Unit - II

Artificial Intelligence

Logical Agents

Logical Agents use a process of inference to derive new representations about the world and use there new representations what to do?

Logical Agents claim that the human intelligence is achieved not by purely reflex mechanisms but by processes of reasoning that operate on internal representations of knowledge.

This A.I approach belongs to knowledge based Agents

Knowledge based Agents

→ The Central Component of a knowledge based agents is it's knowledge base.

→ A knowledge base is a set of Sentences
↳ (knowledge representation language)

axiom:- when a Sentence is taken as given without being derived from other Sentences.

Inference:- Deriving new Sentences from old.
→ Inference must obey the requirement that when one Ask's a question of the knowledge base, the ans should follow from what has been told to the knowledge base Previously.

To add new Sentences use TELL and ADD operations.

Background knowledge :- The agent maintains a knowledge base, KB, which may initially contain some background knowledge.

A generic knowledge Base Agent

function KB-AGENT (Percept) returns an action

Persistent: KB, t, a counter, initially 0, indicating time.

TELL (KB, MAKE-PERCEPT-Sentence (Percept, t))

action ← ASK (K.B, MAKE-ACTION-QUERY (t))

TELL (KB, MAKE-ACTION-SENTENCE (action, t))

t ← t+1

return action

System Building approach for knowledge base

A KB System Can be built by TELLing it what it needs to know starting with an empty knowledge base, the agent designer Can tell Sentences one by one until the agent knows how to operate in its enviornment.

This is called declarative approach.

Procedural Approach :- Procedural approach encodes desired behaviors directly as program code.

Logic :- A logic is must for defining the semantics of meaning of sentences.

Semantics :- The semantics deal with the truth of each Sentences with respect to each possible world.

Example :- Sentence "x+y = 4" is true in a world where x=2 & y=2 but false in a world where x=1 & y=1,

In standard logics. every sentence must be either true or false in each possible world - there is no "in between".

→ use the term model in place of possible world.

Syntax :- In knowledge Base. English language Sentences are expressed according to the syntax of representation language.

Satisfaction :- If a Sentence α is true in model m, we say that m satisfies α or sometimes, m is a model of α.

PROPOS

Logical Inference

we use Entailment to carry out logical

Inference: $\alpha \models \beta$ — α entails β

in every model if α is true, β is also true.

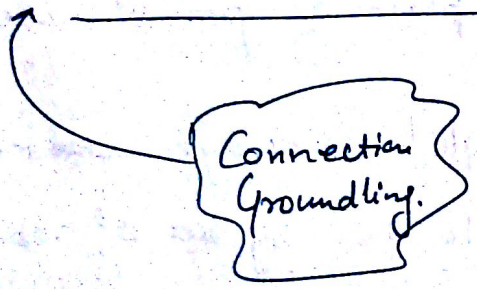
An Inference algorithm that derives only entails Sentences is called sound or Truth Preserving.//

Property of Completeness:-

An inference algorithm is Complete if it can derive any sentence that is entailed.

Groundling:- The Connection between logical reasoning processes and the real enviornment in which the agent exists =

"how do we know that KB is true in the real world"?



Propositional Logic

↳ Check Truth of Sentences

↳ Entailment (Means the relation between sentences)

↳ we use uppercase letters and Subscripts or other letters also

Logical Connectives are used to represent Complex Sentences.

1) $\neg$ (not) negation $\neg W_{1,3}$ (negation of $W_{1,3}$)

2) $\wedge$ (and) Example $W_{13} \wedge P_{3,1}$ is called a Conjuction

3) $\vee$ (or) disjunction $W_{1,3} \vee W_{2,2}$

4.) $\Rightarrow$ (implies) $W_{13} \Rightarrow \neg W_{2,2}$ is called an implication.

Implications are also known as rules or if_then Statements. Sometimes implies symbol may be written as $\supset$.

