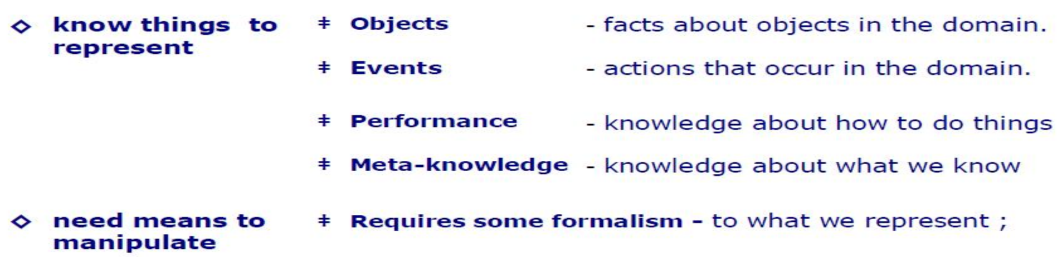
Artificial Intelligence
Knowledge Representation

Representation

- AI agents deal with knowledge (data)
  - Facts (believe & observe knowledge)
  - Procedures (how to knowledge)
  - Meaning (relate & define knowledge)
- Right representation is crucial
  - Early realisation in AI
  - Wrong choice can lead to project failure
  - Active research area

KNOWLEDGE REPRESENTATION FRAMEWORK

- Problem solving requires *large amount of knowledge and some mechanism for manipulating that knowledge.*
- The Knowledge and the Representation are distinct entities, play a central but distinguishable roles in intelligent system.
  - – *Knowledge is a description of the world;* it determines a *system's competence by what it knows.*
  - – *Representation is the way knowledge is encoded;* it defines the *system's performance in doing something.*
- In simple words, we *:*
  - – need to know about *things we want to represent , and*
  - – need some means by which *things we can manipulate.*



◇ **know things to represent**	‡ **Objects**	- facts about objects in the domain.
	‡ **Events**	- actions that occur in the domain.
	‡ **Performance**	- knowledge about how to do things
	‡ **Meta-knowledge**	- knowledge about what we know
◇ **need means to manipulate**	‡ **Requires some formalism** - to what we represent ;	

- Thus, knowledge representation can be considered at two levels :
- (a) *knowledge level at which facts are described, and*
- (b) *symbol level at which the representations of the objects, defined in* terms of symbols, can be manipulated in the programs.
- Note : A good representation enables fast and accurate access to knowledge and understanding of the content.

Choosing a Representation

- For certain problem solving techniques
  - ‘Best’ representation already known
  - Often a requirement of the technique
  - Or a requirement of the programming language (e.g. Prolog)
- Examples
  - First order theorem proving… first order logic
  - Inductive logic programming… logic programs
  - Neural networks learning… neural networks
- Some general representation schemes
  - Suitable for many different (and new) AI applications

Some General Representations

1. Logical Representations
2. Production Rules
3. Semantic Networks
   - Conceptual graphs, frames
4. *Description Logics*

What is a Logic?

- A language with concrete rules
  - No ambiguity in representation (may be other errors!)
  - Allows unambiguous communication and processing
  - Very unlike natural languages e.g. English
- Many ways to translate between languages
  - A statement can be represented in different logics
  - And perhaps differently in same logic
- **Expressiveness** of a logic
  - How much can we say in this language?
- Not to be confused with logical reasoning
  - Logics are languages, reasoning is a process (may **use** logic)

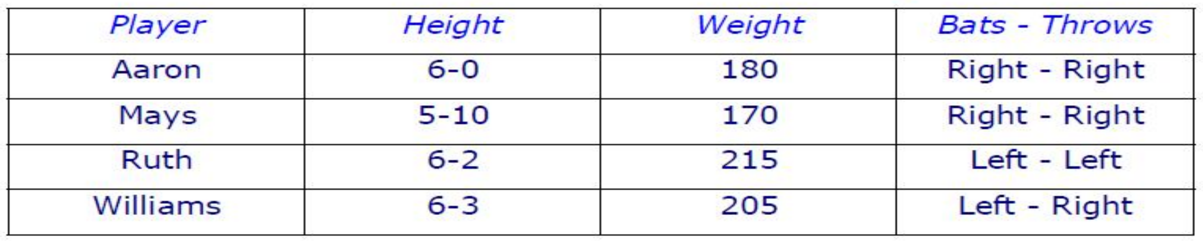
Knowledge Representation schemes

- There are four types of Knowledge representation - *Relational, Inheritable, Inferential, and Declarative/Procedural.*
- ◊ **Relational Knowledge :**
- – provides a framework to compare two objects based on equivalent attributes.
- – any instance in which two different objects are compared is a relational type of knowledge.
- ◊ **Inheritable Knowledge**
- – is obtained from associated objects.
- – it prescribes a structure in which new objects are created which may inherit all or a subset of attributes from existing objects.
- ◊ **Inferential Knowledge**
- – is inferred from objects through relations among objects.
- – e.g., a word alone is a simple syntax, but with the help of other words in phrase the reader may infer more from a word; this inference within linguistic is called semantics.
- ◊ **Declarative Knowledge**
- – a statement in which knowledge is specified, but the use to which that knowledge is to be put is not given.
- – e.g. laws, people's name; these are facts which can stand alone, not dependent on other knowledge;
- **Procedural Knowledge**
- – a representation in which the control information, to use the knowledge, is embedded in the knowledge itself.
- – e.g. computer programs, directions, and recipes; these indicate specific use or implementation;

RELATIONAL KNOWLEDGE

- Used to associate elements of one domain with the elements of another domain or set of design constrains.
  - – Relational knowledge is made up of objects consisting of attributes and their corresponding associated values.
  - – The results of this knowledge type is a mapping of elements among different domains.
- The table below shows a simple way to store facts.
  - – The facts about a set of objects are put systematically in columns.
  - – This representation provides little opportunity for inference.

Table - Simple Relational Knowledge



Player	Height	Weight	Bats - Throws
Aaron	6-0	180	Right - Right
Mays	5-10	170	Right - Right
Ruth	6-2	215	Left - Left
Williams	6-3	205	Left - Right

- Given the facts it is not possible to answer simple question such as :
- *" Who is the heaviest player ? ".*
- **But if a procedure for finding heaviest player is provided, then these** facts will enable that procedure to compute an answer.

INHERITABLE KNOWLEDGE

- • **Inheritable knowledge : *elements inherit attributes from their parents.***
- The knowledge is embodied in the design hierarchies found in the functional, physical and process domains. Within the hierarchy, elements inherit attributes from their parents, but in many cases, not all attributes of the parent elements be prescribed to the child elements.
  - – The basic KR needs to be augmented with *inference mechanism, and*
  - – Inheritance is a powerful form of inference, but not adequate.
- The KR in hierarchical structure, shown below, is called *"semantic network"* or a collection of *"frames" or "slot-and-filler structure". It shows property* inheritance and way for insertion of additional knowledge.
- – Property inheritance : Objects/elements of specific classes inherit attributes and values from more general classes.
- – Classes are organized in a generalized hierarchy.

