MODULE III KNOWLEDGE AND REASONING 9

Knowledge Representation-Knowledge based Agents-The Wumpus World Logic-**Propositional Logic-Predicate Logic-Unification and Lifting** - Representing Knowledge using Rules-Semantic Networks Frame Systems Inference – Types of Reasoning.

What is a Logic?

Logic consists of

1. A formal system for expressing knowledge about a domain consisting of Syntax Set of legal sentences (well formed formulae) Semantics Interpretation of legal sentences.
2. A proof system — a set of axioms plus rules of inference for deducing sentences from a knowledge base.

Why do we need logic?

- Problem solving agents were very inflexible: hard code for every possible state.
- Search is almost always exponential in the number of states.
- Problem solving agents cannot infer unobserved information.
- We want an algorithm that reasons in a way that resembles reasoning in humans.

Types of Logic:

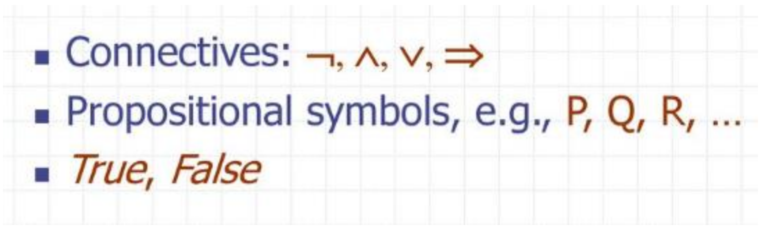
- Propositional logic-deals with specific objects and concrete statements that are either true or false.

  Example: John is married to Sue.
- Predicate logic- allows statement to contain variables, functions, and quantifiers.
- Fuzzy logic-deals with statement that are somewhat vague, such as this paint is grey, or the sky is cloudy.
- Probability-deals with statements that are possibly true, such as whether I will win the lottery next week.
- Temporal logic-deals with statements about time, such as John was a student at UC Irvine for four years.
- Modal logic-deals with statements about belief or knowledge, such as Mary believes that John is married to Sue, or Sue knows that search is NP-complete.

PROPOSITIONAL LOGIC

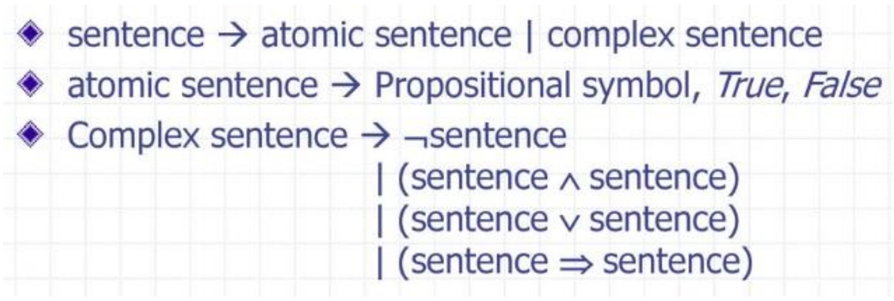
Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.

Sybmols of propositional logic:



- Connectives: ¬, ∧, ∨, ⇒
- Propositional symbols, e.g., P, Q, R, …
- *True, False*

Syntax of propositional logic:

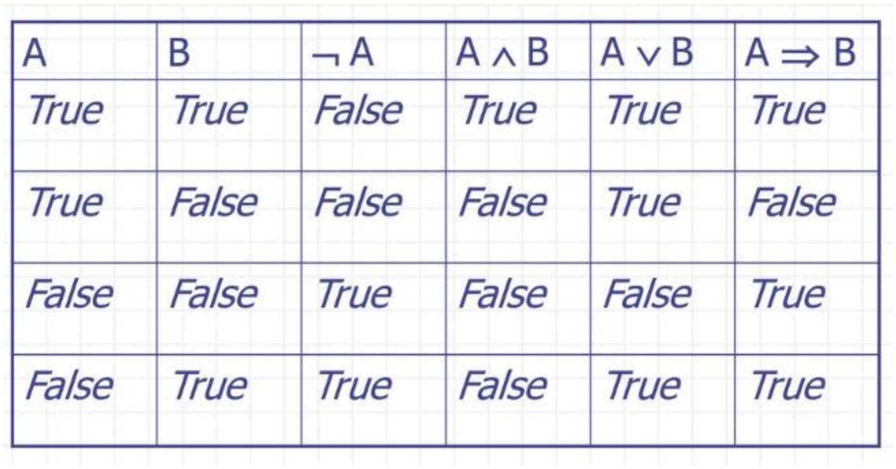


- sentence → atomic sentence | complex sentence
- atomic sentence → Propositional symbol, *True, False*
- Complex sentence → ¬sentence
  | (sentence ∧ sentence)
  | (sentence ∨ sentence)
  | (sentence ⇒ sentence)

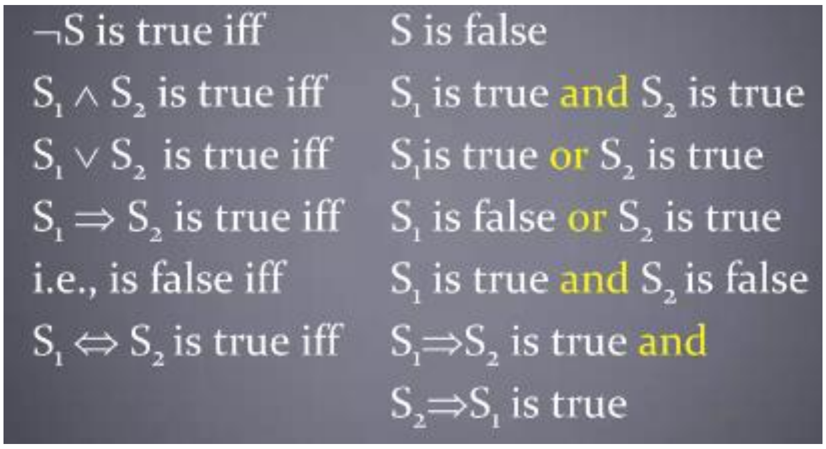
Semantics of propositional logic:

The truth value of every other sentence is obtained recursively by using truth tables.

