

Unit – II

Logic Concepts and Logic Programming
Knowledge Representation

Unit – III (Part 1/2)

Expert System, Shells and Tools
Development of Rule-Base and Answering Queries

Unit - II

Propositional Logic

A sentence or well formed formula wff α is called a **tautology** if and only if the value of α is true (**valid**) for all its interpretations.

A sentence or well formed formula wff α is said to be **satisfiable** if there exist at least one interpretation for which α is true.

A sentence or well formed formula wff α is said to be **unsatisfiable** if the value of α is false under all interpretations.

The relationship of **entailment** between sentences is crucial to our understanding of reasoning. A sentence α entails another sentence β if β is true in all worlds where α is true. Equivalent definitions include the **validity** of the sentence $\alpha \rightarrow \beta$ and the **unsatisfiability** of the sentence $\alpha \wedge \neg\beta$.

Inference is the process of deriving new sentences from old ones. **Sound** inference algorithms derive only sentences that are entailed: **complete** algorithms derive all sentences that are entailed.

Propositional logic is a very simple language consisting of **proposition symbols** and **logical connectives**. It can handle propositions that are known true, known false, or completely unknown.

The set of possible models, given a fixed propositional vocabulary, is finite, so entailment can be checked by enumerating models. Efficient model-checking inference algorithms for propositional logic include back tracking and local searching methods and can often solve large problems very quickly.

Inference rules are patterns of sound inference that can be used to find proofs. The **resolution** rule yields a complete inference algorithm for knowledge bases that are expressed in **conjunctive normal form**. **Forward chaining** and **backward chaining** are very natural reasoning algorithms for knowledge bases in **Horn form**.

When use of Truth Table approach result in wastage of time exploring all possibilities, we need some other methods to prove validity of the formula directly. Some such methods are:

1. Natural Deduction System
2. Axiomatic System
3. Semantic Tableau Method
4. Resolution Refutation Method

Propositional logic is reasonably effective for certain tasks within an agent, but does not scale to environments of unbounded size because it **lacks the expressive power to deal** concisely with time, space, and universal patterns of **relationships among objects**.

The Propositional Calculus

1. Atoms: The two distinguished atoms T and F are countably infinite set of characters that begin with a capital letter. Example: P, Q, R, . . . , P1, P2, ON_A_B ...

2. Connectives:

\vee	OR
\wedge	AND
\rightarrow	IMPLIES
\neg	NOT

3. Sentences also known as well-formed formulas (wffs).

Syntax of wffs:

1. Any atom is a wff. Example: P, R, P3
2. If w_1 and w_2 are wffs, so are:
 - $w_1 \vee w_2$ (called a disjunction of w_1 and w_2)
 - $w_1 \wedge w_2$ (called a conjunction of w_1 and w_2)
 - $w_1 \rightarrow w_2$ (called an implication)
 - $\neg w_1$ (called a negation of w_1)

Literals: atoms and atoms with a sign in front.

$w_1 \rightarrow w_2$ w_1 is antecedent and w_2 is consequent

Examples of wffs:

1. $(P \wedge Q) \rightarrow \neg P$
2. $P \rightarrow \neg P$
3. $P \vee P \rightarrow P$
4. $(P \rightarrow Q) \rightarrow (\neg Q \rightarrow \neg P)$
5. $\neg \neg P$

4. Rules of Inference: Number of ways by which additional wffs can be produced.

Some common inference rules:

1. The wff w_2 can be inferred from the wffs w_1 and $w_1 \rightarrow w_2$ (modus ponens)
2. The wff $w_1 \wedge w_2$ can be inferred from the two wffs w_1 and w_2 (\wedge introduction)
3. The wff $w_2 \wedge w_1$ can be inferred from $w_1 \wedge w_2$ (commutativity of \wedge)
4. The wff w_1 can be inferred from $w_1 \wedge w_2$ (\wedge elimination)
5. The wff $w_2 \vee w_1$ can be inferred from the single wff w_2 (\vee introduction)
6. The wff w_1 can be inferred from $\neg(\neg w_1)$ (\neg elimination)

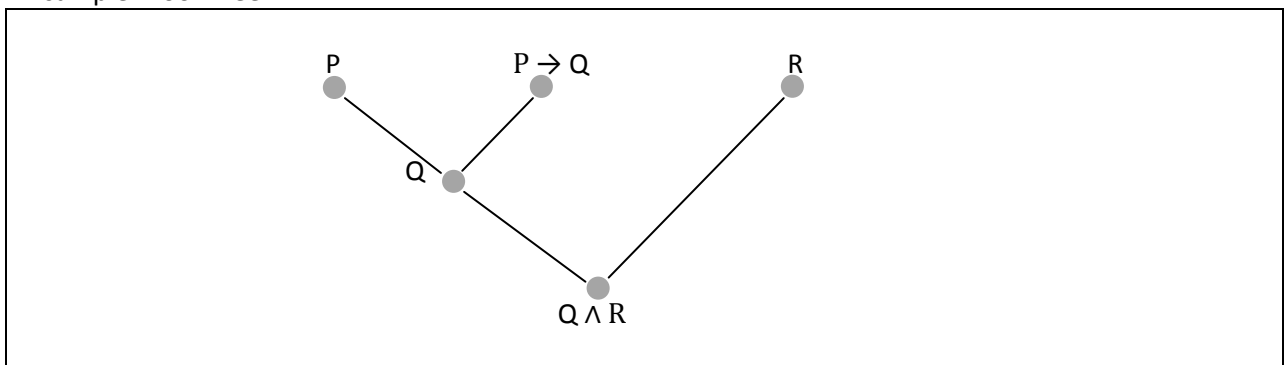
5. Proof (also called Deduction)

The sequence of wffs $\{w_1, w_2, w_3, \dots, w_n\}$ is called a proof of w_n from a set of wffs δ iff each w_i in the sequence is either in δ or can be inferred from a wff (or wffs) earlier in the sequence by using one of the rules of inference.

If there is a proof of w_n from δ , we say that w_n is a theorem of the set δ .

Example: Given a set, δ of wffs: $\{P, R, P \rightarrow Q\}$

A sample Proof Tree:



6. Interpretation: An association of atoms with propositions. Atoms have values – True or False.

w_1	w_2	$w_1 \wedge w_2$	$w_1 \vee w_2$	$\neg(\neg w_1)$	$w_1 \rightarrow w_2$
True	True	True	True	False	True
True	False	False	True	False	False
False	True	False	True	True	True
False	False	False	False	True	True

The Propositional Truth Table.

7. Resolution in Propositional Calculus

Several rules of inference can be combined into one rule called **Resolution**.