5) $\Leftrightarrow$ (if and only if) $W_{1,3} \Leftrightarrow \neg W_{2,2}$ is a biconditional

if may be written as $\equiv$

Sentence → Atomic Sentence | Complex Sentence

Atomic Sentence ⇒ True | False | P | Q | R

Complex Sentence ⇒ (Sentence) | [Sentence]

| ¬ Sentence

| Sentence ∧ Sentence

| Sentence ∨ Sentence

| Sentence ⇒ Sentence

| Sentence ⇔ Sentence

Operator Precedence :— ¬, ∧, ∨, ⇒, ⇔

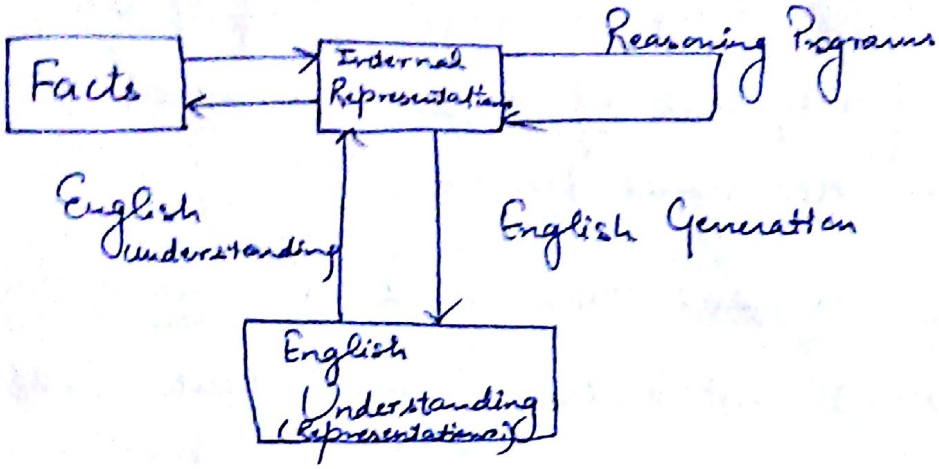
Five Rules for Complex Sentences

{where m is a model}

- ¬P is true iff P is false in m.
- P ∧ Q is true iff both P and Q are true in m.
- P ∨ Q is true iff either P or Q is true in m
- P ⇒ Q is true unless P is true and Q is false in m
- P ⇔ Q is true iff P and Q are both true or both false in m

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	false	True	false	false	true	true
false	True	True	false	true	true	false
true	false	false	false	true	false	false.
true	true	false	true	true	true	true

Knowledge Representation



Properties for knowledge representation Systems.

→ Representational adequacy :-
The ability to represent the required knowledge.

→ Inferential adequacy :-
Ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original

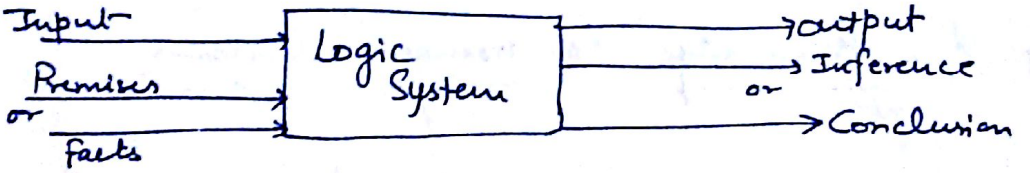
→ Inferential efficiency : The ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides.

→ Acquisitional efficiency :- The ability to acquire new knowledge using automatic methods wherever possible. rather than reliance on human intervention.

Theory for first order logic

Logic means drawing Conclusions on the basis of Conditions Logic Can be defined as scientific study of the proces of reasoning of a system of rules and procedures that help in the reasoning proces

The logic proceeded takes input as some information (called Premises) and produces some outputs Called Conclusions:



Logic Can be classified into 2 Categories

→ Prepositional Logic
→ Predicate Logic

3 types of Logic	
ordinary	1+1=2
Binary	1+1=10
boolean	1+1=1

PREPOSITIONAL LOGIC

Simplest form of logic where all statements made are prepositions. In prepositional logic, takes only 2 values, either the preposition is true or false.

1. It is raining
2. It is Sunny
3. Diamond is a hard metal.

FOPC

First order logic

* Propositional logic is a declarative language

There are two kinds of Propositions:

1.) Atomic Propositions (Simple Proposition)
2) Molecular Propositions (Compound Proposition) or Propositional Calculus.