Truth table for connectives:



A	B	¬ A	A ∧ B	A ∨ B	A ⇒ B
True	*True*	*False*	*True*	*True*	*True*
True	*False*	*False*	*False*	*True*	*False*
False	*False*	*True*	*False*	*False*	*True*
False	*True*	*True*	*False*	*True*	*True*



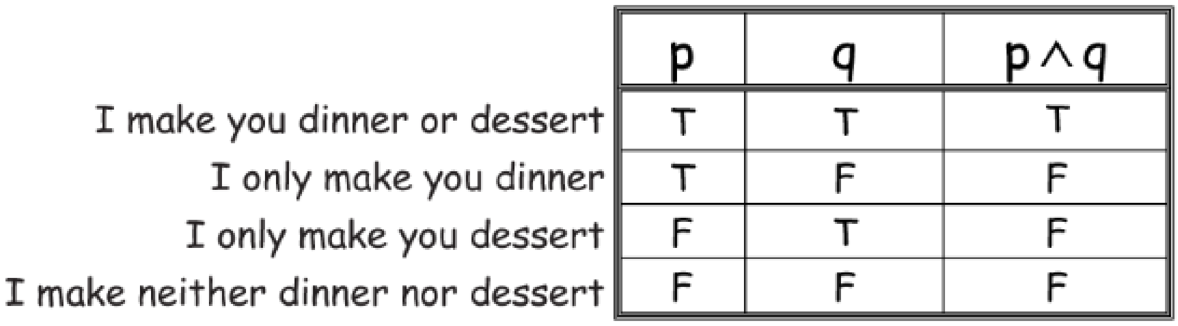
$\neg S$ is true iff	S is false
$S_1 \wedge S_2$ is true iff	S_1 is true and S_2 is true
$S_1 \vee S_2$ is true iff	$S_1$is true or S_2 is true
$S_1 \Rightarrow S_2$ is true iff	S_1 is false or S_2 is true
i.e., is false iff	S_1 is true and S_2 is false
$S_1 \Leftrightarrow S_2$ is true iff	$S_1 \Rightarrow S_2$ is true and $S_2 \Rightarrow S_1$ is true

Example:

Suppose we let:

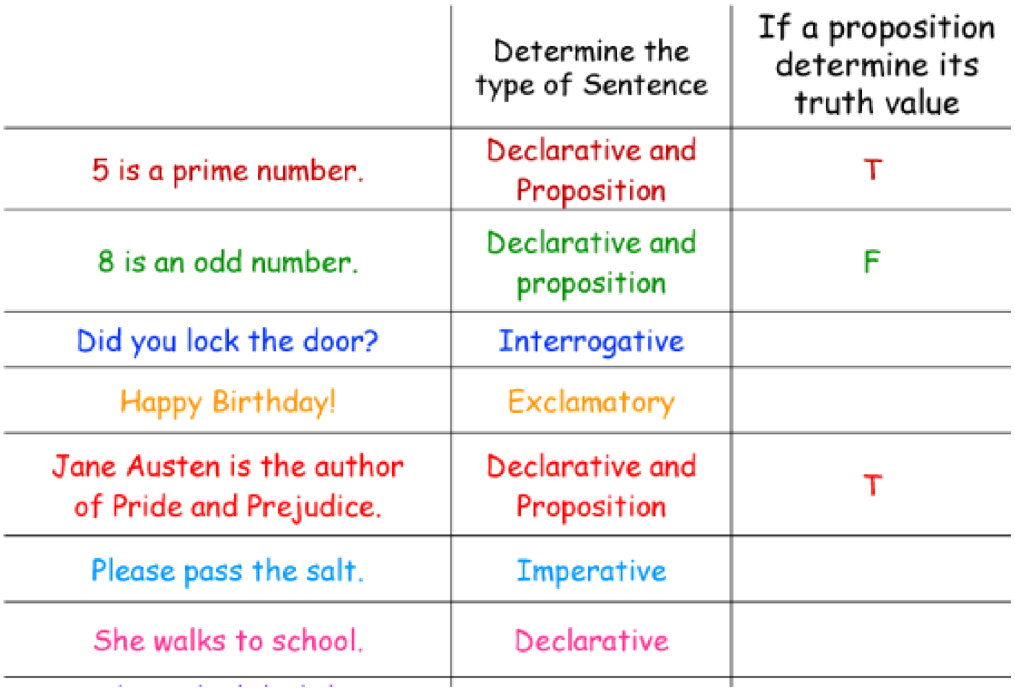
- p represent the statement, "make supper"
- q represent "make dessert"

What is the truth value of "I will make you supper, and I will make your dessert."



	p	q	$p \wedge q$
I make you dinner or dessert	T	T	T
I only make you dinner	T	F	F
I only make you dessert	F	T	F
I make neither dinner nor dessert	F	F	F

Types of sentences



	Determine the type of Sentence	If a proposition determine its truth value
5 is a prime number.	Declarative and Proposition	T
8 is an odd number.	Declarative and proposition	F
Did you lock the door?	Interrogative	
Happy Birthday!	Exclamatory	
Jane Austen is the author of Pride and Prejudice.	Declarative and Proposition	T
Please pass the salt.	Imperative	
She walks to school.	Declarative	