Literal: either an atom (positive literal) or negation of atom (negative literal)

Clause: is a set of literals. (s special kind of wff)

Example: $\{P, Q, \neg R\}$ (equivalent to $P \vee Q \vee \neg R$) is a wff.

Resolution on Clauses:

Resolution Rule for Propositional Calculus:

From $\{\lambda\} \cup \Sigma_1$ and $\{\neg \lambda\} \cup \Sigma_2$ where Σ_1 and Σ_2 are sets of literals and λ is an atom.

We can infer, $\Sigma_1 \cup \Sigma_2$ called the **resolvent** of the clauses, and the atom λ is the atom resolved upon.

The process is called **resolution**.

Examples:

1. Resolving $R \vee P$ and $\neg P \vee Q$ yields $R \vee Q$

The two clauses being resolved can be rewritten as the **implications:** $\neg R \rightarrow P$ and $P \rightarrow Q$.

A rule of inference called **chaining** applied to these implications yields: $\neg R \rightarrow Q$.

Which is equivalent to the **resolvent** $R \vee Q$. Thus, *chaining is a special case of resolution*.

2. Resolving R and $\neg R \vee P$ yields P .

Since the second clause is equivalent to $R \rightarrow P$. Thus, *modus ponens is also a special case of resolution*.

3. Resolving $R \vee Q \vee R \vee S$ with $\neg P \vee Q \vee W$ on P yields $Q \vee R \vee S \vee W$. Note that only one instance of Q appears in the resolvent.

4. Resolving $P \vee Q \vee \neg R$ with $P \vee W \vee \neg Q \vee R$ on Q yields $P \vee \neg R \vee R \vee W$.

Resolving them on R yields $P \vee Q \vee \neg Q \vee W$.

In this case, since both $\neg R \vee R$ and $Q \vee \neg Q$ have value True, the value of each of these resolvents is True. In this example, we must resolve either on Q or on R – not on both simultaneously, that is, $P \vee W$ is not a resolvent of two clauses!

8. Converting Arbitrary wffs to Conjunction of Clauses

Any wff in the propositional calculus can be converted to an equivalent conjunction of clauses.

A wff written as a conjunction of clauses is said to be in **conjunctive normal form (CNF)**.

A wff written as a disjunction of conjunction of literals is said to be in **disjunctive normal form (DNF)**.

Example:

1. wff: $\neg(P \rightarrow Q) \vee (R \rightarrow P)$

Step 1: Eliminate implication signs by using the equivalent form using \vee :

$$\neg(\neg P \vee Q) \vee (R \vee P)$$

Step 2: Reduce the scopes of \neg signs by using De Morgan's laws and by eliminating double \neg signs:

$$(P \wedge \neg Q) \vee (\neg R \vee P)$$

Step 3: Convert to CNF by using the associative and distributive laws. First,

$$(P \wedge \neg R \vee P) \wedge (\neg Q \vee \neg R \vee P) \quad \text{then,}$$

$$(P \wedge \neg R) \wedge (\neg Q \vee \neg R \vee P)$$

A conjunction of clauses (that is, the CNF form of a wff) is usually expressed as a set of clauses (with conjunction of the clauses implied); thus,

$$\{(P \wedge \neg R), (\neg Q \vee \neg R \vee P)\}$$

9. Resolution Refutations

A resolution refutation for proving an arbitrary wff w , from a set of wffs δ .

Proceed as follows:

1. Convert wffs in δ to clause form – a (conjunctive) set of clauses.
2. Convert the negation of the wff to be proved w , to clause form
3. Combine the clauses resulting from steps 1 and 2 into a single set T .
4. Iteratively apply resolution to the clauses in T and add the results to T either until there are no more resolvents that can be added or until the empty clause is produced.

Example: Robot lifting a block

Consider a robot that is able to lift a block, if that block is liftable, and if the robot's battery power source is adequate. If both these conditions are satisfied, then when the robot tries to lift a block it is holding, its arm moves. We represent the above conditions by binary-valued features:

x_1 (BAT_OK)

x_2 (LIFTABLE)

x_3 (MOVES)

Given a set of wffs δ :

1. BAT_OK
2. \neg MOVES
3. $\text{BAT_OK} \wedge \text{LIFTABLE} \rightarrow \text{MOVES}$

The clause form of the 3rd wff is

4. $\neg \text{BAT_OK} \vee \neg \text{LIFTABLE} \vee \text{MOVES}$

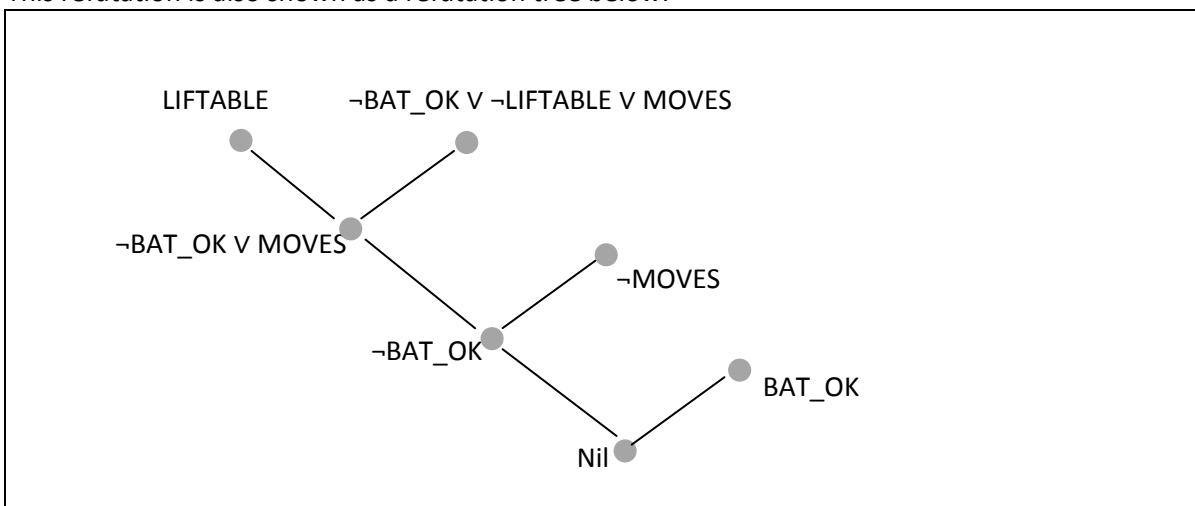
The negation of the wff to be proved yields another clause:

5. LIFTABLE

Now we perform resolutions to produce the following sequence of clauses.

