftrightarrow \exists e,t_1 \;\; Happens(e,t_1) \wedge Terminates(e,f,t_1) \wedge (t < t_1) \wedge (t_1 < t_2) \; .$$

Generalized Events

A generalized event is collected from features of a few "space time-chunk''. It is a portion of this multi-dimensional space time world. This removal simplifies the majority of the ideas we have observed so far, locations, fluent, time, physical objects, and including actions.

3.14. Mental Events and Mental Objects

A traditional theory of principles, we start with the connection among mental objects and agents, associations such as Wants, Knows, and Believes. Associations of this type are known as propositional attitudes, be-ATTITUDE because they explain an approach, which an agent may obtain near a theorem. Rotating a theorem into an object is called reification.

Professionally, the property of being capable to replacement a expression openly for an equivalent expression is known as **Referential Transparency**.

There are 2 approaches to attain this.

The **primary** is to utilize a dissimilar type of logic known as **Modal Logic**, in which propositional approaches such as Knows and Believes turn into Modal Operators, which is attentively unclear. This method is enclosed in the old comments part.

The **secondary** method that we can follow is to attain efficient dullness inside a Referentially-Transparent language by utilizing a syntactic-theory of mental-objects. This denotes that mental-objects are presented by a group of characters.

Knowledge and Belief

The relations among knowing and believing have been prepared broadly in beliefs. It is generally conveyed that intelligence is explained true belief. Action, time, and knowledge.

Actions may contain knowledge effects and knowledge preconditions.

Workout to collect and utilize knowledge is often presented by using a shorthand document called **runtime variables**.

Question Bank

Unit - III

Part - A

1. Write short notes on knowledge based agent.
2. Write the two functions of KB agent.
3. Define inference.
4. Explain three levels of knowledge based agent with an example.
5. Define logic.
6. Define entailment.
7. Define truth preserving in logic.
8. Give example for syntax and semantic representation.
9. Define inference procedure.
10. Define logical inference or deduction.
11. Define validity with one example.
12. Describe satisfiablity with one example.
13. List the names of five different types of logic.
14. Differentiate propositional logic with FOL.
15. Write the BNF grammar representation for propositional logic.
16. List the names of inference rules of propositional logic.
17. Write the BNF grammar representation for FOL.
18. Define predicate symbol with an example.
19. Define WFF with an example.
20. Differentiate function and relation in FOL with an example.
21. Define ground term.

22. Define horn clause.
23. Write short notes on higher order logic.
24. Write short notes on uniqueness quantifier.
25. Write short notes on uniqueness operator.
26. Define domain. Give example.
27. Define axiom.
28. Define independent axiom.
29. List the names of logical agents for wumpus world problem.
30. List the names of inference rules with quantifiers.
31. State the advantage of generalized modus ponen rule.
32. Write short notes on canonical form.
33. Write short notes on unification.
34. Differentiate forward and backward chaining.
35. Define refutation.
36. Define skolimization with an example.
37. Differentiate CNF and implicative normal form.
38. Write short notes on resolution strategies.
39. Covert the given sentence into propositional form.
40. Convert the given sentence into FOL form.
41. Convert the given sentence into CNF & INF form.
42. Define production system. Give example.
43. Define binding list with an example.
44. Write short notes on subsumption method.
45. How parallelization can be achieved in logic programming?
46. Define ontological engineering with an example.
47. Write short notes on situation calculus.
48. Differentiate suff nouns with count nouns with an example.
49. Define frame problem.
50. List the types of frame problem.
51. List the predicates of time intervals.
52. Define verification.

Part - B

1. Define the problem of wumpus world environment and derive the steps to reach a goal Test with corresponding structure representation.

2. Solve the wumpus world problem using propositional logic.
3. Discuss the logical agents of wumpus world problem.
4. Solve the given KB problem using FOL representation.
5. Explain the inference rules of FOL with an example for each.
6. A). Write the rules to convert a FOL sentence into Normal form
   B). Solve the given KB problem using resolution with refutation technique in CNF & INF form.
7. Solve the given KB problem using resolution with refutation technique in INF & CNF.
8. Explain knowledge engineering in FOL with an example.
9. Explain Forward chaining algorithm with one example.
10. Explain Backward algorithm with one example.
11. Explain with suitable example how the real world happening are represented in FOL.
12. Describe unification algorithm with one example.