PREDICATE LOGIC OR FIRST ORDER LOGIC

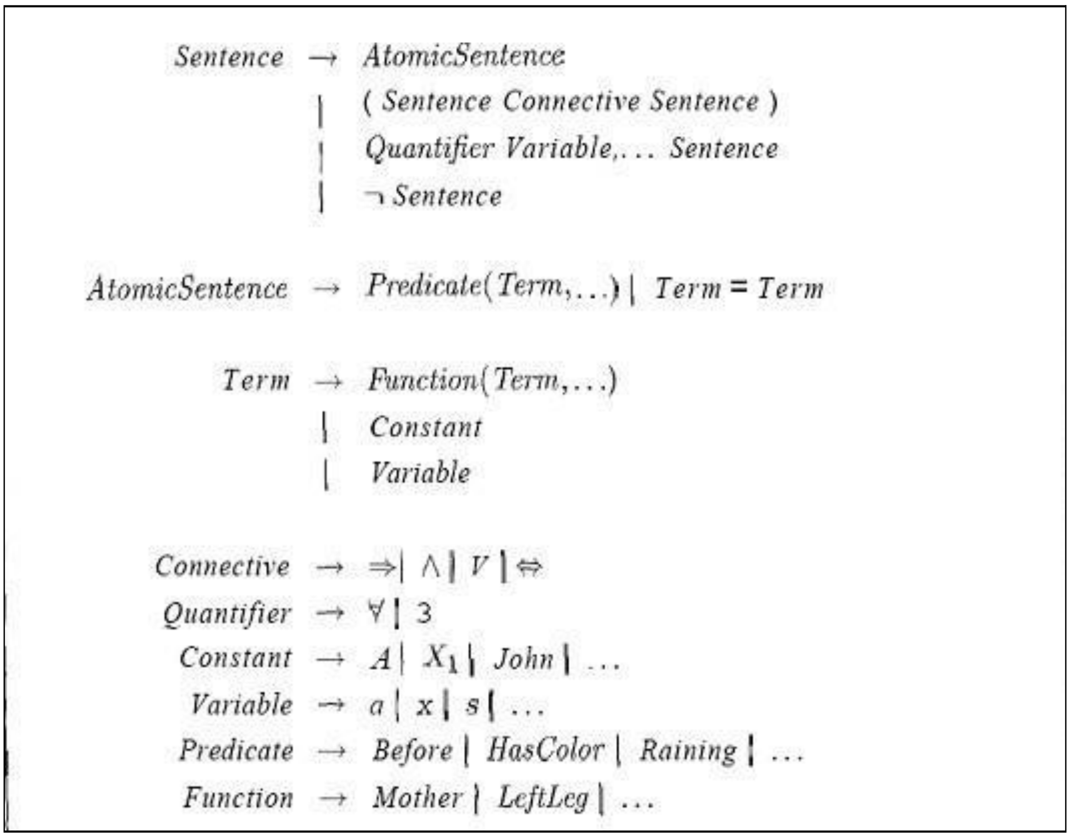
- First order logic is declarative, compositional and more expressive than propositionallogic.
- Meaning in propositional logic is context-independent
  - (unlike natural language, where meaning depends on context)

Propositional logic has very limited expressive power
  - (unlike natural language)
  - E.g., cannot say "pits cause breezes in adjacent squares“

- except by writing one sentence for each square

– Whereas propositional logic assumes the world contains facts,
  - first-order logic (like natural language) assumes the world contains
    - Objects: people, houses, numbers, colors, baseball games, wars, …
    - Relations: red, round, prime, brother of, bigger than, part of, comesbetween,…
    - Functions: father of, best friend, one more than, plus, …

Syntax of FOL: Basic elements



Sentence	$\rightarrow$	*AtomicSentence*
	|	(*Sentence Connective Sentence*)
	|	*Quantifier Variable*,… *Sentence*
	|	$\neg$ *Sentence*
AtomicSentence	$\rightarrow$	*Predicate*(*Term*,…) | *Term* = *Term*
Term	$\rightarrow$	*Function*(*Term*,…)
	|	*Constant*
	|	*Variable*
Connective	$\rightarrow$	$\Rightarrow$ | $\wedge$ | $\vee$ | $\Leftrightarrow$
Quantifier	$\rightarrow$	$\forall$ | $\exists$
Constant	$\rightarrow$	A | X_1 | *John* | …
Variable	$\rightarrow$	a | x | s | …
Predicate	$\rightarrow$	*Before* | *HasColor* | *Raining* | …
Function	$\rightarrow$	*Mother* | *LeftLeg* | …

Components of First-Order Logic

- **Term**
  - Constant, e.g. Red
  - Function of constant, e.g. Color(Block1)
- **Atomic Sentence**
  - Predicate relating objects (no variable)
    - Brother (John, Richard)
    - Married (Mother(John), Father(John))
- **Complex Sentences**
  - Atomic sentences + logical connectives
    - Brother (John, Richard) $\wedge\neg$Brother (John, Father(John))
- **Quantifiers**
  - Each quantifier defines a variable for the duration of the following expression, andindicates the truth of the expression…

Quantifiers:

i.. Universal Quantifier:

Syntax:

$\forall$***<variables<sentence>***

Example:

Everyone at NUS is smart:

$\forall$x At(x,NUS) $\Rightarrow$ Smart(x)

- **$\forall$x *P* is true in a model *m* iff *P* is true with *x* being each possible object in the model**
- **Roughly speaking, equivalent to the conjunction of instantiations of *P***

Existential Quantifier:

- $\exists$***<variables> <sentence>***
- **Someone at NUS is smart:**
- **$\exists$*x* At(x,NUS) $\wedge$ Smart(x)$**
- **$\exists$*x* *P* is true in a model *m* iff *P* is true with *x* beingsome possible object in the model**
- **Roughly speaking, equivalent to the disjunction ofinstantiations of *P***

Connections between $\forall$ and $\exists$

The two quantifiers are actually intimately connected with each other, through negation.“Everyone likes icecream “ is equivalent “there is no one who does not like ice cream”

This can be expressed as :

$\forall$x Likes(x,IceCream) is equivalent to $\exists$ Likes(x,IceCream)

Properties of quantifiers

- $\forall x \neg P \equiv \neg \exists x P$
- $\neg\forall x P \equiv \exists x \neg P$
- $\forall x P \equiv \neg\exists x \neg P$
- $\exists x P \equiv \neg\forall x \neg P$
- $\forall x \forall y$ is the same as $\forall y \forall x$
- $\exists x \exists y$ is the same as $\exists y \exists x$
- $\exists x \forall y$ is not the same as $\forall y \exists x$