6. $\neg \text{BAT_OK} \vee \text{MOVES}$ (5 with 4)
7. $\neg \text{BAT_OK}$ (6, 2)
8. NIL (6, 1)

This refutation is also shown as a refutation tree below:



Boolean Function

Definition:

Let $B = \{0, 1\}$. Then $B^n = \{(x_1, x_2, \dots, x_n) \mid x_i \in B \text{ for } 1 \leq i \leq n\}$ is the set of all possible n -tuples of 0s and 1s. The variable x is called a Boolean Variable if it assumes values only from B , that is, its only possible values are 0 and 1. A function from B^n to B is called **Boolean Function of Degree n** .

Example:

A Boolean Function of degree two is a function from a set with four elements, namely, pairs of elements from $B = \{0, 1\}$ to B , a set of two elements. Hence, there are 16 different Boolean Functions of degree two, example, $F(xy) = F_{10} = xy'$.

x	y	F ₁	F ₂	F ₃	F ₄	F ₅	F ₆	F ₇	F ₈	F ₉	F ₁₀	F ₁₁	F ₁₂	F ₁₃	F ₁₄	F ₁₅	F ₁₆
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
0	1	1	1	0	0	1	1	0	1	1	0	0	1	1	1	0	0
0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Number of Boolean Functions of degree n

Degree, n	Number of Boolean Functions, $N = 2^m$ where $m = 2^n$
1	4
2	16
3	256
4	65, 536
5	4, 294, 967, 296
6	18, 446, 744, 073, 709, 551, 616

Boolean Algebra

Boolean algebra provides the operations and the rules for working with the set $\{0, 1\}$.

Operations include complementation (\neg), sum (\vee), and product (\wedge).

Examples: $1.0 + (0 + 1)' = 0$; $(T \wedge F) \vee \neg (T \vee F) = F$;

$(T \wedge T) \vee \neg F = T$; $(1.1) + 0' = 1$

Boolean Identities

Identity	Name
$x'' = x$	Law of double compliments
$x + x = x$ $x.x = x$	Idempotent
$x + 0 = x$ $x.1 = x$	Identity
$x + 1 = 1$ $x.0 = 0$	Domination
$x + y = y + x$ $xy = yx$	Commutative
$x + (y + z) = (x + y) + z$	Associative
$x(y + z) = xy + xz$ $x + yz = (x + y)(y + z)$	Distributive
$(xy)' = x' + y'$ $(x + y)' = x'y'$	DeMorgan's
$x + xy = x$ $x(x + y) = x$	Absorption
$x + x' = 1$	Unit property
$x.x' = 0$	Zero property

Quine – Mc Clusky Method

To simplify a sum-of-products expression

1. Express each minterm in n variables by a bit string of length n with a 1 in the i^{th} position of x_i occurs and a 0 in this position if x_i' occurs.
2. Group the bit strings according to the number of 1s in them.
3. Determine all products in $n-1$ variables that can be formed by taking the Boolean sum of minterms in the expression. Minterms that can be combined are represented by bit strings that differ in exactly one position. Represent these products in $n-1$ variables with strings that have a 1 in the i^{th} position if x_i occurs in the product, a 0 in this position if x_i' occurs, and a dash in this position if there is no literal involving x_i in the product.
4. Determine all products in $n-2$ variables that can be formed by taking the Boolean sum of the products in $n-1$ variables found in the previous step. Products in $n-1$ variables that can be combined are represented by bit strings that have a dash in the same position and differ in exactly one position.
5. Continue combining Boolean products in fewer variables as long as possible.
6. Find all the Boolean products that arose that were not used for to form a Boolean product in one fewer literal.
7. Find the smallest set of these Boolean products such that sum of these products represent the Boolean functions. This is done by forming a table showing which minterms are covered by which products. Every minterm must be covered by at least one product. The first step in using this table is to find all essential prime implicants. Each essential prime implicant must be included because is the only prime implicant that covers one of the minterms. Once we have found essential prime implicants, we can simplify the table by eliminating the column for minterms covered by this prime implicant. Furthermore, we can eliminate any prime implicants that cover a subset of minterms covered by another prime implicant. Moreover, we can eliminate from the table the column for a minterm if there is another minterm that is covered by a subset of all the prime implicants that cover this minterm. This process of identifying essential prime implicants that must be included, followed by eliminating redundant prime implicants and identifying minterms that can be ignored, is iterated until the table does not change. At this point we use a backtracking procedure to find the optimal solution which we compare to the best solution found so far at each step.

Given:

Term	Bit String	Number of 1s
$wxyz'$	1110	3
$wx'yz$	1011	3
$w'xyz$	0111	3
$wx'yz'$	1010	2
$w'xy'z$	0101	2
$w'x'yz$	0011	2
$w'x'y'z$	0001	1

Use the Quine- Mc Clusky method to simplify the sum-of-products expansion:

$$wxyz' + wx'yz + w'xyz + wx'yz' + w'xy'z + w'x'yz + w'x'y'z$$

S. No	Term	Bit String	Step 1		Step 2	
			Term	String	Term	String
1	$wxyz'$	1110	(1, 4) wyz'	1-10	(3, 5, 6, 7) $w'z$	0--1
2	$wx'yz$	1011	(2, 4) $wx'y$	101-		
3	$w'xyz$	0111	(2, 6) $x'yz$	-011		
4	$wx'yz'$	1010	(3, 5) $w'xz$	01-1		
5	$w'xy'z$	0101	(3, 6) $w'yz$	0-11		
6	$w'x'yz$	0011	(5, 7) $w'y'z$	0-01		
7	$w'x'y'z$	0001	(6, 7) $w'x'z$	00-1		

	wyz'	$wx'yz$	$w'xyz$	$wx'yz'$	$w'xy'z'$	$w'x'yz$	$w'x'y'z$
$w'z$			x		x	x	x
wyz'	x			x			
$wx'y$		x		x			
$x'yz$		x				x	

Answer: $w'z + wyz' + wx'y$ or $w'z + wyz' + x'yz$

Predicate Calculus (First Order Calculus)

A language that refers to **Objects** in the world and **Propositions** about the world.

Components:

1. Object Constants: strings of alphanumeric characters. Example: Aw3.
2. Function Constants: strings of alphanumeric characters beginning always with a lower case letter and superscripted by their arity. Example: distanceBetween², times².
3. Relation Constants: strings of alphanumeric characters beginning with a capital letter and superscripted by their arity. Example: B15³, clear¹.
4. Connectives and delimiters: $\wedge, \vee, \neg, \rightarrow$
(,), [,], ,

Terms:

1. An object constant is a term.
2. A function constant of arity n, followed by n terms in parentheses and separated by commas, is a term. This type of term is called a functional expression. Example: *fatherOf*(john, Bill), *times*(4, plus(3,6)).

