Parul® University

ARTIFICIAL INTELLIGENCE

(203105322)

Dr. Pooja Sapra, Associate Professor
Information Technology

KNOWLEDGE REPRESENTATION

The object of a knowledge representation is to express knowledge in a computer tractable form, so that it can be used to enable our AI agents to perform well.

A knowledge representation language is defined by two aspects:

1. Syntax The syntax of a language defines which configurations of the components of the language constitute valid sentences.

2. Semantics The semantics defines which facts in the world the sentences refer to,and hence the statement about the world that each sentence makes.

This is a very general idea, and not restricted to natural language.

GOOD Knowledge Representation Scheme

A good knowledge representation system for any particular domain should possess the following properties:

1. Representational Adequacy – the ability to represent all the different kinds of knowledge that might be needed in that domain.

2. Inferential Adequacy – the ability to manipulate the representational structures to derive new structures (corresponding to new knowledge) from existing structures.

3. Inferential Efficiency – the ability to incorporate additional information into the knowledge structure which can be used to focus the attention of the inference mechanisms in the most promising directions.

4. Acquisitional Efficiency – the ability to acquire new information easily. Ideally the agent should be able to control its own knowledge acquisition, but direct insertion of information by a ‘knowledge engineer’ would be acceptable.

Parul® University

DIGITAL LEARNING CONTENT

Approaches to knowledge representation

Simple Relational Knowledge	Inheritable Knowledge	Inferential Knowledge	Procedural Knowledge
• This is a relational method of storing facts which is among the simplest of the method. This method helps in storing facts where each fact regarding an object is providing in columns. This approach is prevalent in DBMS (database management systems).	• Knowledge here is stored hierarchically. A well-structured hierarchy of classes is formed where data is stored, which provides the opportunity for inference. Here we can apply inheritance property, allowing us to have inheritable knowledge. This way, the relations between instance and class (aka instance relation) can be identified. Unlike Simple Relations, here, the objects are represented as nodes.	• In this method, logics are used. Being a very formal approach, facts can be retrieved with a high level of accuracy.	• This method uses programs and codes that use simple if-then rules. This is the way many programming languages such as LIST, Prolog save information. We may not use this method to represent all forms of knowledge, but domain-specific knowledge can very efficiently be stored in this manner.

LOGIC

A logic language consists of semantics and syntax

Semantics: What the sentences mean.

Syntax: How sentences can be assembled.

Parul® University

Propositional Logic: Syntax I

Vocabulary

A set of *propositional* symbols

P, Q, R, ….

A set of logical connectives

$\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$

$\wedge$ (and) $\vee$ (or) $\neg$(not) $\Rightarrow$ (implication) $\Leftrightarrow$ (equivalence)

Parenthesis (for grouping) ( )

Logical constants

`True`, **`False`**

Propositional Logic: Syntax II

Each symbol P, Q, R etc is a (atomic) sentence
Both **`True`** and **`False`** are (atomic) sentences
A sentence wrapped in parentheses is a sentence

If α and β are sentences, then so are

- $\alpha \wedge \beta$ conjunction
- $\alpha \vee \beta$ disjunction
- $\neg\alpha$ negation
- $\alpha \Rightarrow \beta$ implication
- $\alpha \Leftrightarrow \beta$ equivalence

The above are complex sentences

Truth Tables

- Conjunction ($\wedge$)

α	β	$\alpha \wedge \beta$
false	false	false
false	true	false
true	false	false
true	true	true

- Disjunction ($\vee$)

α	β	$\alpha \vee \beta$
false	false	false
false	true	true
true	false	true
true	true	true

- Implication ($\rightarrow$)

α	β	$\alpha \rightarrow \beta$
false	false	true
false	true	true
true	false	false
true	true	true

- Negation ($\neg$)

α	$\neg\alpha$
false	true
true	false

Conversion to Clause Form

1. Eliminate →.

   $P \rightarrow Q \equiv \neg P \vee Q$

2. Reduce the scope of each ¬ to a single term.

   $\neg(P \vee Q) \equiv \neg P \wedge \neg Q$
   $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$
   $\neg \forall x: P \equiv \exists x: \neg P$
   $\neg \exists x: p \equiv \forall x: \neg P$
   $\neg\neg P \equiv P$

3. Standardize variables so that each quantifier binds a unique variable.

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

Conversion to Clause Form

4. Move all quantifiers to the left without changing their relative order.

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

5. Eliminate $\exists$ (Skolemization).

   $\exists x: P(x) \equiv P(c)$ Skolem constant

   $\forall x: \exists y\ P(x, y) \equiv \forall x: P(x, f(x))$ Skolem function

6. Drop $\forall$.

   $\forall x: P(x) \equiv P(x)$

7. Convert the formula into a conjunction of disjuncts.

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

8. Create a separate clause corresponding to each conjunct.

9. Standardize apart the variables in the set of obtained clauses.

Parul® University

Conversion to Clause Form

1. Eliminate →.
2. Reduce the scope of each ¬ to a single term.
3. Standardize variables so that each quantifier binds a unique variable.
4. Move all quantifiers to the left without changing their relative order.
5. Eliminate ∃ (Skolemization).
6. Drop ∀.
7. Convert the formula into a conjunction of disjuncts.
8. Create a separate clause corresponding to each conjunct.
9. Standardize apart the variables in the set of obtained clauses.