The directed arrows represent attributes (*isa, instance, and team) originating* at the object being described and terminating at the object or its value.
The box nodes represents objects and values of the attributes

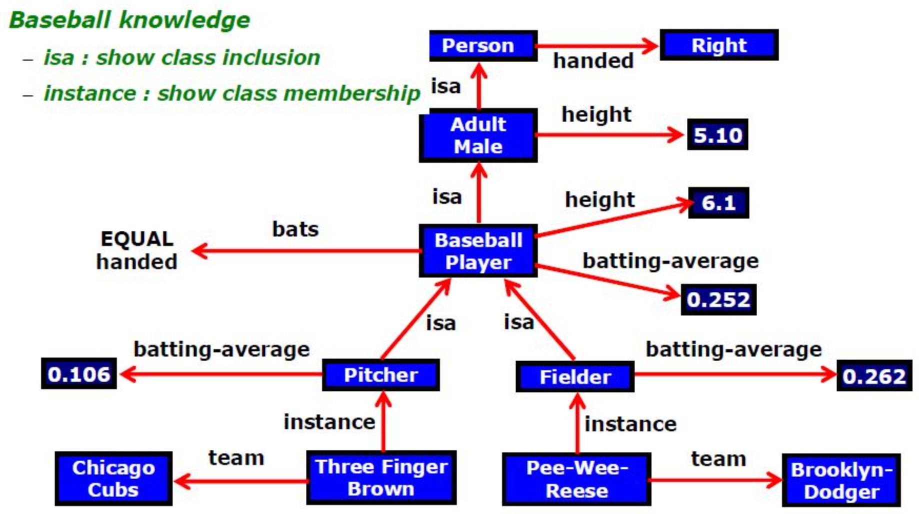


Fig. Inheritable knowledge representation (KR)

INFERENTIAL KNOWLEDGE

- Generates new information from the given information. This new information does not require further data gathering form source, but does require analysis of the given information to generate new knowledge.
  - – Given a set of relations and values, one may infer other values or relations.
  - – In addition to algebraic relations, a predicate logic (mathematical deduction)
- is used to infer from a set of attributes.
  - – Inference through predicate logic uses a set of logical operations to relate individual data. The symbols used for the logic operations are :
- ***" → " (implication), " ¬ " (not), " V " (or), " Λ " (and),***
- ***" ∀ " (for all), " ∃ " (there exists).***
- Examples of predicate logic statements :

1. Wonder is a name of a dog :	***dog (wonder)***
2. All dogs belong to the class of animals :	$\forall$ ***x : dog (x) → animal(x)***
3. All animals either live on land or in water :	$\forall$ ***x : animal(x) → live (x, land) V live (x, water)***

- We can infer from these three statements that :
- *" Wonder lives either on land or on water."*
- As more information is made available about these objects and their
- relations, more knowledge can be inferred.

Syntax and Semantics

- Syntax
  - Rules for constructing legal sentences in the logic
  - Which symbols we can use (English: letters, punctuation)
  - How we are allowed to combine symbols
- Semantics
  - How we interpret (read) sentences in the logic
  - Assigns a meaning to each sentence
- Example: “All lecturers are seven foot tall”
  - A valid sentence (syntax)
  - And we can understand the meaning (semantics)
  - This sentence happens to be false (there is a counterexample)

Declarative/Procedural knowledge

- The difference between Declarative/Procedural knowledge is not very clear.
- **Declarative knowledge :**
- Here, the knowledge is based on declarative facts about axioms and domains.
- – axioms are assumed to be true unless a counter example is found to invalidate them.
- – domains represent the physical world and the perceived functionality.
- – axiom and domains thus simply exists and serve as declarative statements that can stand alone.
- **Procedural knowledge:**
- Here the knowledge is a mapping process between domains that specifies
- "what to do when" and the representation is of *"how to make it" rather than*
- *"what it is". The procedural knowledge :*
- – may have inferential efficiency, but no inferential adequacy and acquisitional efficiency.
- – are represented as small programs that know how to do specific things,
- how to proceed.
- Example : a parser in a natural language has the knowledge that a noun phrase may contain articles, adjectives and nouns. It thus accordingly call routines that know how to process articles, adjectives and nouns.

LOGIC

- **Assumptions about KR**
  - – Intelligent Behavior can be achieved by manipulation of symbol structures.
  - – KR languages are designed to facilitate operations over symbol structures, have precise syntax and semantics;
    - Syntax tells which expression is legal ?, *e.g., red1(car1), red1 car1, car1(red1), red1(car1 & car2) ?; and*
    - Semantic tells what an expression means ? *e.g., property "dark red" applies to my car.*
  - – Make Inferences, draw new conclusions from existing facts.
- To satisfy these assumptions about KR, we need *formal notation that allow* automated inference and problem solving. ***One popular choice is use of logic.***

LOGIC

- *Logic is concerned with the truth of statements about the world.*
- Generally each statement is either *TRUE or FALSE.*
- Logic includes : *Syntax , Semantics and Inference Procedure.*
  - ◊ **Syntax :**
    - Specifies the *symbols in the language about how they can be combined* to form sentences. The facts about the world are represented as sentences in logic.
  - ◊ **Semantic :**
    - Specifies how to assign a truth value to a sentence based on its *meaning in the world. It Specifies what facts a sentence refers to. A* fact is a claim about the world, and it may be *TRUE or FALSE.*
  - ◊ **Inference Procedure :**
    - Specifies *methods for computing new sentences from an existing* sentences.
- **Note :**
  - *Facts are claims about the world that are True or False.*
  - *Representation is an expression (sentence), stands for the objects and relations.*
  - *Sentences can be encoded in a computer program*

Logic

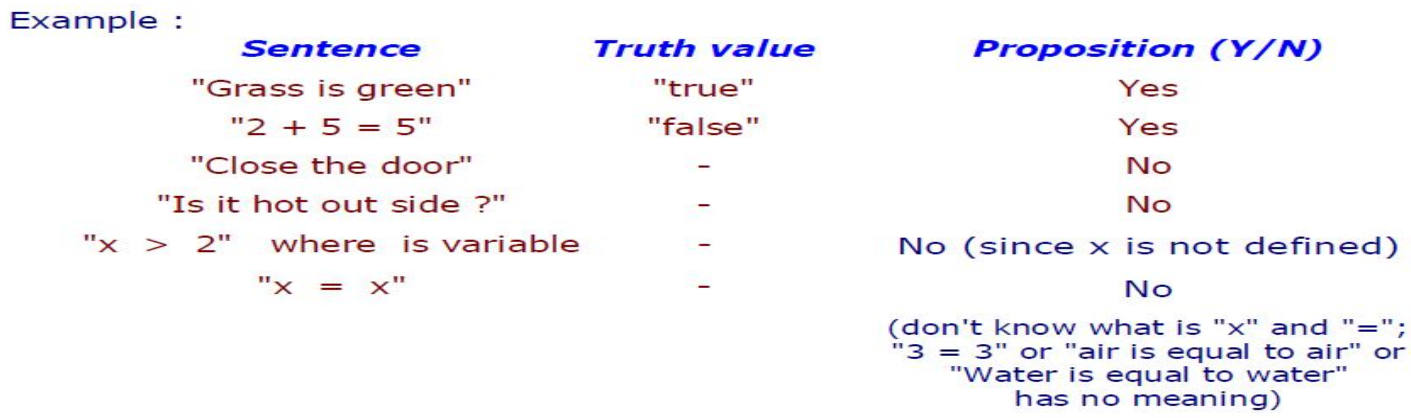
- • **Logic as a KR Language**
- *Logic is a language for reasoning, a collection of rules used while doing* logical reasoning. Logic is studied as KR languages in artificial intelligence.
- ◊ **Logic is a formal system in which the formulas or sentences have true** or false values.
- ◊ **The problem of designing a KR language is a tradeoff between that** which is :
- (a) *Expressive enough to represent important objects and relations in* a problem domain.
- (b) *Efficient enough in reasoning and answering questions about* implicit information in a reasonable amount of time.
- ◊ **Logics are of different types : *Propositional logic, Predicate logic, Temporal logic, Modal logic, Description logic etc;***
- They represent things and allow more or less efficient inference.
- ◊ ***Propositional logic and Predicate logic are fundamental to all logic.***
- *Propositional Logic is the study of statements and their connectivity.*
- *Predicate Logic is the study of individuals and their properties.*

LOGIC REPRESENTATION

- The *Facts are claims about the world that are True or False.*
- *Logic can be used to represent simple facts.*
- **To build a Logic-based representation :**
- ◊ **User defines a set of primitive *symbols and the associated semantics.***
- ◊ **Logic defines ways of putting symbols together so that user can define** legal *sentences in the language that represent TRUE facts.*
- ◊ **Logic defines ways of inferring *new sentences from existing ones.***
- ◊ **Sentences - either *TRUE or false but not both are called propositions.***
- ◊ **A declarative sentence expresses a *statement with a proposition as***
- content; example:
- the declarative *"snow is white" expresses that snow is white;* further, *"snow is white" expresses that snow is white is TRUE.*

Propositional Logic (PL)

- *A proposition is a statement, which in English would be a declarative sentence. Every proposition is either TRUE or FALSE.*
- Examples: (a) The sky is blue., (b) Snow is cold. , (c) 12 * 12=144
- **propositions are "sentences" , either true or false but not both.**
- **a sentence is smallest unit in propositional logic.**
- **if proposition is true, then truth value is "true" .**
- if proposition is false, then truth value is "false" .