Wffs:

1. Atoms: a relation constant of arity n followed by n terms in parentheses and separated by commas is an atom (atomic formula). A relation constant of arity 0 omits the parentheses. An atom is a wff. Example: P(A, B, C, D), Q.
2. Propositional wffs: any expression formed out of predicate-calculus wffs in the same way that the propositional calculus forms wffs out of other wffs is a wff, called a propositional wff. Example: [Greaterthan(7, 2) \wedge Lessthan(15, 41)] \vee P.

Semantics:

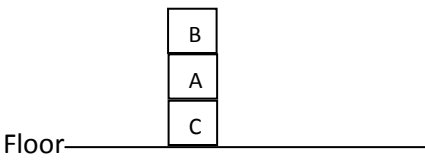
Worlds

1. The world can have infinite number of objects (individuals) in it.
2. Functions on individuals: we can have an infinite number of functions of all arities that map m tuples of individuals into individuals. Example: A function that maps a person into his/her father.
3. Relations over individuals: the individuals can participate in an arbitrary number of relations. These will each have arities. (A relation of arity 1 is called property).

Interpretations:

Interpretation of an expression is an assignment that maps object constants into objects in the world, n-ary function into n-ary functions, and n-ary relation constants into n-ary relations. These assignments are called denotations of their corresponding predicate-calculus expressions. The set of objects to which object constant assignments are made is called domain of the interpretation.

Example: Blocks World

	Individuals: A, B, C, and Floor Relations: On, Clear Representation in predicate calculus: A, B, C, Fl Binary Constant: On Unary Relation Constant: Clear
---	---

Determination of some predicate calculus wffs:

On (A, B) is False because $\langle A, B \rangle$ is not in the relation On.

Clear (B) is True because $\langle B \rangle$ is in the relation Clear.

Models and Related Notations

1. An interpretation satisfies a wff if the wff has the value True under that interpretation.
2. An interpretation that satisfies a wff is a model of that wff.
3. Any wff that has the value True under all interpretations is valid.
4. Any wff that does not have a model is inconsistent or unsatisfiable.
5. If a wff w has value True under all of those interpretations for which each of the wffs in a set Δ has value True, then Δ logically entails w .
6. Two wffs are equivalent if and only if their truth values are identical under all interpretations.

Quantification

Quantification over variable symbols gives the predicate calculus the expressive power. Quantification over relation and function symbols are not allowed in First-Order Predicate Calculus. Higher-Order Predicate Calculi allow quantification over relation and function symbols.

1. Variable symbols: strings $P1, f(x, Bob, C17)$
2. \forall Universal Quantifier
 \exists Existential Quantifier
3. $(\forall \xi)w$ and $(\exists \xi)w$ are wffs.
 $(Q\xi)w$: closed wff (closed sentence) Q : either \forall or \exists

Wff w is said to be within the scope of quantification. Variable symbol ξ is quantified over.

Rules of Inference

1. Universal Instantiation (UI)
From $(\forall \xi) w(\xi)$, infer $w(\alpha)$, where $w(\xi)$ is any wff with variables ξ , α is any constant symbol, and $w(\alpha)$ is $w(\xi)$ with α substituted for ξ throughout w .
Example: from $(\forall x) P(x, f(x), B)$ infer $P(A, f(A), B)$.
2. Existential Generalization (EG)
From $w(\alpha)$, infer $(\exists \xi) w(\xi)$, where $w(\alpha)$ is a wff containing a constant symbol α , and $w(\xi)$ is a form with ξ replacing every occurrence of α throughout w .
Example: From $(\forall x) Q(A, g(A), x)$ infer $(\exists y) (\forall x) Q(y, g(y), z)$
3. Include propositional-calculus rules of inference
 1. modem ponens
 2. \wedge introduction and elimination
 3. \vee introduction
 4. \neg elimination
 5. Resolution

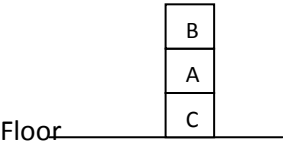
Predicate Calculus as a language for representing knowledge.

Examples:

1. All packages in room 27 are smaller than any of the packages in room 28.
 $(\forall x, y) \{ [\text{package}(x) \wedge \text{Package}(y) \wedge \text{Inroom}(x, 27) \wedge \text{Inroom}(y, 28)] \rightarrow \text{Smaller}(x, y) \}$
2. Every package in room 27 is smaller than one of the packages in room 29.
 $(\exists y)(\forall x) \{ [\text{package}(x) \wedge \text{Package}(y) \wedge \text{Inroom}(x, 27) \wedge \text{Inroom}(y, 29)] \rightarrow \text{Smaller}(x, y) \}$
 $(\forall x)(\exists y) \{ [\text{package}(x) \wedge \text{Package}(y) \wedge \text{Inroom}(x, 27) \wedge \text{Inroom}(y, 29)] \rightarrow \text{Smaller}(x, y) \}$

Three Blocks-World Knowledge Representation and Reasoning

A, B, C, Floor are Object Constants. On is a Binary Relation Constant, and Clear is a Unary Relation Constant.

	<p>Predicate Object Constants</p> <p>Calculus Relation</p> <p>A B C Fl On Clear</p>	<p>World Objects</p> <p>A B C Floor On = {< >, < >} Clear = {(B)}</p>	<p>On(C, Fl) \wedge \negOn(A, B) is True. Because both C, Fl) and \negOn(A, B) are True.</p> <ol style="list-style-type: none"> 1. \negClear(C) \wedge \negClear(A) \rightarrow Clear(B) 2. On(B, A) \wedge On(A, C) \rightarrow On(C, Fl)
<p>Figure 1. A Configuration of Blocks World</p>	<p>Table 1. Mapping between Predicate Calculus and the World.</p>		<p>Figure 2. A set of Formulas embodying knowledge</p>

Prolog

Prolog is an acronym for Programming in Logic by Alain Colmeraur, 1970 (Univ. of France). David H. D. Warren (Scotland) wrote first compiler. It is a declarative language and is an approximate implementation of logic programming language on a sequential machine. Logic program is a set of clauses. Prolog views set of clauses as a sequence of Horn Clauses. A Horn Clause is a clause with at most one positive literal. PARALOG, concurrent Prolog, GHC are based on parallel execution.

In Prolog, the Horn clauses function as statements of the language and are written in the following formats:

Rules: $\lambda_h :- \lambda_{b1}, \dots, \lambda_{bn}$

(which is a special way of writing the implication $\lambda_{b1} \wedge \dots \wedge \lambda_{bn} \rightarrow \lambda_h$), where each λ_i is a positive literal. The literal λ_h is called the head of the clause, and the ordered list of literals, $\lambda_{b1}, \dots, \lambda_{bn}$ is called the body.