Sample for molecular propositions

"Ram is a doctor and his clinic is in Delhi".

Premise :- A Premise is a claim that is a reason for or objection against some other claim.

Some Commonly used logical Equivalence Normal forms in Propositional logic.

. CNF (Conjunctive Normal Form)
. DNF (Disjunctive Normal Form)

A formula is said to be in CNF iff

$A = A_1 \cap A_2 \cap A_3 \ldots\ldots A_n \quad n \geq 1$

($\cap$: disjunction)

A formula is said to be in DNF iff

$A = A_1 \cup A_2 \cup A_3 \ldots\ldots A_n \quad n \geq 1$

($\cup$: Conjunction)

Conversion Procedure

Step 1. Eliminate implications and double Conditionals, for this use the laws.

Step 1

$A \to B = \neg A \cup B$

$A \leftrightarrow B = (A \to B) \wedge (B \to A)$

$(\neg A \cup B) \quad (\neg B \cup A)$

Step 2 Reduce the scope of not Symbol by the formula $\neg(\neg A) = A$ & apply De-morgan's

$\neg(A \cup B) = \neg A \wedge \neg B$

Step 3 use distributive laws of other Equivalent formulas

$A \wedge (B \cup C) = (A \wedge B) \cup (A \wedge C)$

$A \cup (B \wedge C) = (A \cup B) \wedge (A \cup C)$

Example

$A \to ((B \wedge C) \to (D))$

sol

$A \to (\neg(B \wedge C) \cup D)$ — using Step 1

$A \to (\neg B \cup \neg C) \cup D$ — using Step 2

$\neg A \cup ((\neg B \cup \neg C) \cup D)$

$\neg A \wedge \neg B \wedge \neg C \cup D$

$(A \cup C) \wedge (\neg B \wedge C)$

$\neg (CA \rightarrow B) \rightarrow C$

Resolution Proof

Resolution is a Proof Procedure that Carries out a single operation, the Variety of Processes involved increasing with statements in Predicate logic.

Basics of Resolution

1. A Simple Iterative Process, in each Step at which two clauses Called parent Clauses are Compared, attempts a new clause that has been inferred from them. the new Clause represent ways that the two parent clauses interact with each other.

2. The literal must occur in positive form in one Clause and in -ve form in other clause.

3. The Resolution is obtained by Combining all the literals of the two parent clauses.

4. If the clause that is produced is the empty clause. then a Contradiction has been found.

A: $P \cup Q$

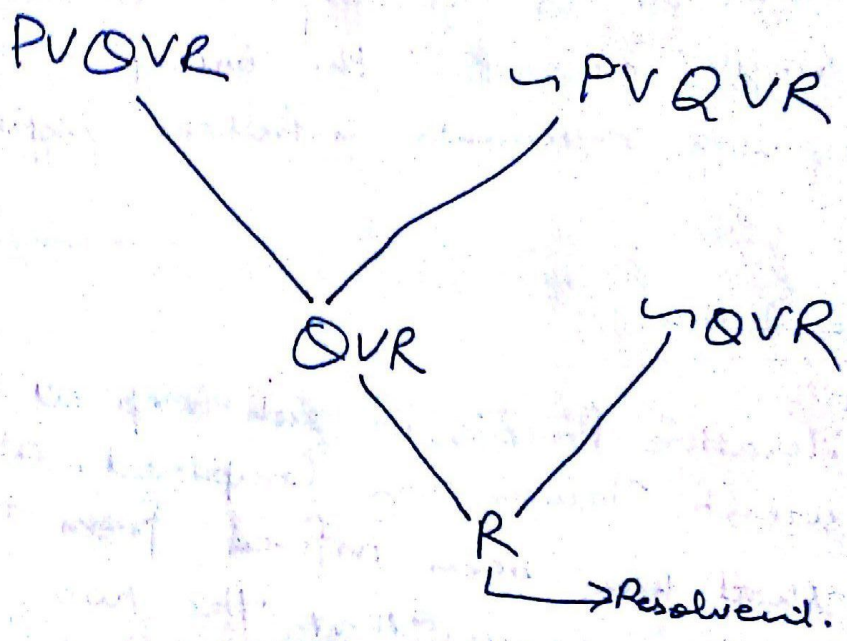
B: $\neg P \vee \neg Q$.