Example :

Sentence	Truth value	Proposition (Y/N)
"Grass is green"	"true"	Yes
"2 + 5 = 5"	"false"	Yes
"Close the door"	-	No
"Is it hot out side ?"	-	No
"x > 2" where is variable	-	No (since x is not defined)
"x = x"	-	No (don't know what is "x" and "="; "3 = 3" or "air is equal to air" or "Water is equal to water" has no meaning)

– *Propositional logic is fundamental to all logic.*
– *Propositional logic is also called Propositional calculus, Sentential calculus, or Boolean algebra.*
– *Propositional logic tells the ways of joining and/or modifying entire* propositions, statements or sentences to form more complicated propositions, statements or sentences, as well as the logical relationships and properties that are derived from the methods of combining or altering statements

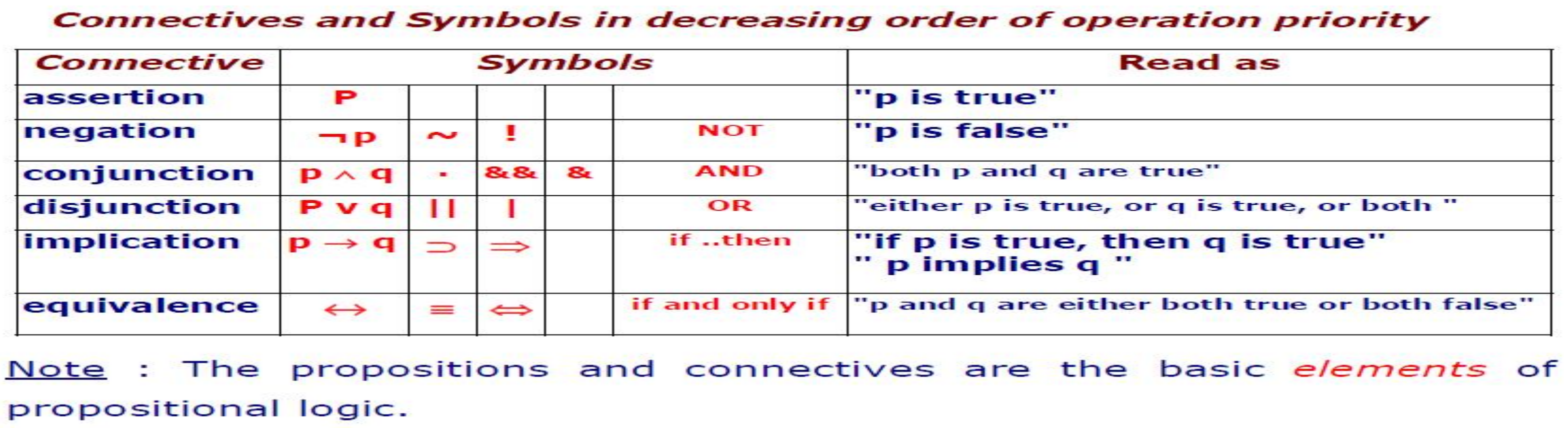
PROPOSITIONAL LOGIC

- Syntax
  - Propositions, e.g. “it is wet”
  - Connectives: and, or, not, implies, iff (equivalent)

    $\wedge \quad \vee \quad \neg \quad \longrightarrow \quad \longleftrightarrow$
  - Brackets, T (true), and F (false)
- Semantics (Classical AKA Boolean)
  - Define how connectives affect truth
    - “P and Q” is true if and only if P is true and Q is true
  - Use **truth tables** to work out the truth of statements

Propositional Logic

- **■ Statement, variables and symbols**
- These and few more related terms, such as, *connective, truth value***,** *contingencies, tautologies, contradictions, antecedent, consequent and argument are explained below.*
- **◊ Statement**
  - *Simple statements (sentences), TRUE or FALSE, that does not* contain any other statement as a part, are *basic propositions;*
  - lower-case letters, *p, q, r, are symbols for simple statements.*
  - *Large, compound or complex statement are constructed from basic* propositions by combining them with *connectives.*
- **◊ Connective or Operator**
- The *connectives join simple statements into compounds, and joins* compounds into larger compounds.
- Table below indicates, five basic connectives and their symbols :
  - – listed in decreasing order of operation priority;
  - – operations with higher priority is solved first.
- **Example of a formula : $((((a \land \neg b) \lor c \to d) \leftrightarrow \neg (a \lor c))$**

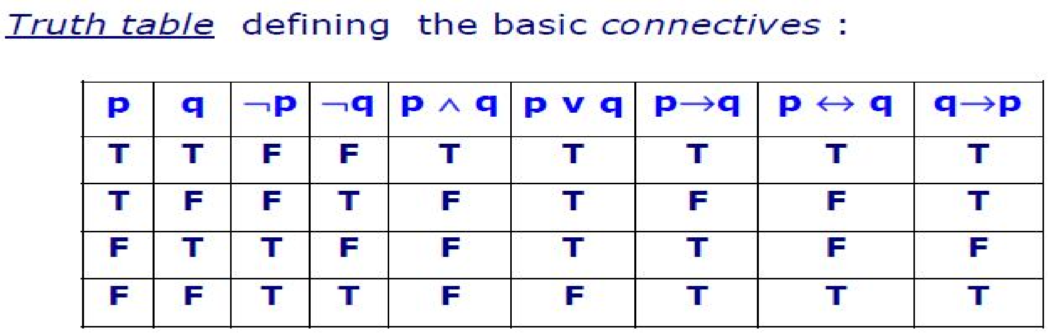


Connectives and Symbols in decreasing order of operation priority

Connective	Symbols					Read as
assertion	P					"p is true"
negation	¬p	∼	!		NOT	"p is false"
conjunction	p ∧ q	·	&&	&	AND	"both p and q are true"
disjunction	P ∨ q	||	|		OR	"either p is true, or q is true, or both "
implication	p → q	⊃	⇒		if ..then	"if p is true, then q is true" " p implies q "
equivalence	↔	≡	⇔		if and only if	"p and q are either both true or both false"

Note : The propositions and connectives are the basic *elements* of propositional logic.