Facts: $\lambda_h :-$

Goals: $-\lambda_{b1}, \dots, \lambda_{bn}$

The literals in goals and in the bodies of rules are ordered lists, and this order plays an important role in execution of a Prolog program.

Prolog program is expressed in terms of program clauses (rules and facts) and the goals are solved using these clauses. An interpreter is a program that can take another program as input. An interpreter is known as meta interpreter for a language if it is written in the same language. A meta interpreter for Prolog may be used for generation of Proof Tree. Prolog interpreter uses the resolution refutation method. If it succeeds in refuting (contradiction), the goal is said to be a logical consequence of the Prolog program. Both recursive and iterative programming can be attempted in Prolog. Prolog uses the following control strategies:

1. Forward movement
2. Unification (Matching)
3. Backtracking – Deep Backtracking, and Shallow Backtracking

Recursive Data Types are implemented in Prolog. Linked Lists and Binary Trees can be implemented.

The inference over Prolog clauses consists of attempting to 'prove' a goal clause and is performed by executing a Prolog program. Such a Proof is achieved by resolution-like operations performed on Prolog facts, goals, and rules. Each resolution is performed between a goal and either a fact or a rule:

1. A goal can resolve with a fact by unifying the fact with one of the literals in the goal. We call this literal the one resolved upon. The resolvent is a new goal consisting of a list of all of the substitution instances of the other literals in the original goal written in the same order as in original goal. The substitution instances are obtained by applying the mgu of the unification to all of these other literals.
2. A goal can resolve with rule by unifying the head of the rule with one of the literals in the goal. The resolvent is a new goal formed by appending the list of substitution instances of all of the literals in the body of the rule to the front of the list of substitution instances of all of the other (non-resolved-upon) literals in the goal.

Prolog handles two major issues:

1. Choosing the first sub goal from the resolvent. (Uses depth-first strategy while satisfying sub goals in the resolvent. Order of rules and goals are significant and lead to different results with different orders)
2. Choosing the first clause that matches when the clauses are sequentially searched from top to bottom in the program.

Issues in Prolog

1. Redundancy – generation of same solution many times
2. Termination – fails to find the solution using depth first strategy with infinite branch that does not terminate computation.

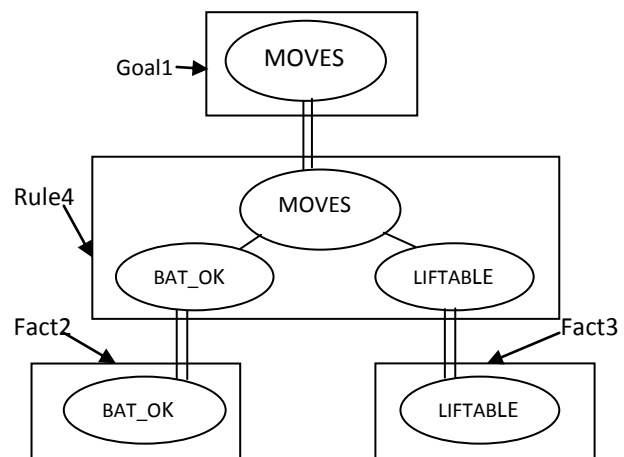
Syntax

1. Rules and facts are terminated by a full stop (.)
2. Prolog goal (query) is written after the symbol ?- and terminated by full stop (.). A goal may be simple or conjunction of sub goals.
3. Variable names must start with uppercase alphabets and may be constituted by letters, numerals, and the underscore symbol (_).
4. Constants may be in the form of integers (such as 4), symbols (such as mary), and strings ('This is a string'). Strings are always enclosed within single quotes.
5. Function and Predicate names must start with lower case alphabets and are formed using lowercase letters, numerals, and the underscore symbol (_).
6. All the rules and facts of the same predicate name should appear together in the program.

A Prolog program for proving that the arm moves, given that the block is liftable and the battery is charged:

1. :- MOVES
2. BAT_OK :-
3. LIFTABLE :-
4. MOVES :- BAT_OK, LIFTABLE

AND-OR PROOF TREE



Assignment:

1. Study Reference Manual and prepare a summary note not exceeding two pages bringing out salient features offered in SWI Prolog version 7.4, January 2017, University of Amsterdam.
2. Write a Prolog program and print the solution using the notes provided on Four disc Towers-of-Hanoi on the reverse of this page.

Simple Prolog Programs

cs.toronto.edu

1. Here are some simple clauses.

```
likes(mary,food).  
likes(mary,wine).  
likes(john,wine).  
likes(john,mary).
```

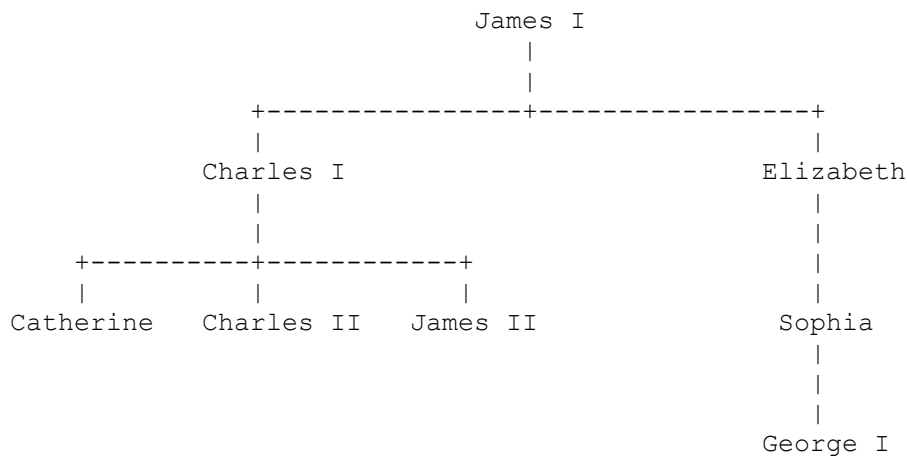
The following queries yield the specified answers.

```
| ?- likes(mary,food).  
yes.  
| ?- likes(john,wine).  
yes.  
| ?- likes(john,food).  
no.
```

How do you add the following facts?