  $\exists x \forall y$ Loves(x,y)

  - “There is a person who loves everyone in the world”

  $\forall y \exists x$ Loves(x,y)

  - “Everyone in the world is loved by at least one person”
- **Quantifier duality**: each can be expressed using the other

  $\forall$x Likes(x,IceCream)$\neg\exists$x $\neg$Likes(x,IceCream)

∃x Likes(x,Broccoli) ¬∀x ¬Likes(x,Broccoli)

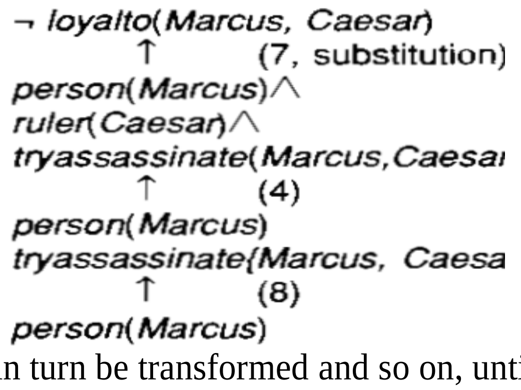
Representing Simple facts in Logic

1. Marcus was a man.
   man(Marcus)
2. Marcus was a Pompeian.
   Pompeian(Marcus)
3. All Pompeians were Romans.
   ∀x: Pompeian(x) → Roman(x)
4. Caesar was a ruler.
   ruler(Caesar)
5. All Romans were either loyal to Caesar or hated him.
   ∀x: Roman(x) → loyalto(x, Caesar) ∨ hate(x, Caesar)
6. Every one is loyal to someone.
   ∀x: ∃*y* : loyalto(x, y)
7. People only try to assassinate rulers they are not loyal to.
   ∀x: ∀y: person(x) ∧ ruler(y) ∧ tryassassinate(x, y) → ¬ loyalto(x, y)
8. Marcus tried to assassinate Caesar.
   tryassassinate(Marcus, Caesar)

To find an answer for the question Was Marcus loyal to Caesar?

To produce a formal proof , reasoning backward from the desired goal

To prove the goal, rules of inference are to be used to transform into



¬ loyalto(Marcus, Caesar)
↑ (7, substitution)
person(Marcus)∧
ruler(Caesar)∧
tryassassinate(Marcus,Caesa
↑ (4)
person(Marcus)
tryassassinate(Marcus, Caesa
↑ (8)
person(Marcus)

another goal that in turn be transformed and so on, until there are no in satisfied goals remaining.

- This attempt fails , since there is no way to satisfy the goal person(Marcus)with the statementsavailable.
- The problem is although its known that Marcus was a man there is no way to conclude it.
- Therefore another representation is added namely

9. **All men are people**
   ∀x: **man(x)→person(x)**

- **This satisfies the last goal and produce a proof that Marcus was not loyal to ceasar.**

Examples

- Marcus was a man
  Man(Marcus).
- Marcus was a Pompeian
  Pompeian(Marcus).
- Marcus was born in 40 A.D
  Born(Marcus,40)
- All men are mortal
  $\forall x \;\; Man(x) \Rightarrow Mortal(x)$
- All Pompeians died when the volcano erupted in 79 A.D.
  $Erupted(Volcano, 79) \wedge \forall x \;\; Pompeian(x)$
  $\rightarrow Died(x, 79)$
- No mortal lives longer than 150 years.
  $\forall x \; \forall t1 \; \forall t2 \;\; Mortal(x) \wedge Born(x, t1) \wedge$
  $GT(t2 - t1, 150) \rightarrow Dead(x, t2)$
- It is now 1985
  Now = 1985.
- Alive means not dead
  $\forall x \; \forall t \;\; Alive(x, t) \leftrightarrow \sim Dead(x, t)$
- If someone dies, then he is dead at all later times.
  $\forall x \; \forall t1 \; \forall t2 \;\; Died(x, t1) \wedge GT(t2, t1) \rightarrow Dead(x, t2)$

1. **Some dogs bark.**
   ∃x dog(x) ∧ bark(x)
2. **All dogs have four legs.**
   ∀x(dog(x)→have_four_l
   egs(x))(or)
   ∀x(dog(x)→legs(x,4))
3. **All barking dogs are irritating**
   ∀x(dog(x) ∧barking(x)→irritating(x))
4. **No dogs purr.**
   ¬∃x (dog(x) ∧ purr(x))

5. **Fathers are male parents with children.**

$\forall x(father(x) \rightarrow male(x) \land haschildren(x))$

6. **Students are people who are enrolled in courses.**

$\forall x(student(x) \rightarrow enrolled(x, courses))$

7. **Some students took French in spring 2001.**

$\exists x\ Student(x) \land Takes(x, F, Spring2001).$

8. **Every student who takes French passes it.**

$\forall\ x, s\ Student(x) \land Takes(x, F, s) \Rightarrow Passes(x, F, s).$

9. **Only one student took Greek in spring 2001.**

$\exists x\ Student(x) Takes(x,G, Spring2001) \land \forall y\ y \neq x \Rightarrow \neg\ Takes(y,G, Spring2001).$

10. **The best score in Greek is always higher than the best score in French.**

$\forall\ s\ \exists\ x\ \forall y\ Score(x,G, s) > Score(y,F, s).$

11. **Every person who buys a policy is smart.**

$\forall x\ Person(x) \land (\exists\ y, z\ Policy(y) \land Buys(x, y, z)) \Rightarrow Smart(x).$

12. **No person buys an expensive policy.**

$\forall\ x, y, z\ Person(x) \land Policy(y) \land Expensive(y) \Rightarrow\ \neg\ Buys(x, y, z).$

13. **There is an agent who sells policies only to people who are not insured.**

$\exists x\ Agent(x) \land \forall y, z\ Policy(y) \land Sells(x, y, z) \Rightarrow (Person(z) \land \neg\ Insured(z)).$

14. **There is a barber who shaves all men in town who do not shave themselves.**