Parul® University

Conversion to Clause Form : Example

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Pompeians were either loyal to Caesar or hated him.
6. Every one is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.

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).
6. $\forall x$: $\exists y$: loyalto(x, y).
7. $\forall x$: $\forall y$: person(x) $\wedge$ ruler(y) $\wedge$ tryassassinate(x, y) $\rightarrow \neg$loyalto(x, y).
8. tryassassinate(Marcus, Caesar).

Parul®
University

RESOLUTION

- Resolution is such a procedure, which gains its efficiency from the fact that it operates on statements that have been converted to a very convenient standard form, which is described below.
- Resolution produces proofs by *refutation*. In other words, to prove a statement (i.e., show that it is valid), resolution attempts to show that the negation of the statement produces a contradiction with the known statements (i.e. , that it is unsatisfiable).

The Basis of Resolution

- The resolution procedure is a simple iterative process: at each step, two clauses, called the parent clauses, are compared (resolved), yielding a new clause that has been inferred from them.
- The new clause represents ways that the two parent clauses interact with each other. Suppose that there are two clauses in the system:

winter V summer

¬ winter V cold

Now we observe that precisely one of winter and ¬ winter will be true at any point. If winter is true, then cold must be true to guarantee the truth of the second clause.

Algorithm: Propositional Resolution

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

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

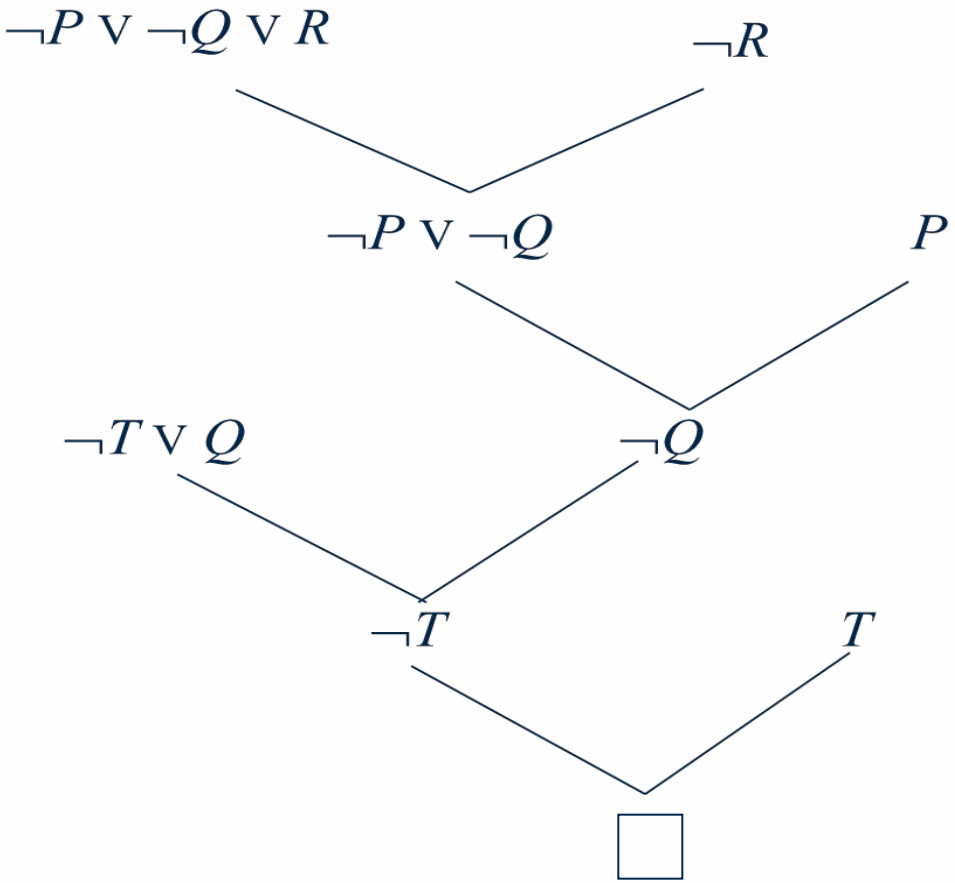
  b) Resolve them together. The *resolvent* will be the disjunction of all of the literals of both of the parent clauses with the following exception: If there are any pairs of literals *L* and ¬ *L* such that one of the parent clauses contains *L* and the other contains ¬*L*, then select one such pair and eliminate both *L* and ¬*L* 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.