1. John likes anything that Mary likes
2. John likes anyone who likes wine
3. John likes anyone who likes themselves

2. Slightly more complicated family tree.



Here are the resultant clauses:

```
male(james1).
male(charles1).
male(charles2).
male(james2).
male(georgel).

female(catherine).
female(elizabeth).
female(sophia).

parent(charles1, james1).
parent(elizabeth, james1).
parent(charles2, charles1).
parent(catherine, charles1).
parent(james2, charles1).
parent(sophia, elizabeth).
parent(georgel, sophia).
```

Here is how you would formulate the following queries:

```
Was George I the parent of Charles I?
    Query: parent(charles1, georgel).
Who was Charles I's parent?
    Query: parent(charles1,X).
Who were the children of Charles I?
    Query: parent(X,charles1).
```

Now try expressing the following rules:

```
M is the mother of X if she is a parent of X and is female
F is the father of X if he is a parent of X and is male
X is a sibling of Y if they both have the same parent.
```

Furthermore add rules defining:

```
"sister", "brother",
"aunt", "uncle",
"grandparent", "cousin"
```

3. Recursion: Towers of Hanoi

The 3-disk setup is like this:

```
      |      |      |
      xxx    |      |
      xxxxxx |      |
      xxxxxxx|      |
```

Here's a sample:

```
% move(N,X,Y,Z) - move N disks from peg X to peg Y, with peg Z being the
%                  auxilliary peg
%
% Strategy:
% Base Case: One disc - To transfer a stack consisting of 1 disc from
%   peg X to peg Y, simply move that disc from X to Y
% Recursive Case: To transfer n discs from X to Y, do the following:
%   Transfer the first n-1 discs to some other peg X
%   Move the last disc on X to Y
%   Transfer the n-1 discs from X to peg Y

move(1,X,Y,_) :-
    write('Move top disk from '),
    write(X),
    write(' to '),
    write(Y),
    nl.
move(N,X,Y,Z) :-
    N>1,
    M is N-1,
    move(M,X,Z,Y),
    move(1,X,Y,_),
    move(M,Z,Y,X).
```

- note the use of "anonymous" variables _

Here is what happens when Prolog solves the case N=3.

```
?- move(3,left,right,center).
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from center to left
Move top disk from center to right
Move top disk from left to right
```

yes

4. An example using lists:

(a) length of a list

```
size([],0).
size([H|T],N) :- size(T,N1), N is N1+1.
% or size([_|T],N) :- size(T,N1), N is N1+1.

| ?- size([1,2,3,4],N).
```

N = 4

```
yes
| ?- size([bill,ted,ming,pascal,nat,ron],N).
```

N = 6

```
yes
| ?- size([a, [b, c, d], e, [f | g], h], N).
```

N = 5

yes

(b) summing elements of a list of numbers

```
sumlist([],0).
ssumlist([H|T],N) :- sumlist(T,N1), N is N1+H.
```

(c) list membership

```
member(X,[X|_]).
member(X,[_|T]) :- member(X,T).
```

(d) reversing a list

```
reverse(List, Reversed) :-
    reverse(List, [], Reversed).

reverse([], Reversed, Reversed).
reverse([Head|Tail], SoFar, Reversed) :-
    reverse(Tail, [Head|SoFar], Reversed).
```

```
| ?- myreverse([a,b,c,d],X).
```

X = [d,c,b,a]; <- note semicolon (more solns?)

no

```
| ?- myreverse([a,b,c,d],[d,b,c,a]).
```

no

```
| ?- myreverse([a,b,c,d],[d,c,b,a]).
```

yes

- note difference between reverse/2 and reverse/3

- reverse/3 probably should be called reverseHelper or something else for clarity

Knowledge Based Systems

Three major theoretical properties of logical reasoning systems are:

1. Soundness: for an inferred conclusion is true.
2. Completeness: for an inference will eventually produce a true conclusion.
3. Tractability: for an inference be feasible.

If wff w is not logically entailed by a Δ (set of wffs), the resolution refutation procedure might never terminate. Thus resolution cannot be used as a full decision procedure.

It can be shown that there is no other method that will always tell us that a wff w does not logically follow from a set of wffs Δ when it doesn't. Because of this fact, we say that the predicate calculus is semi-decidable. Semi-decidability makes the predicate calculus inherently intractable. Even on problems for which resolution refutation terminates, the procedure is NP-hard. Although many reasoning problems can be formulated as problems of resolution refutation, the method is intractable for very large problems.

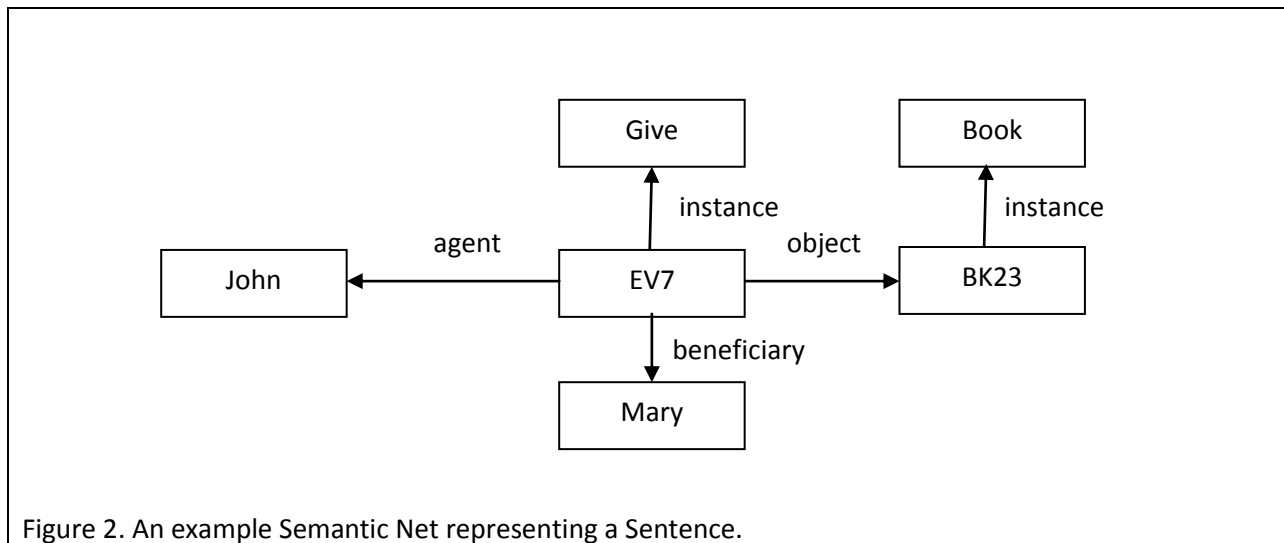
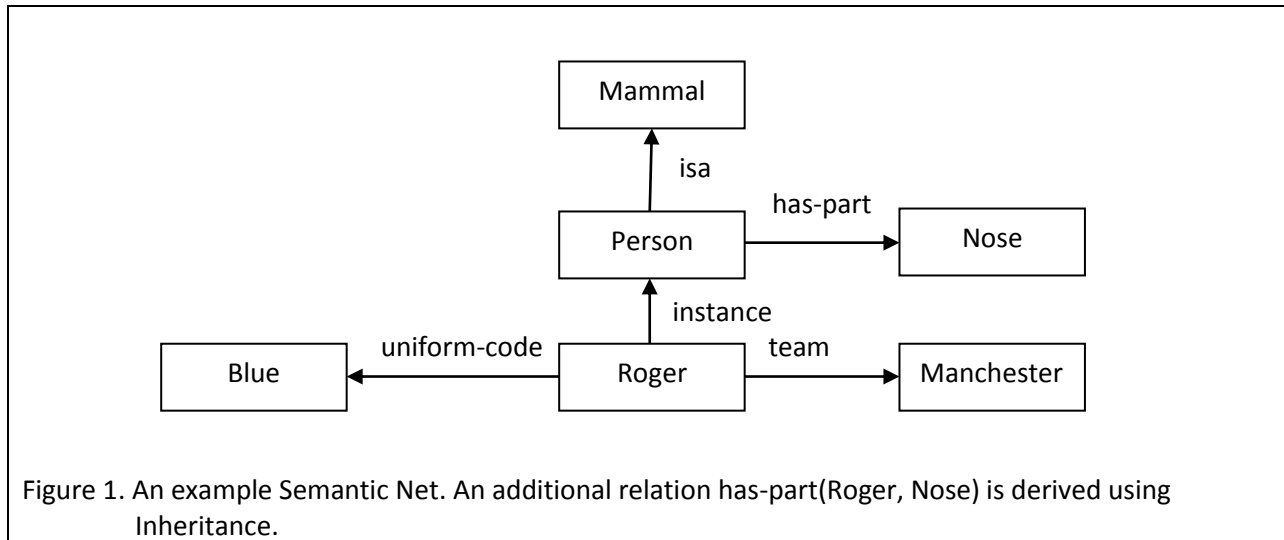
With relaxation on three properties above, a less expressive language than the predicate calculus is sufficient for many applications. Reasoning using Horn clauses is more efficient. Horn clauses form the basis of the programming language Prolog.