$\exists x\ Barber(x) \land \forall y\ Man(y) \land\ \neg\ Shaves(y, y) \Rightarrow Shaves(x, y).$

15. **A person born in the UK, each of whose parents is a UK citizen or a UK resident, is a UK citizen by birth.**

$\forall x\ Person(x) \land Born(x,UK) \land (\forall y\ Parent(y, x) \Rightarrow ((\exists\ r\ Citizen(y,UK, r)) \lor Resident(y,UK))) \Rightarrow Citizen(x,UK,Birth).$

16. **A person born outside the UK, one of whose parents is a UK citizen by birth, is a UK citizenby descent.**

$\forall x\ Person(x)\ \neg\ Born(x,UK) \land (\exists y\ Parent(y, x) \land Citizen(y,UK,Birth)) \Rightarrow Citizen(x,UK,Descent).$

17. **Politicians can fool some of the people all of the time, and they can fool all of the people some of the time, but they can't fool all of the people all of the time.**

$\forall\ x\ Politician(x) \Rightarrow (\exists\ y\ \forall t\ Person(y) \land Fools(x, y, t)) \land (\exists\ t\ \forall y\ Person(y) \Rightarrow Fools(x, y, t)) \land \neg$
$(\forall\ t\ \forall y\ Person(y) \Rightarrow Fools(x, y, t))$

18. **If a perfect square is divisible by a prime p then it is also divisible by square of p.**

$\forall xy\ perfect\_sq(x) \land prime(y) \land divides(x,y) \rightarrow divides(x,square(y))$

19. **Every perfect square is divisible by some prime.**

$\forall x \exists y\ perfect\_sq(x) \land prime(y) \land divides(x,y)$

20. **36 is a perfect square.**

$perfect\_sq(36)$

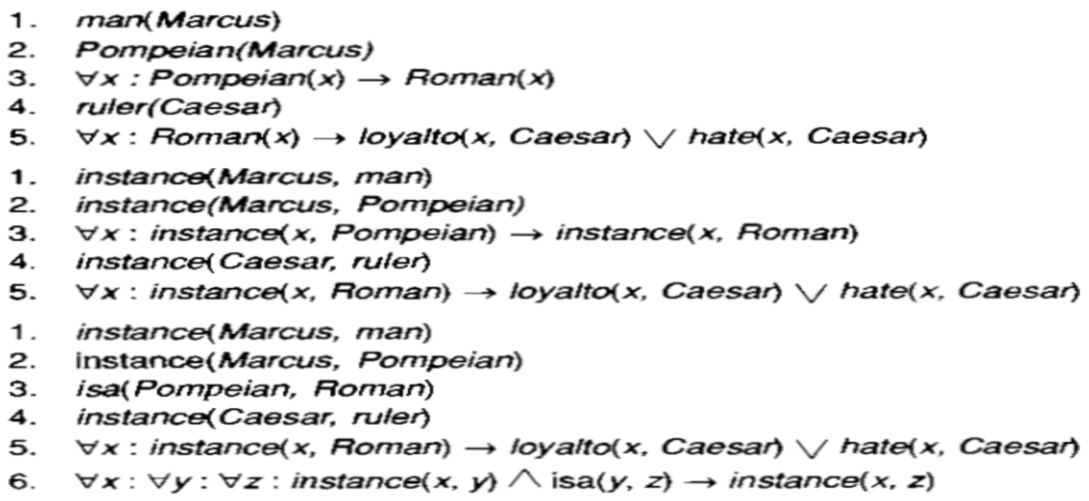
Representing Instance & Isa Relationships

- Attributes “ IsA ” and “ Instance ” support property inheritance and play important role inknowledge representation.
- The ways these two attributes "instance" and "isa", are logically expressed are shown in theexample below :

Example : A simple sentence like "Joe is a musician"

Here "is a" (called IsA) is a way of expressing what logically is called a class-instancerelationship between the subjects represented by the terms "Joe" and "musician".

◊ "Joe" is an instance of the class of things called "musician". "Joe" plays



1. *man(Marcus)*
2. *Pompeian(Marcus)*
3. $\forall x : Pompeian(x) \rightarrow Roman(x)$
4. *ruler(Caesar)*
5. $\forall x : Roman(x) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$

1. *instance(Marcus, man)*
2. *instance(Marcus, Pompeian)*
3. $\forall x : instance(x, Pompeian) \rightarrow instance(x, Roman)$
4. *instance(Caesar, ruler)*
5. $\forall x : instance(x, Roman) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$

1. *instance(Marcus, man)*
2. instance(*Marcus, Pompeian*)
3. *isa(Pompeian, Roman)*
4. *instance(Caesar, ruler)*
5. $\forall x : instance(x, Roman) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$
6. $\forall x : \forall y : \forall z : instance(x, y) \wedge \text{isa}(y, z) \rightarrow instance(x, z)$

the role of instance,"musician" plays the role of class in that sentence.

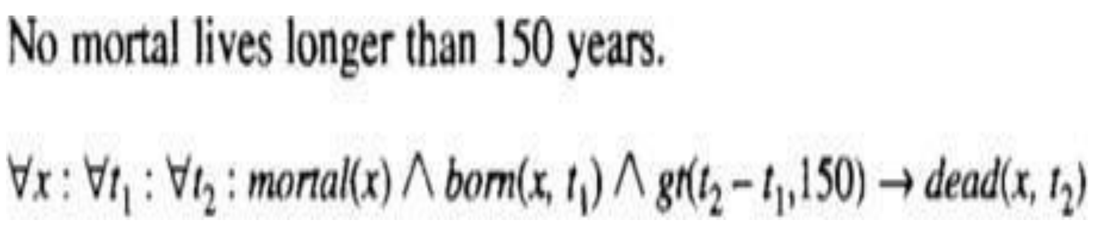
<u>Computable Functions and Predicates</u>

- All the simple facts can be expressed as combination of individual predicates such astryassassinate(Marcus, Caesar)

This is fine if the number of facts is not very large. But suppose if we want to express simple factssuch as greater-than and less-than relationships:

Computable predicates greater-than(1, 0)greater-than(2, 1)
greater-than(3, 2)
….

less-than(0, 1)
less-than(1, 2)
less-than(2, 3)
….