A Few Facts in Propositional Logic

Given Axioms	Clause Form	
P	P	(1)
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$	(2)
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$	(3)
	$\neg T \vee Q$	(4)
T	T	(5)

Resolution in Propositional Logic



Resolution in Predicate Logic

- Convert all the propositions of *F* to clause form.
- Negate *P* and convert the result to clause form. Add it to the set of clauses obtained in step 1.
- 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 of the literals of both of the parent clauses with the following exception: If there are any pairs of literals *L* and ¬ *L* such that one of the parent clauses contains *L* and the other contains ¬*L* ***and they unify with each other***, then select one such pair and eliminate both *L* and ¬*L* 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.

Parul® University

Algorithm: Unify(L1,L2)

1. If *L*1 or *L*2 is a variable or constant, then:
   a) If *L*1 and *L*2 are identical, then return NIL.
   b) Else if *L*1 is a variable, then if *L*1 occurs in *L*2 then return FAIL, else return {(*L*2/*L*1)}.
   c) Else if *L*2 is a variable, then if *L*2 occurs in *L*1 then return FAIL, else return {(*L*1/*L*2)}.
   d) Else return FAIL.
2. If the initial predicate symbols in *L*1 and *L*2 are not identical, then return FAIL.
3. If *L*1 and *L*2 have a different number of arguments, then return FAIL
4. Set *SUBST* to NIL.
5. For *i* □ 1 to number of arguments in *L*1:
   a) Call Unify with the *i*th argument of *L*1 and the *i*th argument of *L*2, putting result in *S*.
   b) If *S* = FAIL then return FAIL.
   c) If S is not equal to NIL then:
      i. Apply *S* to the remainder of both *L*1 and *L*2.
      ii. *SUBST* := APPEND(*S*, *SUBST*).
6. Return *SUBST*.

Parul® University

DIGITAL LEARNING CONTENT

Unification

Q(x)
P(y) ⇒ FAIL

P(x)
P(y) ⇒ *x/y*

P(Marcus)
P(y) ⇒ Marcus/*y*

P(Marcus)
P(Julius) ⇒ FAIL

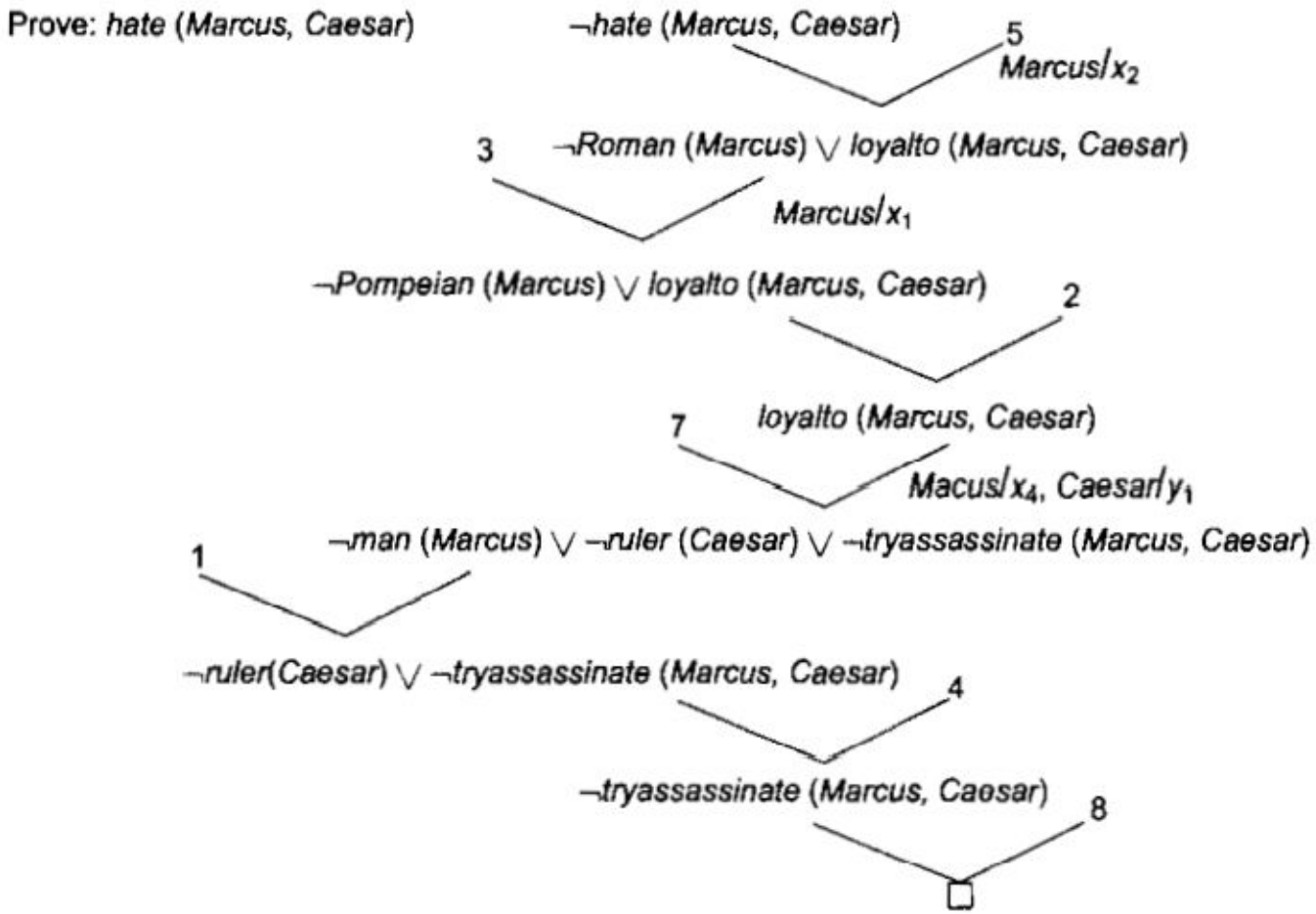
P(x,x)
P(y,y) ⇒ *(y/x)*
P(y,z) ⇒ *(z/y , y/x)*