Horn clause: A clause having at most one positive literal.

1. If there is at least one negative literal and a single positive literal, the Horn clause can be written as an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal. Such a clause is called a rule.
2. There may be no negative literal in the clause, in which case, we write it as an implication whose antecedent is empty and whose consequent is a single positive literal. Such a clause is called a fact.
3. There may be no positive literal in the clause, in which case, we write it as an implication whose consequent is empty and whose antecedent is a list of positive literals. Such a clause is called a goal.

Semantic Nets

The main idea behind semantic nets is that the meaning of a concept comes from the ways in which it is connected to other concepts. In a semantic net, information is represented as a set of nodes connected to each other by a set of labeled arcs, which represent relationships among the nodes.



Frames

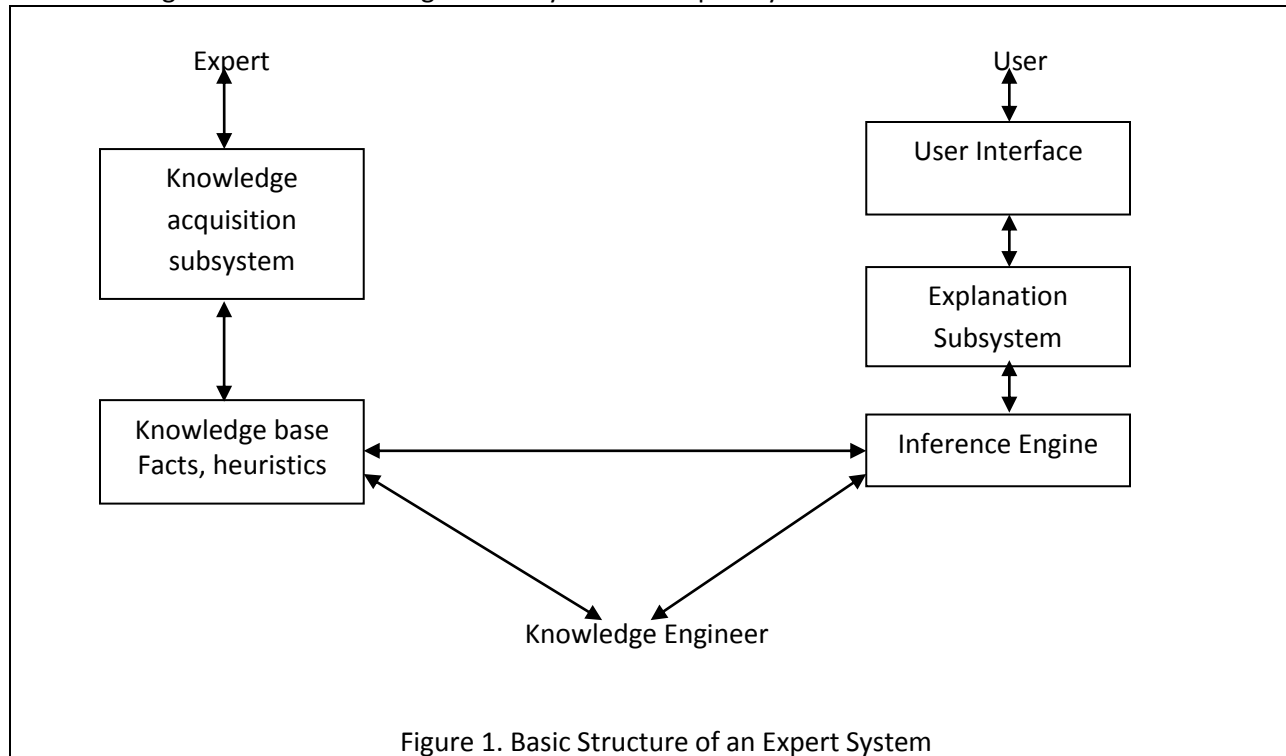
A frame is a collection of attributes (usually called slots) and associated values (and possibly constraints on values) that describe some entity in the world. Sometimes it represents the entity from a particular point of view. A single frame is rarely useful. The value of attribute of one frame may be another frame.

Person	
Isa:	Mammal
Cardinality:	6, 000, 000, 000
*handed:	Right
Adult-Male	
Isa:	Person
Cardinality:	2, 000, 000, 000
*handed:	5-10
ML-Baseball-Player	
Isa:	Adult-Male
Cardinality:	624
*height:	6-1
*bats:	equal to handed
Fielder	
Isa:	ML-Baseball-Player
Cardinality:	376
*batting-average:	.262
Pee-Wee-Reese	
Instance:	Fielder
Height:	5-10
Bats:	Right
ML-Baseball-Team	
Isa:	Team
Cardinality:	26
*team-size:	24
Brooklyn-Dodgers	
Instance:	ML-Baseball-Team
Team-size:	24
Manager:	Leo-Durocher

A Simplified Frame Structure

Rule Based Expert Systems

AI programs that achieve expert level competence in solving problems by bringing to bear a body of knowledge are called knowledge-based systems or expert systems.