No mortal lives longer than 150 years.

$$\forall x : \forall t_1 : \forall t_2 : mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \rightarrow dead(x, t_2)$$

If someone dies, then he is dead at all later times.

$$\forall x : \forall t_1 : \forall t_2 : died(x, t_1) \wedge gt(t_2, t_1) \rightarrow dead(x, t_2)$$

INFERENCE IN PREDICATE LOGIC

1. **MODUS PONENS- GENERALIZED MODUS PONEN METHOD**

$$\frac{p1', p2', \ldots, pn', (p1 \wedge p2 \wedge \ldots \wedge pn \Rightarrow q)}{Subst(\theta, q)}$$

where $p_i'\theta = p_i\,\theta$ for all i

- GMP used with KB of definite clauses (positive literal)
- All variables assumed universally quantified

2. **AND ELIMINATION**

This says that, from a conjunction, any of the conjuncts can be inferred.

$$\frac{\alpha \wedge \beta}{\alpha}, \alpha \wedge \beta = \alpha, \beta$$

Example: Marcus was a man and a Pompiean

Man (Marcus) ^ Pompeian (Marcus). We infer : Man (Marcus), Pompeian (Marcus). We infer, Marcus was a Pompeian. Marcus was a man

3. **CHAIN RULE:**

$P \Rightarrow Q \wedge Q \Rightarrow R$

$\therefore\ P \Rightarrow R$

Example: Marcus was a man.

Propositional Logic:

Man (Marcus)

All men are mortal.

Propositional Logic:

$\forall$x: Man(x) $\Rightarrow$ Mortal(x)

We infer, Mortal (Marcus)

We infer, Marcus is mortal.

4. SUBSTITUTION

- Substitution of variables by *ground terms*: SUBST({g/v},P)
  - Result of SUBST({Harry/x, Sally/y}, Loves(x,y)): Loves(Harry,Sally)
  - Result of SUBST({John/x}, King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)): King(John) $\wedge$ Greedy(John) $\Rightarrow$ Evil(John)

5. UNIVERSAL INSTANTIATION

- A universally quantified sentence entails every instantiation of it:

$$\frac{\forall_v P(v)}{\text{SUBST}(\{g/v\}, P(v))}$$

for any variable v and ground term g

- E.g., $\forall$x King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x) yields: King(John) $\wedge$ Greedy(John) $\Rightarrow$ Evil(John) King(Richard) $\wedge$ Greedy(Richard) $\Rightarrow$ Evil(Richard) King(Father(John)) $\wedge$ Greedy(Father(John)) $\Rightarrow$ Evil(Father(John))

6. **EXISTENTIAL INSTANTIATION**

- An existentially quantified sentence entails the instantiation of that sentence with a newconstant:

$$\frac{\exists v\ P(v)}{\text{SUBST}(\{C/v\}, P(v))}$$

for any sentence P, variable v, and constant C that does not appear elsewhere inthe knowledge base

- E.g., $\exists x\ Crown(x) \wedge OnHead(x,John)$ yields:

$Crown(C_1) \wedge OnHead(C_1,John)$

provided C_1 is a new constant symbol, called a *Skolem constant*

7. RESOLUTION-PREDICATE LOGIC

1. Convert all the statements of F to clause form.
2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.
3. Repeat until either a contradiction is found or no progress can be made

   (a) Select two clauses. Call these the parent clauses.

   (b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exceptions: If there is one pair of literals T1 and ¬T2 such that one of the parent clause contains T2 and the other contains t1 and if t1 and t2 are unifiable, then neither T1 nor T2 should appear in the resolvent. We call T1 and T2 as complementary literals. Use the substitution produced by the unificationto create the resolvent. If there is more than pair of complementary literals, only one pair should be omitted from the resolvent.

   (c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

There exists a procedure for making the choice that can speed up the process considerably

- Only resolve pairs of clauses that contain complementary literals
- Eliminate certain clauses as soon as they are generated so that they cannot participate in later resolutions.
- Whenever possible resolve either with one of the clauses that is part of the statement we are trying to refute or with clause generated by a resolution with such clause. This is called set of support strategy
- Whenever possible resolve with clauses that have a single literal. Such resolution generate new clauses with fewer literals. This is called unit preference strategy.

<u>Problem 1</u>

1. All people who are graduating are happy.
2. All happy people smile.
3. Someone is graduating.
4. Conclusion: Is someone smiling?

<u>Solution:</u>

Convert in to predicate Logic

1. ∀x[graduating(x)→happy(x)
2. ∀x(happy(x)→smile(x)
3. ∃x graduating(x)
4. ∃x smile(x)

Convert to clausal form

(i) Eliminate the → sign

1. ∀x¬graduating(x) ∨happy(x)
2. ∀x¬happy(x) ∨smile(x)
3. ∃x graduating(x)
4. ¬∃x smile(x)

(ii) Reduce the scope of negation

1. ∀x¬graduating(x) ∨happy(x)
2. ∀x¬happy(x) ∨smile(x)
3. ∃x graduating(x)
4. ∀x ¬ smile(x)

(iii) Standardize variables apart

1. ∀x¬graduating(x) ∨happy(x)
2. ∀y¬happy(y) ∨smile(y)
3. ∃x graduating(z)
4. ∀w ¬ smile(w)

(iv) Move all quantifiers to the left

1. ∀x¬graduating(x) ∨happy(x)
2. ∀y¬happy(y) ∨smile(y)
3. ∃x graduating(z)
4. ∀w ¬ smile(w)

(v) Eliminate ∃

1. ∀x¬graduating(x) ∨happy(x)
2. ∀x¬happy(y) ∨smile(y)

3. graduating(name1)
4. ∀w ¬ smile(w)

(vi) Eliminate ∀
1. ¬graduating(x) ∨happy(x)
2. ¬happy(y) ∨smile(y)
3. graduating(name1)
4. ¬ smile(w)

(vii) Convert to conjunct of disjuncts form.