NIL $\rightarrow$ Contradiction.

Example:- Consider the following clauses:-

A: PVQVR
B: PV¬QVR
C: ¬QVR

and find the Solvent-



Propositional Calculus formula

First order Inference

Inference Rules for quantifiers

Example

$$\forall x \ \text{king}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$$

All Greedy kings are Evil.

Then it is quite permissible to infer any of the sentences like

$$\text{king}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$$

$$\text{king}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard}).$$

Universal Instantiation

UI Says that we can infer any Sentence obtained by Substituting a ground term.

~~Extens~~ Existential Instantiation

In this rule the Variable is replaced by single new Constant Symbol

Example :— $\exists x \ \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$

we can infer the Sentence

$$\text{Crown}(C_1) \wedge \text{onHead}(C_1, \text{John})$$

as C_1 doesnot appear elsewhere in the knowledgebase Basically the existential sentence Says there is some object Satisfying a Condition and applying the existential instantiation rule just gives a name to that object.

the name of the new object must not be created or be redundant and well known as Skolem Constant

Inferential Equivalence

Example $\exists x$ kill (x, victim)

Instead we use

kill(murderer, Victim)

the new knowledge base is not logically equivalent to the old, but it can be shown to be inferentially Equivalent

use of UI to Produce many different Consequences in addition to existential Instantiation

Reduction to Propositional Inference

Example

$\forall x$ king(x) $\wedge$ greedy(x) $\Rightarrow$ Evil(x)

king (John)
greedy (John)
Brother (Richard, John)

} given

Then we apply UI to first Sentence. using all possible ground term Substitutions

like king (John) $\wedge$ Greedy (John) $\Rightarrow$ Evil (John)

king (Richard) $\wedge$ Greedy (Richard) $\Rightarrow$ Evil (Richard)

Now the knowledge base is essentially Propositional. if we view the ground atomic sentences

and King(John), Greedy(John) as Proposition Symbols.

Therefore we can now apply any propositional algo's to obtain Conclusions Such as Evil(John)

Else use Entailment.

First Order Inference Rules

Example

Suppose that instead of ~~knowledge~~ knowing Greedy(John) we know that Everyone is greedy

$\forall(y)$ greedy(y).

we will be able to Conclude that Evil(John). because we know that john is a king (given) and John is greedy (because Everyone is Greedy)

we need to find a substitution both for the Variables in the implication sentence and for the Variables in the Sentences that are in the knowledge base.

In this Case applying the substitution $\{x/John, y/John\}$ to the implication premises king(x) and greedy(x) and the knowledge base sentences king(John) and greedy(y) will make them identical. thus we can infer the Conclusion of the implication.

Generalized modus ponens rule

Unification

first order inference rules require finding substitutions that make different logical expressions look identical. This process is called unification and is a key component of all first order inference algorithms.

The unify algorithm takes two sentences and returns a unifier for them if one exists.

UNIFY(p, q) = θ where SUBST(θ, p) = SUBST(θ, q)

where θ is the substitution and p and q are the sentences

Example

AskVars(knows(John, x))

If we ask whom does John know?

So, Ans to this query can be found by finding all sentences in the knowledge base that unify with knows(John, x)

Results may be

UNIFY(knows(John, x), knows(John, Jane)) = {x/Jane}

UNIFY(knows(John, x), knows(y, bill)) = {x/Bill, y/John}

UNIFY(knows(John, x), knows(x, Elizabeth)) = fail

Last unification fails because x cannot take on the values John and Elizabeth at the same time.

Example knows(x, Elizabeth) means "Everyone knows Elizabeth".

and we should infer that John knows Elizabeth

this problem arise because the two sentences happen to use same Variable x.

To avoid this we can rename x in Knows (x, Elizabeth) to x_{17} a new Variable name

Now unification will work.

Unify (Knows (John, x), knows (x_{17}, Elizabeth)) = $\{x/\text{Elizabeth}, x_{17}/\text{John}\}$.