PROPOSITIONAL LOGIC



Truth table defining the basic connectives :

p	q	¬p	¬q	p ∧ q	p v q	p→q	p ↔ q	q→p
T	T	F	F	T	T	T	T	T
T	F	F	T	F	T	F	F	T
F	T	T	F	F	T	T	F	F
F	F	T	T	F	F	T	T	T

- ◊ **Tautologies**
- A proposition that is always true is called a *tautology.* e.g., ***(P v ¬P) is always true regardless of the truth value of the*** proposition ***P.***
- ◊ **Contradictions**
- A proposition that is always false is called a *contradiction.* e.g., ***(P ∧ ¬P) is always false regardless of the truth value of*** the proposition ***P.***
- ◊ **Contingencies**
- A proposition is called a *contingency, if that proposition is neither* a *tautology nor a contradiction* e.g., ***(P v Q) is a contingency.***
- ◊ **Antecedent, Consequent**
- In the conditional statements, p → q , the 1st statement or "if - clause" (here p) is called *antecedent ,* 2nd statement or "then - clause" (here q) is called *consequent.*

PROPOSITIONAL LOGIC

- ◊ **Argument**
- Any argument can be expressed as a compound statement. Take all the premises, conjoin them, and make that conjunction the antecedent of a conditional and make the conclusion the consequent. This implication statement is called the corresponding conditional of the *argument.*
- Note :
- – Every argument has a corresponding conditional, and every implication statement has a corresponding argument.
- – Because the corresponding conditional of an argument is a statement, it is therefore either a tautology, or a contradiction, or a contingency.
- **‡ An argument is *valid "if and only if" its corresponding conditional*** is a *tautology.*
- **‡ Two statements are *consistent "if and only if" their conjunction*** is not a contradiction.
- **‡ Two statements are *logically equivalent "if and only if" their*** truth table columns are identical; "if and only if" the statement of their equivalence using " ≡ " is a tautology.
- Note **: The truth tables are adequate to test *validity, tautology,*** *contradiction, contingency, consistency, and equivalence.*

Predicate Logic

- Propositional logic combines atoms
  - An atom contains no propositional connectives
  - Have no structure (today_is_wet, john_likes_apples)
- **Predicates** allow us to talk about objects
  - Properties: is_wet(today)
  - Relations: likes(john, apples)
  - True or false
- In predicate logic each atom is a predicate
  - e.g. first order logic, higher-order logic

PREDICATE LOGIC

- The ***propositional logic, is not powerful enough for all types of assertions;***
- Example : The assertion *"x > 1", where* ***x is a variable, is not a proposition***
- because it is neither true nor false unless value of ***x is defined.***
- For *x > 1 to be a proposition ,*
- – either we substitute a specific number for ***x ;***
- – or change it to something like *"There is a number x for which x > 1 holds";*
- – or *"For every number x, x > 1 holds".*
- Consider example :
- *“All men are mortal.*
- *Socrates is a man.*
- *Then Socrates is mortal” ,*
- These cannot be expressed in propositional logic as a finite and logically valid argument (formula).
- **We need languages : that allow us to describe properties *(predicates) of*** objects, or a relationship among objects represented by the variables .
- *Predicate logic satisfies the requirements of a language.*
- – *Predicate logic is powerful enough for expression and reasoning.*
- – Predicate logic is built upon the ideas of *propositional logic*

PREDICATE LOGIC

- ■ **Predicate :**
- Every complete *sentence contains two parts: a subject and a predicate.*
- The *subject is what (or whom) the sentence is about.*
- The *predicate tells something about the subject;*
- Example :
- A *sentence "Judy {runs}".*
- The subject is *Judy and the predicate is runs .*
- Predicate, always includes verb, tells something about the subject.
- *Predicate is a verb phrase template that describes a property of objects, or a relation among objects represented by the variables.*
- Example:
- "The car Tom is driving *is blue" ;*
- "The sky *is blue" ;*
- "The cover of this book *is blue"*
- Predicate is *"is blue" , describes property.*
- Predicates are given names; Let *'B' is name for predicate "is_blue".*
- Sentence is represented as *"B(x)" , read as "x is blue";*
- *"x" represents an arbitrary Object .*

Predicate Logic

- ■ **Predicate logic expressions :**
- The propositional operators combine predicates, like
- *If (p(....) && (!q(....) | | r (....)))*
- Examples of logic operators : disjunction (OR) and conjunction (AND).
- Consider the expression with the respective logic symbols | | and *&&*
- *x < y | | (y < z && z < x)*
- Which is *true | | (true && true) ;*
- Applying truth table, found *True*
- Assignment for < *are 3, 2, 1 for x, y, z and then*
- the value can be *FALSE or TRUE*
- *3 < 2 | | (2 < 1 && 1 < 3)*
- It is *False*

First Order Logic

- More expressive logic than propositional
- **Constants** are objects: john, apples
- **Predicates** are properties and relations:
  - likes(john, apples)
- **Functions** transform objects:
  - likes(john, fruit_of(apple_tree))
- **Variables** represent any object: likes(X, apples)
- **Quantifiers** qualify values of variables
  - True for all objects (Universal): $\forall X.$ likes(X, apples)
  - Exists at least one object (Existential): $\exists X.$ likes(X, apples)

EXAMPLE: FOL SENTENCE

- “Every rose has a thorn”

$$\forall X.(rose(X) \rightarrow \exists Y.(has(X,Y) \wedge thorn(Y)))$$

- For all X
  - if (X is a rose)
  - then there exists Y
    - (X has Y) and (Y is a thorn)

Example: FOL Sentence

- "On Mondays and Wednesdays I go to John's house for dinner"

$$\forall X.((is\_mon(X) \vee is\_wed(X)) \rightarrow eat\_meal(me, houseOf(john), X))$$

- Note the change from "and" to "or"
  - Translating is problematic

Beyond True and False

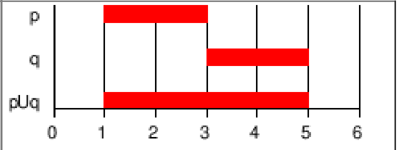
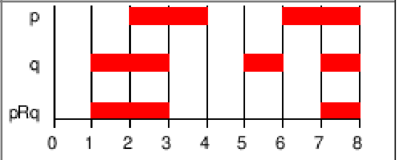
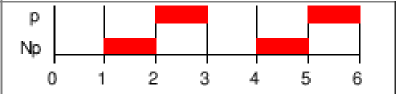
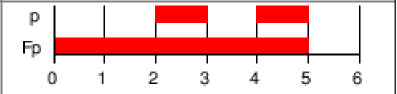
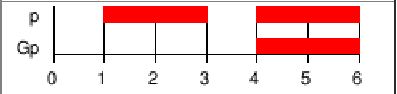
- Multi-valued logics
  - More than two truth values
  - e.g., true, false & unknown
  - **Fuzzy logic** uses probabilities, truth value in [0,1]
- Modal logics
  - Modal operators define mode for propositions
  - **Epistemic logics** (belief)
    - e.g. □p (necessarily p), ◊p (possibly p), …
  - **Temporal logics** (time)
    - e.g. □p (always p), ◊p (eventually p), …

Temporal Logic

- used to describe any system of rules and symbolism for representing, and reasoning about, propositions qualified in terms of time.
- We can then express statements like "I am *always* hungry", "I will *eventually* be hungry", or "I will be hungry *until* I eat something".
- Temporal logic has found an important application in formal verification, where it is used to state requirements of hardware or software systems. For instance, one may wish to say that *whenever* a request is made, access to a resource is *eventually* granted, but it is *never* granted to two requestors simultaneously. Such a statement can conveniently be expressed in a temporal logic.

Temporal Logic

- Temporal operators
- Temporal logic has two kinds of operators: logical operators and modal operators. Logical operators are usual truth-functional operators. The modal operators used in Linear Temporal Logic and Computation Tree Logic are defined as follows.

Textual	Symbolic	Definition	Explanation	Diagram
			Binary operators	
φ U ψ	$\phi\,\mathcal{U}\,\psi$	$(B\,\mathcal{U}\,C)(\phi) = (\exists i : C(\phi_i) \land (\forall j < i : B(\phi_j)))$	**Until:** ψ holds at the current or a future position, and φ has to hold until that position. At that position φ does not have to hold any more.	
φ R ψ	$\phi\,\mathcal{R}\,\psi$	$(B\,\mathcal{R}\,C)(\phi) = (\forall i : C(\phi_i) \lor (\exists j < i : B(\phi_j)))$	**Release:** φ releases ψ if ψ is true until the first position in which φ is true (or forever if such a position does not exist).	
			Unary operators	
N φ	$\bigcirc\phi$	$\mathcal{N}B(\phi_i) = B(\phi_{i+1})$	**Next:** φ has to hold at the next state. (**X** is used synonymously.)	
F φ	$\Diamond\phi$	$\mathcal{F}B(\phi) = (true\,\mathcal{U}\,B)(\phi)$	**Future:** φ eventually has to hold (somewhere on the subsequent path).	
G φ	$\Box\phi$	$\mathcal{G}B(\phi) = \neg\mathcal{F}\neg B(\phi)$	**Globally:** φ has to hold on the entire subsequent path.	
A φ	$\forall\phi$	$(\mathcal{A}B)(\psi) = (\forall\phi : \phi_0 = \psi \to B(\phi))$	**All:** φ has to hold on all paths starting from the current state.	
E φ	$\exists\phi$	$(\mathcal{E}B)(\psi) = (\exists\phi : \phi_0 = \psi \land B(\phi))$	**Exists:** there exists at least one path starting from the current state where φ holds.	