(viii) Make each conjunct a separate clause.
(ix) Standardize variables apart again.

¬happy(y) ∨smile(y)   ¬ smile(w)

¬graduating(x) ∨happy(x)   ¬happy(y)
. graduating(name1)   ¬graduating(x)

Non
e Thus, we proved someone is smiling.

Problem 2

Explain the unification algorithm used for reasoning under predicate logic with an example.

Consider the following facts

a. **Team India**

b. **Team Australia**

c. **Final match between India and Australia**

d. **India scored 350 runs, Australia scored 350 runs, India lost 5 wickets, Australialost 7 wickets.**

e. **The team which scored the maximum runs wins.**

f. **If the scores are same the team which lost minimum wickets wins the match.**

Represent the facts in predicate, convert to clause form and prove by resolution "India wins thematch".

Solution:

Convert in to predicate Logic

(a) team(India)

(b) team(Australia)

(c) team(India) $\wedge$ team(Australia)$\rightarrow$final_match(India, Australia)

(d) score(India,350) $\wedge$ score(Australia,350) $\wedge$wicket(India,5) $\wedge$ wicket(Australia,7)

(e) $\exists$x team(x) $\wedge$ wins(x)$\rightarrow$score(x, max_runs))

(f) $\exists$xy score(x,equal(y)) $\wedge$ wicket(x, min) $\wedge$final_match(x,y)$\rightarrow$win(x)

Convert to clausal form

(i) Eliminate the $\rightarrow$ sign

(a) team(India)

(b) team(Australia)

(c) $\neg$ (team(India) $\wedge$ team(Australia)) $\vee$ final_match(India, Australia)

(d) score(India,350) $\wedge$ score(Australia,350) $\wedge$wicket(India,5) $\wedge$ wicket(Australia,7)

(e) $\exists$x $\neg$ (team(x) $\wedge$ wins(x)) $\vee$score(x, max_runs))

(f) $\exists$xy $\neg$ (score(x,equal(y)) $\wedge$ wicket(x, min) $\wedge$final_match(x,y)) $\vee$win(x)

(ii) Reduce the scope of negation

(a) team(India)

(b) team(Australia)

(c) ¬ team(India) ∨ ¬team(Australia) ∨ final_match(India, Australia)

(d) score(India,350) ∧ score(Australia,350) ∧wicket(India,5) ∧ wicket(Australia,7)

(e) ∃x ¬ team(x) ∨ ¬wins(x) ∨score(x, max_runs))

(f) ∃xy ¬ score(x,equal(y)) ∨¬wicket(x, min_wicket) ∨¬final_match(x,y)) ∨win(x)

(iii) Standardize variables apart

(iv) Move all quantifiers to the left

(v) Eliminate ∃

(a) team(India)

(b) team(Australia)

(c) ¬ team(India) ∨ ¬team(Australia) ∨ final_match(India, Australia)

(d) score(India,350) ∧ score(Australia,350) ∧wicket(India,5) ∧ wicket(Australia,7)

(e) ¬ team(x) ∨ ¬wins(x) ∨score(x, max_runs))

(f) ¬ score(x,equal(y)) ∨¬wicket(x, min_wicket) ∨¬final_match(x,y)) ∨win(x)

(vi) Eliminate ∀

(vii) Convert to conjunct of disjuncts form.

(viii) Make each conjunct a separate clause.

(a) team(India)

(b) team(Australia)

(c) ¬ team(India) ∨ ¬team(Australia) ∨ final_match(India, Australia)

(d) score(India,3
50)
score(Australia,
350)
wicket(India,5)
wicket(Australi
a,7)

(e) ¬ team(x) ∨ ¬wins(x) ∨score(x, max_runs))

(f) ¬ score(x, equal(y)) ∨¬wicket(x, min_wicket) ∨¬final_match(x, y)) ∨win(x)

(ix) Standardize variables part again.

(x) <u>To prove:</u> win(India)

Disprove: ¬win(India)   ¬ score(x, equal(y)) ∨¬wicket(x, min_wicket)

∨¬final_match(x, y)) ∨win(x)   ¬win(India)

¬ team(India) ∨ ¬team(Australia) ∨ final_match(India, Australia)   ¬ score(x, equal(y))

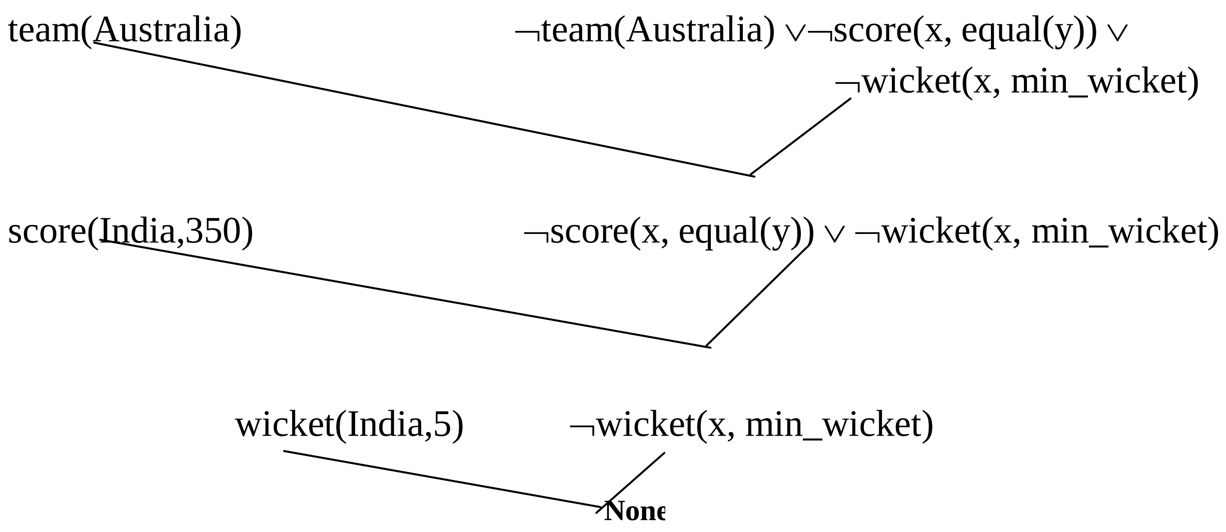
∨¬wicket(x,min_wicket)

∨¬final_match(x, y))

team(India)

¬ team(India) ∨ ¬team(Australia)

∨¬score(x,equal(y))∨¬ wicket(x, min_wicket)


team(Australia)
¬team(Australia) ∨¬score(x, equal(y)) ∨ ¬wicket(x, min_wicket)
score(India,350)
¬score(x, equal(y)) ∨ ¬wicket(x, min_wicket)
wicket(India,5)
¬wicket(x, min_wicket)
None

Thus, proved India wins match.