Rule-based expert systems are often based on reasoning with propositional logic Horn clauses (perhaps with some kind of additional mechanism for dealing with uncertainty). The knowledge base consists of rules gathered from experts.

In many applications, the system has access only to uncertain rules, and the user may also not be able to answer questions with certainty.

Example: MYCIN system [Shortliffe 1976] – bacterial infections diagnosis

Rule 300

If:

1. The infection which requires therapy is meningitis, and
2. The patient does have evidence of serious skins or soft tissue infection, and
3. Organisms were not seen on the stain of the culture, and
4. The type of the infection is bacterial

Then:

There is evidence that the organism (other than those seen on cultures or smears) which might be causing the infection is staphylococcus-coag-pos (0.75); streptococcus-group-a (0.5).

The numbers 0.75 and 0.5 in MYCIN above represent the certainty or strength of a rule. They are used by the system in computing the certainty of conclusions.

Shell: The E.S shell simplifies the process of creating a knowledge base. It is the shell that actually processes the information entered by a user relates it to the concepts contained in the knowledge base and provides an assessment or solution for a particular problem. Thus E.S shell provides a layer between the user interface and the computer O.S to manage the input and output of the data. It also manipulates the information provided by the user in conjunction with the knowledge base to arrive at a particular conclusion.

CLIPS – a tool for building Expert System <http://clipsrules.sourceforge.net/>

About CLIPS

Developed at NASA's Johnson Space Center from 1985 to 1996, the 'C' Language Integrated Production System (CLIPS) is a rule-based programming language useful for creating expert systems and other programs where a heuristic solution is easier to implement and maintain than an algorithmic solution. Written in C for portability, CLIPS can be installed and used on a wide variety of platforms. Since 1996, CLIPS has been available as public domain software.

Notable CLIPS applications

- An Intelligent Training System for Space Shuttle Flight Controllers, IAAI-89
- Applications of Artificial Intelligence To Space Shuttle Mission Control, IAAI-89
- HUB SIAASHING: A Knowledge-Based System for Severe, Temporary Airline Schedule Reduction, IAAI-92
- PI-in-a-Box: A Knowledge-based System for Space Science Experimentation, IAAI-93
- The DRAIR Advisor: A Knowledge-Based System for Materiel Deficiency Analysis, IAAI-93
- The Multimission VICAR Planner: Image Processing for Scientific Data, IAAI-95
- IMPACT: Development and Deployment Experience of Network Event Correlation Applications, IAAI-95
- The NASA Personnel Security Processing Expert System, IAAI-96
- Expert System Technology for Nondestructive Waste Assay, IAAI-98
- Hybrid knowledge based system for automatic classification of B-scan images from ultrasonic rail inspection, IAAI-98
- An Expert System for Recognition of Facial Actions and their Intensity, IAAI-00
- Loads-n-Limits and Release-n-Sequence: The "Brains" behind WEPS, IAAI-05
- Development of a Hybrid Knowledge-Based System for Multiobjective Optimization of Power Distribution System Operations, IAAI-05

Experiment 1: Development of Rule-Base

1. It is possible to generate Rules using a Set of Facts from a given Domain and that these Rules are capable of validating the Queries on the Set of Facts.
2. Are these Rules capable of validating Queries that are not based on the Set of Facts (above) but belong to the same Domain? Explore.

Domain: Eating

Facts:

Rough Form	Proper Form
<u>Eating good food</u> keeps <u>health</u>	Ramu eats good food Ramu is healthy Siva eats good food Siva is healthy Rajani eats good food Rajani is healthy John eats good food John is healthy
<u>Eating when hungry</u> makes <u>life better</u>	Ramu eats when hungry Rajani eats when hungry
<u>Eating food</u> makes us <u>survive</u>	Siva is alive Ramu is alive John is alive
<u>Eating food</u> makes us <u>grow</u>	Ramu grows Rajani grows
<u>Eating food more</u> than required leads to <u>obesity</u>	Jhonny eats more food Prabhu eats more food Jhonny is obese Prabhu is obese
<u>Obesity</u> leads to <u>bad health</u>	Prabhu is not healthy Jhonny is not healthy
<u>Gulping food without chewing</u> leads to <u>bad health</u>	Srinu gulps food to eat Srinu does not chew food to eat Ramu chews food to eat Siva chews food to eat Jhonny gulps food to eat Prabhu gulps food to eat
Eating <u>natural food</u> keeps <u>body</u> and <u>mind healthy</u>	Ramu eats natural food Siva eats natural food Rajani eats natural food
<u>Not eating any food</u> for days is <u>dangerous to life</u>	Jhanavi is not eating food for some days Raghava is not eating food for some days
Eating <u>junk materials</u> is <u>dangerous to life</u>	Madhav eats junk material Raghava eats junk material
Junk materials are not <u>food</u>	Junk materials are not food
<u>Wheat</u> is good food	Wheat is food Rice is food Cereals are food
<u>Tobacco</u> is <u>junk material</u>	Tobacco is junk material Pan Masala is junk material Cocaine is junk material
<u>Vegetables</u> are good <u>food</u>	Vegetables are good food
<u>Fruits</u> are good <u>food</u>	Fruits are good food
<u>Apple</u> is a <u>fruit</u>	Apple is a fruit Banana is a fruit Orange is a fruit
Beans is a vegetable	Beans is a vegetable Brinjal is a vegetable Tomato is a vegetable

Person	Eats Food									Health		
	What		When		How		Quantity			Alive		Leads to Death
	G	B	H	NH	C	G	N	R	MR	Healthy	Not Healthy	Dangerous
Ramu	Apple		H		C			R		T		
Siva	Beans		H		C			R		T		
Rajani	Wheat		H		C			R		T		
John	Oranges		H		C			R		T		
Jhonny	Rice			NH					MR		T	
Prabhu		Tobacco		NH	C				MR		T	
Srinu			H				N					T
Jhanhavi			H				N					T
Raghava		Pan Masala	H		C				MR		T	
Madhav		Cocaine				G		R			T	

G: Good Food B: Bad Food H: Hungry NH: Not Hungry C: Chews G: Gulps N: No Food for Many Days R: Required Quantity MR: More than Required Quantity T: True

Table 1. Summary of Facts for development of Rules

Rules: (Using Set Theory and Logic – Propositional/Predicate)

Rules: (Reasoning with insufficient information i.e., with less than the required set of attributes and their values requires Rough Sets and Rough Logic)