Modal Logic

- a type of formal logic that extends the standards of formal logic to include the elements of modality (for example, possibility and necessity).
- Modals qualify the truth of a judgment. For example, if it is true that "John is happy," we might qualify this statement by saying that "John is *usually* happy," in which case the term "usually" would be a modality.
- Traditionally, there are three "modes" or "moods" or "modalities" represented in modal logic, namely, *possibility*, *probability*, and *necessity*.
- A formal modal logic represents modalities using modal operators. For example, "It might rain today" and "It is possible that rain will fall today" both contain the notion of possibility. In a modal logic this is represented as an operator, *Possibly*, attached to the sentence *It will rain today*.
- The basic unary (1-place) modal operators are usually written $\Box$ for *Necessarily* and $\Diamond$ for *Possibly*.
- In a classical modal logic, each can be expressed by the other with negation:

$$\Diamond P \leftrightarrow \neg\Box\neg P;$$
$$\Box P \leftrightarrow \neg\Diamond\neg P.$$

- Thus it is *possible* that it will rain today if and only if it is *not necessary* that it will *not* rain today;
- and it is *necessary* that it will rain today if and only if it is *not possible* that it will *not* rain today.

Epistemic Logic

- Deals with the *certainty* of sentences.
- The ☐ operator is translated as "x knows that…", and the ◊ operator is translated as "For all x knows, it may be true that…"
- The following contrasts may help:
  - A person, Jones, might reasonably say *both*: (1) "No, it is *not* possible that Bigfoot exists; I am quite certain of that"; *and*, (2) "Sure, Bigfoot possibly *could* exist". What Jones means by (1) is that given all the available information, there is no question remaining as to whether Bigfoot exists. This is an epistemic claim. By (2) he makes the *metaphysical* claim that it is *possible for* Bigfoot to exist, *even though he does not* (which is not equivalent to "it is *possible that* Bigfoot exists – for all I know", which contradicts (1)).
- Epistemic possibilities also bear on the actual world in a way that metaphysical possibilities do not.
- Metaphysical possibilities bear on ways the world *might have been,* but epistemic possibilities bear on the way the world *may be* (for all we know).
  - Suppose, for example, that I want to know whether or not to take an umbrella before I leave. If you tell me "it is *possible that* it is raining outside" – in the sense of epistemic possibility – then that would weigh on whether or not I take the umbrella. But if you just tell me that "it is *possible for* it to rain outside" – in the sense of *metaphysical possibility* – then I am no better off for this bit of modal enlightenment.
- Some features of epistemic modal logic are in debate. For example, if *x* knows that *p*, does *x* know that it knows that *p*? That is to say, should $\Box p \rightarrow \Box\Box p$ be an axiom in these systems? While the answer to this question is unclear, there is at least one axiom that *must* be included in epistemic modal logic, because it is minimally true of all modal logics

Descriptive Logic

- More expressive than propositional logic but has more efficient decision problems than first-order predicate logic.
- DL is used for formal reasoning on the concepts of an application domain (known as *terminological knowledge*).
- It is of particular importance in providing a logical formalism for Ontologies and the Semantic Web.
- Notable application in bioinformatics where DL assists in the codification of medical knowledge.
- models *concepts*, *roles* and *individuals*, and their relationships.
- The fundamental modeling concept of a DL is the *axiom* - a logical statement relating roles and/or concepts. This is a key difference from the frames paradigm where a *frame specification* declares and completely defines a class.

Logic is a Good Representation

- Fairly easy to do the translation when possible
- Branches of mathematics devoted to it
- It enables us to do logical reasoning
  - Tools and techniques come for free
- Basis for programming languages
  - Prolog uses logic programs (a subset of FOL)
  - λProlog based on HOL

NON-LOGICAL REPRESENTATIONS?

- Production rules
- Semantic networks
  - Conceptual graphs
  - Frames
- Logic representations have restricitions and can be hard to work with
  - Many AI researchers searched for better representations

Production Rules

- Rule set of <condition,action> pairs
  - “if condition then action”
- Match-resolve-act cycle
  - **Match**: Agent checks if each rule’s condition holds
  - **Resolve**:
    - Multiple production rules may fire at once (**conflict set**)
    - Agent must choose rule from set (**conflict resolution**)
  - **Act**: If so, rule “fires” and the action is carried out
- Working memory:
  - rule can write knowledge to working memory
  - knowledge may match and fire other rules

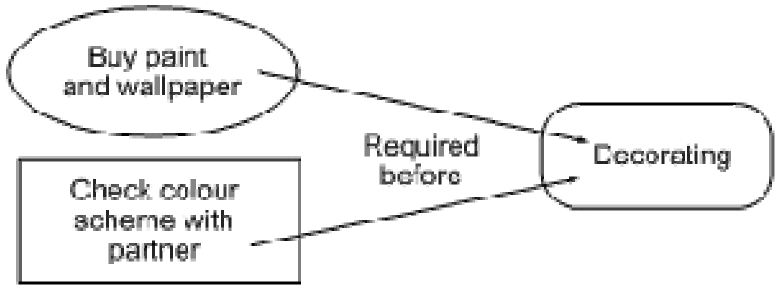
Production Rules Example

- IF (at bus stop AND bus arrives) THEN action(get on the bus)
- IF (on bus AND not paid AND have oyster card) THEN action(pay with oyster) AND add(paid)
- IF (on bus AND paid AND empty seat) THEN sit down

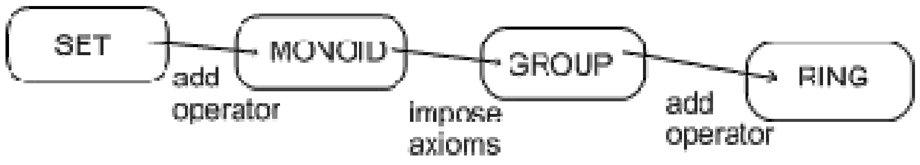
- conditions and actions must be clearly defined
  - can easily be expressed in first order logic!

GRAPHICAL REPRESENTATION

- Humans draw diagrams all the time, e.g.
  - Causal relationships

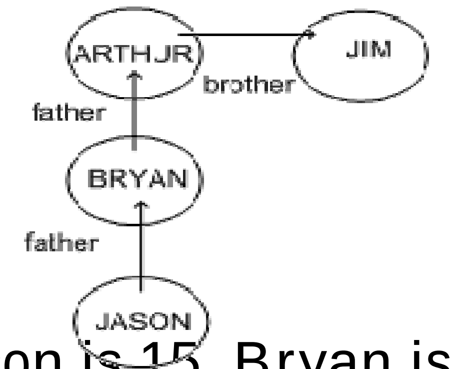


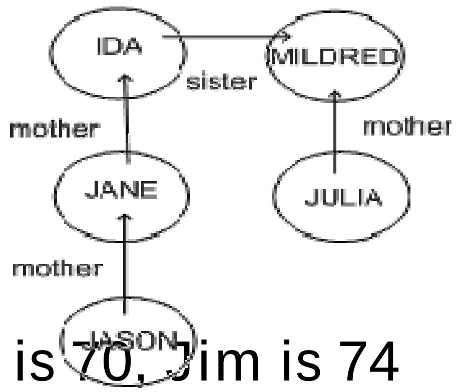
  - And relationships between ideas



Graphical Representation

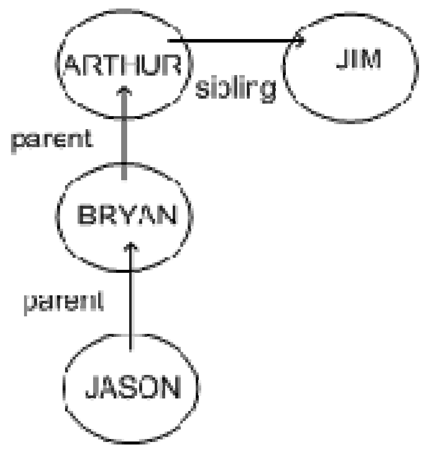
- Graphs easy to store in a computer
- To be of any use must impose a formalism