Problems in Unification

There could be more that one such unifier that returns a substitution that makes the two arguments look the same.

Example UNIFY (Knows (John, x), Knows (y, z))

Could return $\{y/\text{John}, x/z\}$ or

$\{y/\text{John}, x/\text{John}, z/\text{John}\}$

find MGU Most General Unifier. that is Unique up to renaming and substitution of Variables.

First Unifier

Forward chaining

Bottom up approach

→ starts with available data and uses inference rules to extract more data until an optimal goal is reached.

Example 1. If X Croaks and eats flies Then X is a frog

2. If X is a frog Then X is green.

→ X Croaks and eats flies

→ X is a frog

→ X is green (goal accomplished)

Commonly used in Expert Systems such as CLIPS.

~~Advantage of forward chaining.~~

Advantage of forward chaining

Reception of new data can trigger new inferences, which makes the engine better suited to dynamic situations in which conditions are likely to change.

Backward Chaining

→ Starts with Conclusion and works backward

→ goal is broken into many subgoals or subgoals which can be solved more easily known as top down approach.

Example

1. If X Croaks and eat flies Then X is a frog.
2. If X is a frog Then X is green.

In this Conclusion the Then clause matches the goal that (X is green) It is not yet known that X is a frog. So the if Statement is added to the goal list (in order for X to be green he must be a frog). The rulebase is again searched and this time the first rule is Selected because its Then clause matches the new goal that was just added to the list (whether X is a frog).

Because the list of goals determines which rules are Selected and used. This method is Called goal driven.

Employed By Expert System

Resolution

Consider the following set of facts

1. Marcus was a man.
   man(marcus)
2. Marcus was a Pompeian
   Pompeian(marcus)
3. Marcus was born in 40 A.D.
   born(Marcus, 40)
4. All men are mortal
   $\forall x$: man (x) $\rightarrow$ mortal(x)

5. All pompeians died when the Volcano erupted in 79 A.D

erupted (volcano, 79) $\wedge$ $\forall x$: [Pompeian (x) $\rightarrow$ died (x, 79)]

6. 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)

7. It is now 1991

now = 1991

8. Alive means not dead.

$\forall x : \forall t : [$alive $(x, t) \rightarrow \neg$ dead $(x, t) \rightarrow$ alive $(x, t)]$

9. 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)

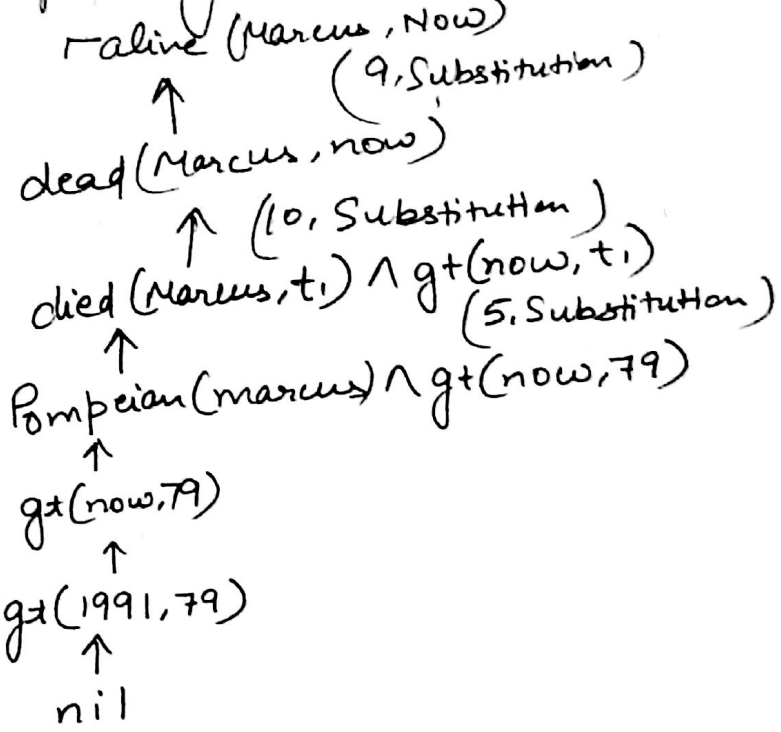
Proving $\neg$ alive (Marcus, now)

Notice these Proofs that whenever a Strut of the form

$a \wedge b \rightarrow ($

Resolution

way of proving Marcus is dead.