A Resolution Proof in Predicate Logic

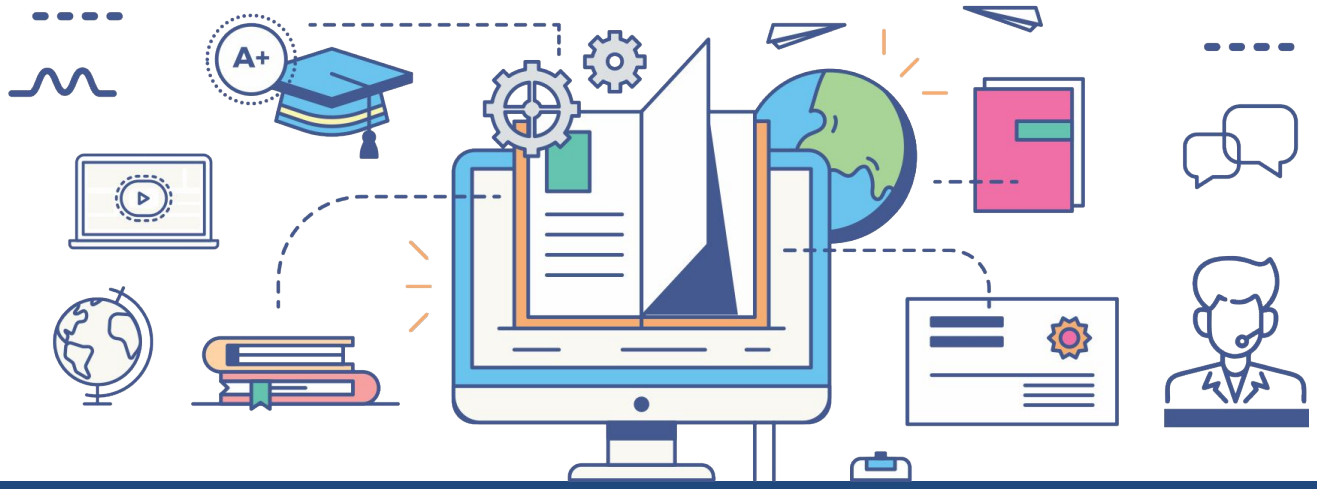
- Axioms in clause form:

1. *man(Marcus)*
2. *Pompeian(Marcus)*
3. ¬ *Pompeian*(x_1) v Roman(x_1)
4. *Ruler*(*Caesar*)
5. ¬ *Roman*(x_2) v *loyalto*(x_2, *Caesar*) v *hate*(x_2, *Caesar*)
6. *loyalto*(x_3, *f*1(x_3))
7. ¬ *man*(x_4) v ¬ *ruler*(y_1) v ¬ *tryassassinate*(x_4, y_1) v *loyalto* (x_4, y_1)
8. *tryassassinate*(*Marcus, Caesar*)

RESOLUTION IN PREDICATE LOGIC



DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in