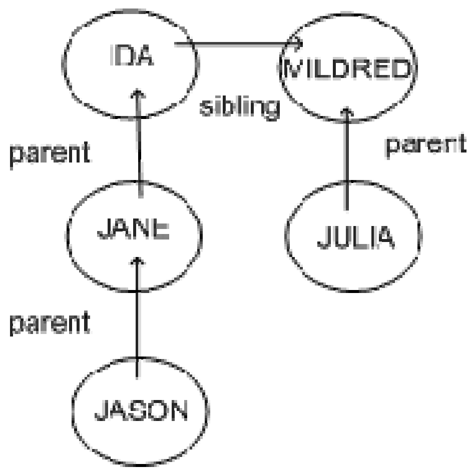




  - Jason is 15, Bryan is 40, Arthur is 70, Jim is 74
  - How old is Julia?

Semantic Networks





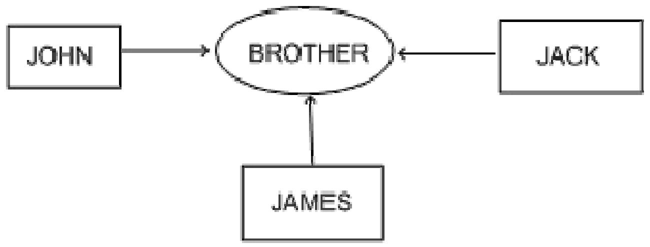
- Because the syntax is the same
  - We can guess that Julia's age is similar to Bryan's
- Formalism imposes restricted syntax

SEMANTIC NETWORKS

- Graphical representation (a graph)
  - Links indicate subset, member, relation, ...
- Equivalent to logical statements (usually FOL)
  - Easier to understand than FOL?
  - Specialised SN reasoning algorithms can be faster
- Example: natural language understanding
  - Sentences with same meaning have same graphs
  - e.g. Conceptual Dependency Theory (Schank)

Conceptual Graphs

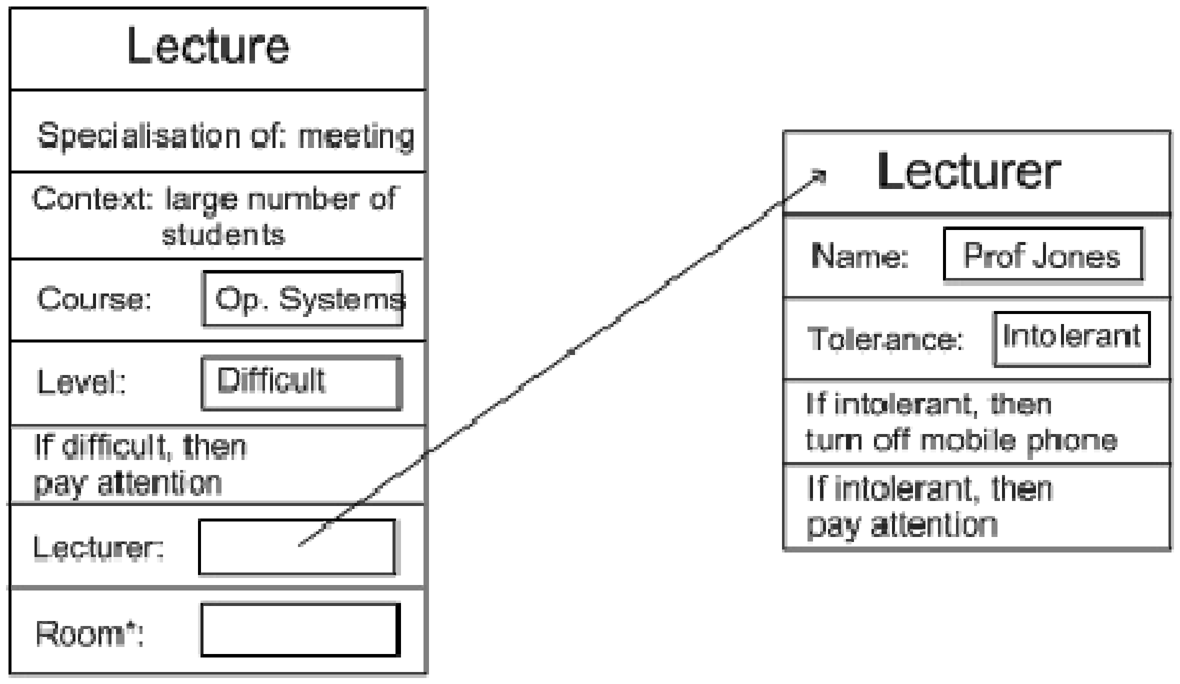
- Semantic network where each graph represents a single proposition
- Concept nodes can be
  - Concrete (visualisable) such as restaurant, my dog Spot
  - Abstract (not easily visualisable) such as anger
- Edges do not have labels
  - Instead, conceptual relation nodes
  - Easy to represent relations between multiple objects



Frame Representations

- Semantic networks where nodes have structure
  - Frame with a number of slots (age, height, ...)
  - Each slot stores specific item of information
- When agent faces a new situation
  - Slots can be filled in (value may be another frame)
  - Filling in may trigger actions
  - May trigger retrieval of other frames
- Inheritance of properties between frames
  - Very similar to objects in OOP

Example: Frame Representation



Lecture
Specialisation of: meeting
Context: large number of students
Course:
Op. Systems
Level:
Difficult
If difficult, then pay attention
Lecturer:
Room*:
Lecturer
Name:
Prof Jones
Tolerance:
Intolerant
If intolerant, then turn off mobile phone
If intolerant, then pay attention

Flexibility in Frames

- Slots in a frame can contain
  - Information for choosing a frame in a situation
  - Relationships between this and other frames
  - Procedures to carry out after various slots filled
  - Default information to use where input is missing
  - Blank slots: left blank unless required for a task
  - Other frames, which gives a hierarchy
- Can also be expressed in first order logic

Representation & Logic

- AI wanted “non-logical representations”
  - Production rules
  - Semantic networks
    - Conceptual graphs, frames
- But all can be expressed in first order logic!
- Best of both worlds
  - Logical reading ensures representation well-defined
  - Representations specialised for applications
  - Can make reasoning easier, more intuitive

Logic Programming

- Logic programming offers a formalism for specifying a computation in terms of logical relations between entities.
- – logic program is a collection of logic statements.
- – programmer describes all relevant logical relationships between the various entities.
- – computation determines whether or not, a particular conclusion follows from those logical statements.
- **• characteristics of Logic program**
- Logic program is characterized by set of relations and inferences.
- – the *program consists of a set of axioms and a goal statement.*
- – the *Rules of inference determine whether the axioms are sufficient to* ensure the truth of the goal statement.
- – the *execution of a logic program corresponds to the construction of a* proof of the goal statement from the axioms.
- – the *Programmer specify basic logical relationships, does not specify the m*anner in which inference rules are applied.
- Thus Logic + Control = Algorithms
- **• Examples of Logic Statements**
- – Statement
- A grand-parent is a parent of a parent.
- – Statement expressed in more closely related logic terms as
  - A person is a grand-parent if she/he has a child and that child is a parent.
- – Statement expressed in first order logic as
  - (for all) x: grand-parent(x) ← (there exist) y, z : parent(x, y) & parent(y, z)