A Proof Procedure that is carried out in a single operation the Variety of Processes involved in reasoning with Statements in Predicate logic.

Resolution is a procedure, which gains its efficiency from the fact that it operates on Statements that have been Converted to a Very Convenient- standard form.

resolution attempts to show that the negation of the Statement produces a Contradiction with the known Statements

Steps to Resolution / Clause form

1. Eliminate $\rightarrow$ using the fact that $a \rightarrow b$ is equivalent to $\neg a \vee b$.

2. Reduce the scope of each $\neg$ to a single term, using the fact that $\neg(\neg P) = P$, deMorgan's law or $\neg(a \wedge b) = \neg a \vee \neg b$ and $\neg(a \vee b) = \neg a \wedge \neg b$

3. Standardize Variables so that each quantifier binds a unique Variable. Since Variables are just dummy names

$$\forall x = P(x) \vee \forall x : Q(x)$$

Converted to

$$\forall (x) : P(x) \vee \forall y : Q(y)$$

4. Move all Quantifiers to the left of the formula changing their relative order. This is Possible if there is no Conflict among Variable names

$$\forall x : \forall y : \forall z : [\neg \text{Roman}(x) \vee \neg \text{know}(x, \text{Marcus})]$$
$$\vee [\text{hate}(x, \text{Caesar}) \vee (\neg \text{hate}(y, z) \vee \text{thinkCrazy}(x, y))]$$

5 Eliminate existential quantifiers.

Example $\exists y : \text{President}(y)$

Can be transformed to $(\text{President}(S_1))$

where S_1 is Substitute,

If Existential quantifiers occur within the scope of universal quantifier then the value that satisfies the Predicate may depend on the values of Universally quantified Variables

Example $\forall x : \exists y : \text{father of } (y, x)$

6. Drop the Prefix.
So, all remaining Variables are Universally quantified.

7. Convert the matrix into a Conjunction of disjuncts
[i.e $(a \wedge b) \vee c = (a \vee c) \wedge (b \wedge c)$] and remove Parenthesis

However it is also necessary to exploit associative property. $[(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)]$.

8. Create a separate clause Corresponding to each Conjunct. If o in order a w-ff to be true all the clauses that are generated from it must be true.

9. Standardize apart the Variables in the set of clauses generated in Step 8.
(Means Rename the Variables so that no two clauses make reference to the same Variable)
i.e $(\forall x : P(x) \wedge Q(x)) = \forall x : P(x) \wedge \forall (x) : Q(x)$.

All romans who know Marcus either hate Caesar or think that anyone who hates anyone is crazy

$$\forall x : [Roman(x) \wedge know(x, Marcus)] \rightarrow [hate(x, Caesar) \vee (\forall y : \exists z : hate(y,z) \rightarrow thinkCrazy(x,y))]$$

CNF

$$\neg Roman(x) \wedge \neg know(x, Marcus) \vee hate(x, Caesar) \vee \neg hate(y,z) \vee thinkCrazy(x,z)$$

Conversion to clause form

1. $\forall x : \neg [Roman(x) \wedge know(x, Marcus)] \vee [hate(x, Caesar) \vee (\forall y : \neg (\exists z : hate(y,z)) \vee thinkCrazy(x,y))]$

2. $\forall x : [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\forall y : \forall z : \neg hate(y,z) \vee thinkCrazy(x,y))]$

3. $\forall x : \forall y : \forall z : [\neg Roman(x) \vee \neg know(x, Marcus)] \vee [hate(x, Caesar) \vee (\neg hate(y,z) \vee thinkCrazy(x,y))]$.

4. Ajust Skolem Constants

5. Drop prefix

$[\neg Roman(x) \vee \neg know(x, marcus)] \vee [hate(x, Caesar) \vee \neg hate(y,z) \vee thinkCrazy(x,y)$