<u>Problem 3</u>

Consider the following facts and represent them in predicate form: F1. There are 500 employees in ABC company.
F2. Employees earning more than Rs. 5000 pay tax. F3. John is a manager in ABC company.
F4. Manager earns Rs. 10,000.

Convert the facts in predicate form to clauses and then prove by resolution: "John pays tax". <u>Solution:</u>

Convert in to predicate Logic

1. company(ABC) ∧employee(500,ABC)
2. ∃x company(ABC) ∧employee(x, ABC) ∧earns(x,5000)→pays(x, tax)
3. manager(John, ABC)
4. ∃x manager(x, ABC)→earns(x,10000)

Convert to clausal form

(i) **Eliminate the → sign**

1. company(ABC) ∧employee(500,ABC)
2. ∃x¬ (company(ABC) ∧employee(x, ABC) ∧earns(x,5000)) ∨ pays(x, tax)
3. manager(John, ABC)
4. ∃x ¬manager(x, ABC) ∨ earns(x, 10000)

(ii) **Reduce the scope of negation**

1. company(ABC) ∧employee(500,ABC)
2. ∃x¬ company(ABC) ∨¬employee(x, ABC) ∨¬earns(x,5000) ∨ pays(x, tax)
3. manager(John, ABC)
4. ∃x ¬manager(x, ABC) ∨ earns(x, 10000)

(iii) **Standardize variables apart**

1. company(ABC) $\wedge$employee(500,ABC)
2. $\exists$ x$\neg$ company(ABC) $\vee\neg$employee(x, ABC) $\vee\neg$earns(x,5000) $\vee$ pays(x, tax)
3. manager(John, ABC)
4. $\exists$ y $\neg$manager(x, ABC) $\vee$ earns(y, 10000)

(iv) Move all quantifiers to the left

(v) **Eliminate ∃**

1. company(ABC) ∧employee(500,ABC)
2. ¬ company(ABC) ∨¬employee(x, ABC) ∨¬earns(x,5000) ∨ pays(x, tax)
3. manager(John, ABC)
4. ¬manager(x, ABC) ∨ earns(y, 10000)

(vi) **Eliminate ∀**

(vii) **Convert to conjunct of disjuncts form.**

(viii) **Make each conjunct a separate clause.**

1. (a)company(ABC)
   (b)employee(500,
   ABC)
2. ¬ company(ABC) ∨¬employee(x, ABC) ∨¬earns(x,5000) ∨ pays(x, tax)
3. manager(John, ABC)
4. ¬manager(x, ABC) ∨ earns(y, 10000)

(ix) **Standardize variables apart again.** **<u>Prove :</u>**pays(John, tax)

<u>Disprove:</u> ¬pays(John, tax)

¬ company(ABC) ∨¬employee(x, ABC) ∨¬earns(x,5000) ∨ pays(x, tax) ¬pays(John,

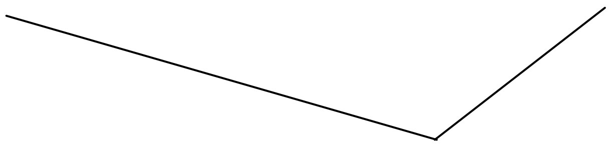
tax)

¬manager(x, ABC) ∨ earns(y, 10000) ¬ company(ABC)
∨¬employee(x,ABC)
∨¬earns(x,5000)

manager(John, ABC) ¬manager(x, ABC)
∨¬company(ABC)
∨¬employee(x, ABC)

company(ABC) ¬company(ABC) ∨¬employee(x, ABC)

employee(500,ABC) ¬employee(x, ABC)



None

Thus, proved john pays tax.

DIFFERENTIATE PREDICATE AND PROPOSITIONAL LOGIC.

Sl.No	Predicate logic	Propositional logic
1.	Predicate logic is a generalization of propositional logic that allows us to express and infer arguments in infinite models.	A proposition is a declarative statement that's either TRUE or FALSE (but not both).
2.	Predicate logic (also called predicate calculus and first-order logic) is an extension of propositional logic to formulas involving terms and predicates. The full predicate logic is undecidable	Propositional logic is an axiomatization of Boolean logic. Propositional logic is decidable, for example by the method of truth table
3.	Predicate logic have variables	Propositional logic has variables. Parameters are all constant
4.	A predicate is a logical statement that depends on one or more variables (not necessarily Boolean variables)	Propositional logic deals solely with propositions and logical connectives
5.	Predicate logic there are objects, properties, functions (relations) are involved	Proposition logic is represented in terms of Boolean variables and logical connectives
6.	In predicate logic, we symbolize subject and predicate separately. Logicians often use lowercase letters to symbolize subjects (or objects) and uppercase letter to symbolize predicates.	In propositional logic, we use letters to symbolize entire propositions. Propositions are statements of the form "x is y" where x is a subject and y is a predicate.

7.	Predicate logic uses quantifiers such as universal quantifier ("∀"), the existentialquantifier ("∃")	Prepositional logic has no quantifiers.
8.	**Example** Everything is green" as "∀x Green(x)"or "Something is blue" as "∃x Blue(x)".	**Example** Everything is green" as "G(x)"or "Something is blue" as "B(x)".