Logic Programming

- **Logic programming Language**
- A programming language includes :
- – the syntax
- – the semantics of programs and
- – the computational model.
- There are many ways of organizing computations.
- The most familiar paradigm is procedural. The program specifies a computation by saying *"how" it is to be performed. FORTRAN, C, and* object-oriented languages fall under this general approach.
- Another paradigm is declarative. The program specifies a computation by giving the properties of a correct answer. Prolog and logic data language (LDL) are examples of declarative languages, emphasize the logical properties of a computation.
- **Prolog and LDL are called logic programming languages.**
- **PROLOG is the most popular Logic programming system.**

Logic Programming

- **• Syntax and terminology (relevant to Prolog programs)**
- In any language, the formation of components (expressions, statements, etc.), is guided by syntactic rules. The components are divided into two parts: (A) data components and (B) program components.
- **(A) Data components :**
- Data components are collection of data objects that follow hierarchy.
- **Data object of any kind is also called** a *term. A term is a constant, a variable* or a compound term.
- **Simple data object is not** decomposable; e.g. atoms, numbers, constants, variables. The syntax distinguishes the data objects, hence
- no need for declaring them.
- **Structured data object are made of** several components; e.g. general, special structure.

Logic Programming

- **(a) Data objects : The data objects of any kind is called a *term.***
- **◊ Term : examples**
- **‡ Constants: denote elements such as *integers, floating point, atoms.***
- **‡ Variables: denote a single but unspecified element; *symbols for*** variables begin with an uppercase letter or an underscore.
- **‡ Compound terms: comprise a *functor and sequence of one or more*** compound terms called *arguments.*
  - **Functor : is characterized by its name, which is an *atom, and its*** *arity or number of arguments.*
    - *f / n = f(t1 , t2, . . . tn)*
    - where *f is name of the functor and is of arity n*
    - *ti 's are the arguments*
    - *f / n denotes functor f of arity n*
  - Functors with the same name but different arities are distinct.
- **‡ Ground and non-ground: *Terms are ground if they contain no*** variables; otherwise they are non-ground. *Goals are atoms or* compound terms, and are generally non-ground.

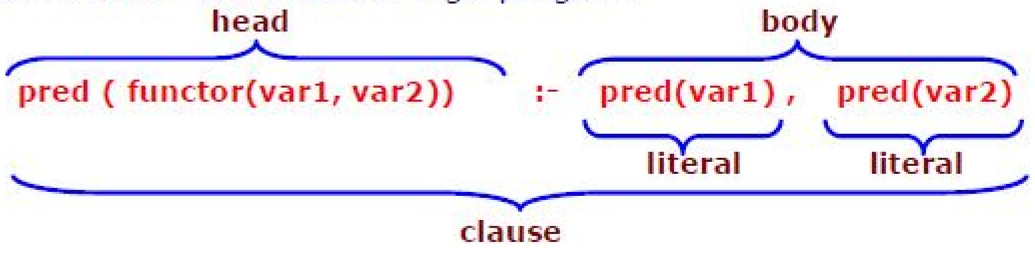
- **(b) Simple data objects : *Atoms, Numbers, Variables***
- **◊ Atoms**
  - ***a lower-case letter, possibly followed by other letters (either case),*** *digits, and underscore character.*
    - e.g. a   greaterThan   two_B_or_not_2_b
  - ***a string of special characters such as: + - * / \ = ^ < > : . ~ @ # $ &***
    - e.g. <> ##&& ::=
  - ***a string of any characters enclosed within single quotes.***
    - e.g. 'ABC' '1234' 'a<>b'
  - **following are also *atoms ! ; [] {}***
- **◊ Numbers**
  - ‡ applications involving heavy numerical calculations are rarely written in Prolog.
  - ‡ *integer representation: e.g. 0 -16 33 +100*
  - ‡ *real numbers written in standard or scientific notation,*
    - e.g. 0.5 -3.1416 6.23e+23 11.0e-3 -2.6e-2
- **◊ Variables**
  - ‡ begins by a capital letter, possibly followed by other letters (either case), digits, and underscore character.
    - e.g. X25 List Noun_Phrase

- **(c) Structured data objects :** ***General Structures , Special Structures***
- **◊ General Structures**
  - **‡ a structured term is syntactically formed by a *functor and a list of*** *arguments.*
  - **‡ *functor is an atom.***
  - **‡ *list of arguments appears between parentheses.***
  - **‡ *arguments are separated by a comma.***
  - **‡ *each argument is a term (i.e., any Prolog data object).***
  - **‡ *the number of arguments of a structured term is called its arity.***
  - **‡ e.g. greaterThan(9, 6) f(a, g(b, c), h(d)) plus(2, 3, 5)**
- Note : a structure in Prolog is a mechanism for combining terms together,
- like integers 2, 3, 5 are combined with the functor *plus.*
- **◊ Special Structures**
- **‡ In Prolog an *ordered collection of terms is called a list .***
- **‡ *Lists are structured terms and Prolog offers a convenient notation to***
- represent them:
- * *Empty list is denoted by the atom [ ].*
- * *Non-empty list carries element(s) between square brackets,* separating elements by comma.
- e.g. [bach, bee] [apples, oranges, grapes] []

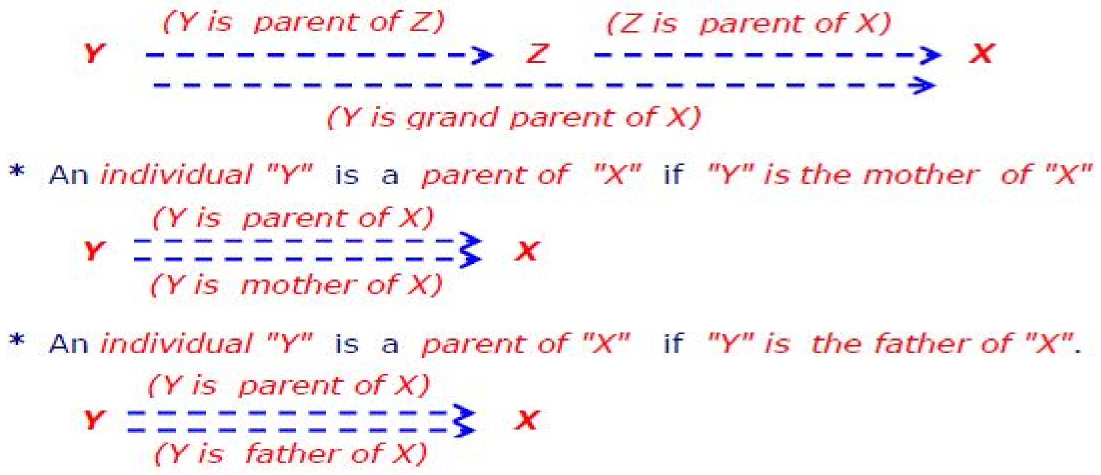
- **(B) Program Components**
- A *Prolog program is a collection of predicates or rules. A predicate* establishes a relationships between objects.
- **(a) Clause, Predicate, Sentence, Subject**
  - ‡ ***Clause is a collection of grammatically-related words .***
  - ‡ ***Predicate is composed of one or more clauses.***
  - ‡ ***Clauses are the building blocks of sentences; every sentence contains*** one or more clauses.
  - ‡ **A Complete Sentence has two parts: *subject and predicate.***
    - o subject is what (or whom) the sentence is about.
    - o predicate tells something about the subject.
  - ‡ **Example 1 : "cows eat grass".**
    - It is a clause, because it contains the subject *"cows" and* the predicate *"eat grass."*
  - ‡ **Example 2 : "cows eating grass are visible from highway"**
    - This is a complete clause. The subject *"cows eating grass" and* the predicate *"are visible from the highway" makes complete thought.*

- **(b) Predicates & Clause**
- Syntactically a predicate is composed of one or more clauses.
- **‡ The general form of clauses is :**
- <left-hand-side> :- <right-hand-side>.
- where LHS is a single goal called *"goal" and*
- RHS is composed of one or more goals, separated by commas, called *"sub-goals" of the goal on left-hand side.*

‡ The structure of a clause in logic program



- **‡ Example : *grand_parent (X, Y) :- parent(X, Z), parent(Z, Y).***
- *parent (X, Y) :- mother(X, Y).*
- *parent (X, Y) :- father(X, Y).*
- **‡ Interpretation:**
- *** a clause specifies the conditional truth of the goal on the LHS;**
- i.e., goal on LHS is assumed to be true if the sub-goals on RHS are all true. A predicate is true if at least one of its clauses is true.
- **** An individual "Y" is the grand-parent of "X" if a parent of that same** "X" is "Z" and "Y" is the parent of that "Z"**.***



- **(c) Unit Clause - a special Case**
- Unlike the previous example of conditional truth, one often encounters unconditional relationships that hold.
- **‡ In Prolog the clauses that are unconditionally true are called *unit*** *clause or fact*
- **‡ Example : Unconditionally relationships** say *'Y' is the father of 'X' is unconditionally true.*
- This relationship as a Prolog clause is :
- *father(X, Y) :- true.*
- Interpreted as relationship of father between *Y and X is always true;*
- or simply stated as *Y is father of X*
- **‡ Goal true is built-in in Prolog and always holds.**
- **‡ Prolog offers a simpler syntax to express unit clause or fact**
- *father(X, Y)*
- ie the *:- true part is simply omitted.*

- **(d) Queries**
- In Prolog the queries are statements called directive. A special case of
- directives, are called *queries.*
- **‡ Syntactically, directives are clauses with an empty left-hand side.**
- Example : ? - grandparent(X, W).
- This query is interpreted as : *Who is a grandparent of X ?*
- By issuing queries, Prolog tries to establish the validity of specific relationships.
- **‡ The result of executing a query is either *success or failure***
- **Success, means the goals specified in the query holds according to the** facts and rules of the program.
- **Failure, means the goals specified in the query does not hold according** to the facts and rules of the program.

PROGRAMMING PARADIGMS : MODELS OF COMPUTATION

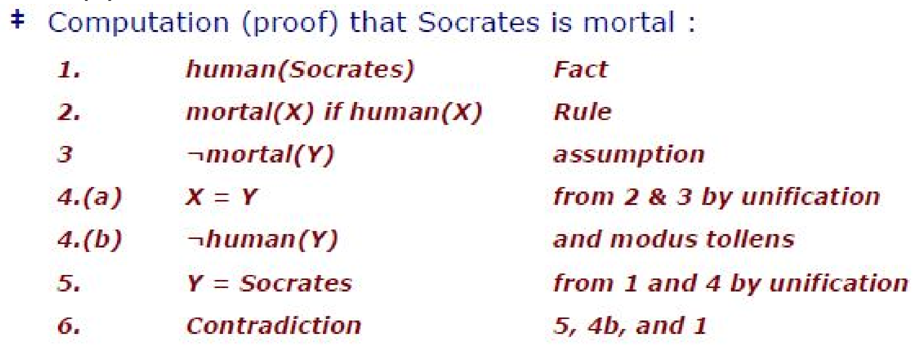
- A complete description of a programming language includes the *computational model, syntax, semantics, and pragmatic onsiderations that* shape the language.
- **Models of Computation :**
- *A computational model is a collection of values and operations, while computation is the application of a sequence of operations to a value* to yield another value. There are three basic computational models :
- *(a) Imperative, (b) Functional, and (c) Logic. In addition to these,* there are two programming paradigms (concurrent and object-oriented programming). While, they are not models of computation, they rank in importance with computational models.

- **(a) Imperative Model :**
- The Imperative model of computation, consists of a state and an operation of assignment which is used to modify the state. Programs consist of sequences of commands. The computations are changes in the state.
- Example 1 **: Linear function**
- A linear function $y = 2x + 3$ *can be written as*
  - $Y := 2 * X + 3$
- The implementation determines the value of X *in the state and then* create a new state, which differs from the old state. The value of Y *in* the new state is the value that $2 * X + 3$ *had in the old state.*
- *Old State:* $X = 3,\ Y = -2,$
  - $Y := 2 * X + 3$
- *New State:* $X = 3,\ Y = 9,$
- The imperative model is closest to the hardware model on which programs are executed, that makes it most efficient model in terms of execution time.

- **(b) Functional model :**
- The Functional model of computation, consists of a set of values, functions, and the operation of functions. The functions may be named and may be composed with other functions. They can take other functions as arguments and return results. The programs consist of definitions of functions. The computations are application of functions to values.
- ‡ Example 1 : Linear function
- A linear function y = 2x + 3 can be defined as :
  - $f(x) = 2 * x + 3$
- ‡ Example 2 : Determine a value for Circumference.
- Assigned a value to Radius, that determines a value for Circumference.
- Circumference = 2 × pi × radius where pi = 3.14
- Generalize Circumference with the variable "radius" ie Circumference(radius) = 2 × pi × radius , where pi = 3.14
- Functional models are developed over many years. The notations and methods form the base upon which problem solving methodologies rest.

- **(c) Logic model :**
- The logic model of computation is based on relations and logical inference. Programs consist of definitions of relations. Computations are inferences (is a proof).
- **‡ Example 1 : Linear function**
- A linear function *y = 2x + 3 can be represented as : f (X , Y) if Y is 2 ∗ X + 3.*
- **‡ Example 2: Determine a value for Circumference.**
- The earlier circumference computation can be represented as:
- *Circle (R , C) if Pi = 3.14 and C = 2 ∗ pi ∗ R.*
- The function is represented as a relation between radious *R and* circumference *C.*
- **‡ Example 3: Determine the mortality of Socrates.**
- The program is to determine the mortality of Socrates. The fact given that Socrates is human.
- The rule is that all humans are mortal, that is
- *for all X, if X is human then X is mortal.*
- To determine the mortality of Socrates, make the assumption that there are no mortals, that is ¬ *mortal (Y)*

- ‡ **The fact and rule are:**
  - *human (Socrates)*
  - *mortal (X) if human (X)*
- ‡ **To determine the mortality of Socrates, make the assumption that** there are no mortals i.e. ¬ *mortal (Y)*



‡ Computation (proof) that Socrates is mortal :

1.	*human(Socrates)*	*Fact*
2.	*mortal(X) if human(X)*	*Rule*
3	*¬mortal(Y)*	*assumption*
4.(a)	*X = Y*	*from 2 & 3 by unification*
4.(b)	*¬human(Y)*	*and modus tollens*
5.	*Y = Socrates*	*from 1 and 4 by unification*
6.	*Contradiction*	*5, 4b, and 1*

- ‡ **Explanation :**
- *** The 1st line is the statement *"Socrates is a man."***
- *** The 2nd line is a phrase *"all human are mortal"*** into the equivalent *"for all X, if X is a man then X is mortal".*
- *** The 3rd line is added to the set to determine the mortality of Socrates.**
- *** The 4th line is the deduction from lines 2 and 3. It is justified by the** inference rule *modus tollens which states that if the conclusion of a* rule is known to be false, then so is the hypothesis.
- *** Variables X and Y are unified because they have same value.**
- *** By unification, Lines 5, 4b, and 1 produce contradictions and identify** Socrates as mortal.
- *** Note that, resolution is the an inference rule which looks for a** contradiction and it is facilitated by unification which determines if there is a substitution which makes two terms the same.
- Logic model formalizes the reasoning process. It is related to relational
- data bases and expert systems.