Knowledge representations

Content

- Representation,Mapping and properties of KR
- Types of Knowledge
- Knowledge representation tools/techniques

Knowledge representation

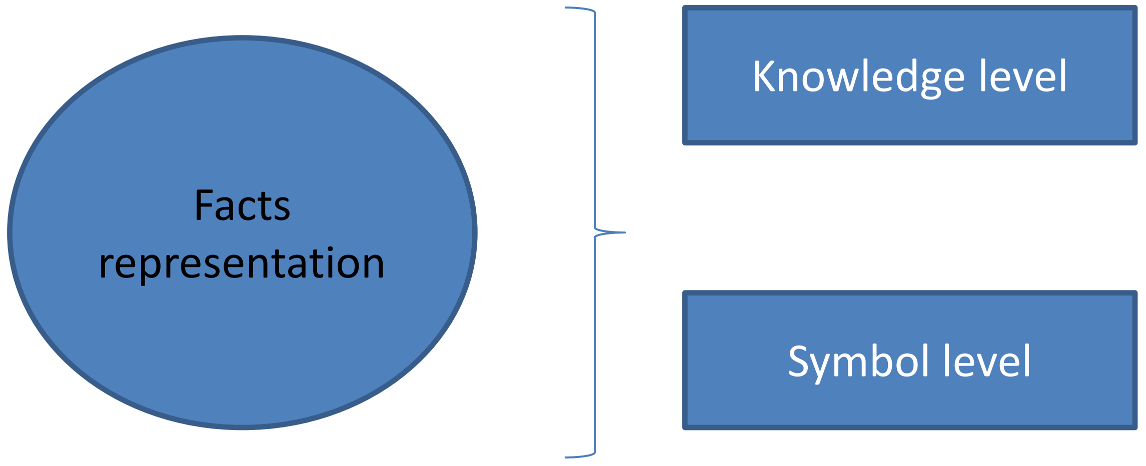
REPRESENTATION MAPPING AND PROPERTIES

Representations and mapping

- To solve complex problem
  - Large knowledge require
  - Manipulation of knowledge is also required
  - Variety of ways of Knowledge representation exist
- Two entities
  - **Facts :** things to be represented
  - Representation of facts in chosen formalism- used for manipulation

Representations and mapping

- This is structured as two levels

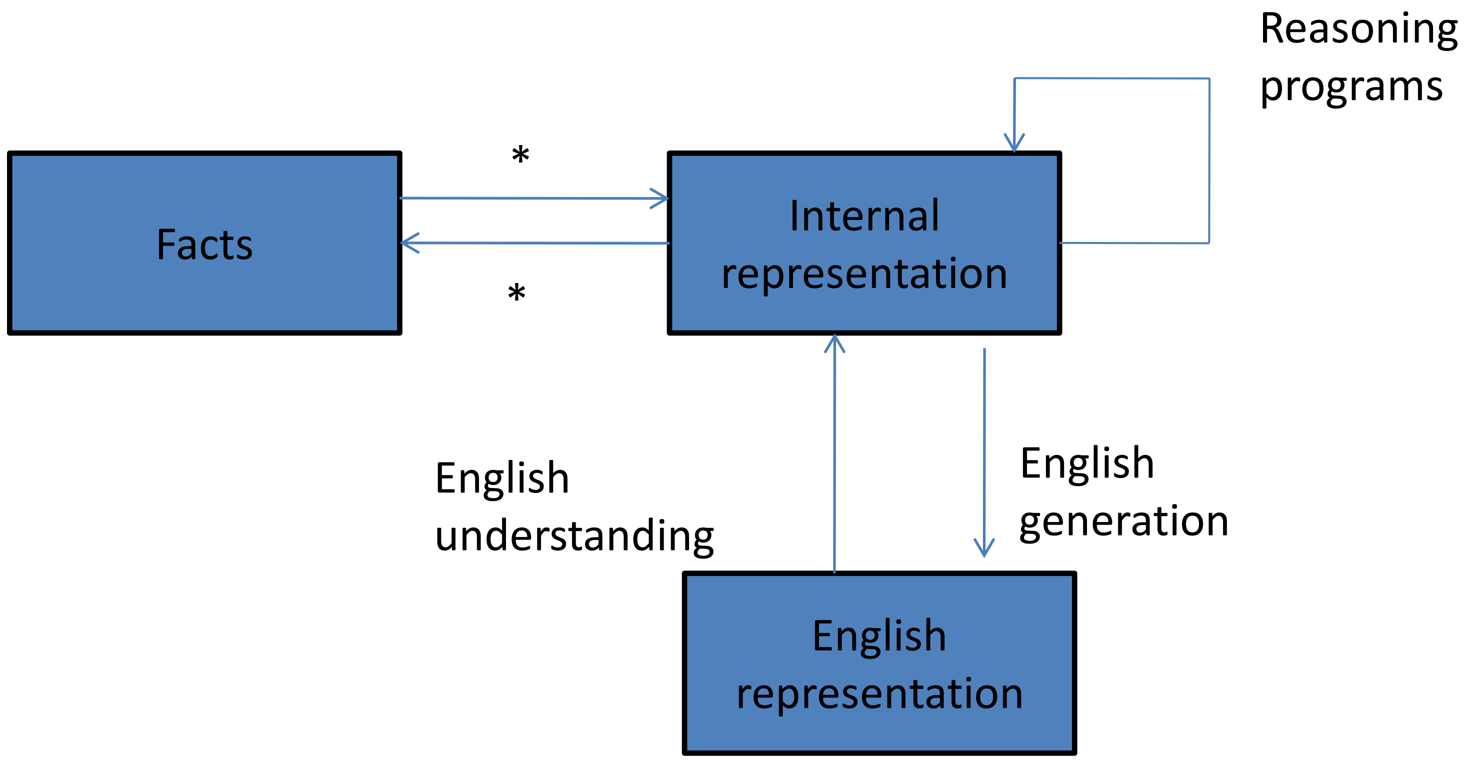


Facts (agent behaviour and current goal)is described

Representation of objects in form of symbol that can be used for manipulation

Newell model

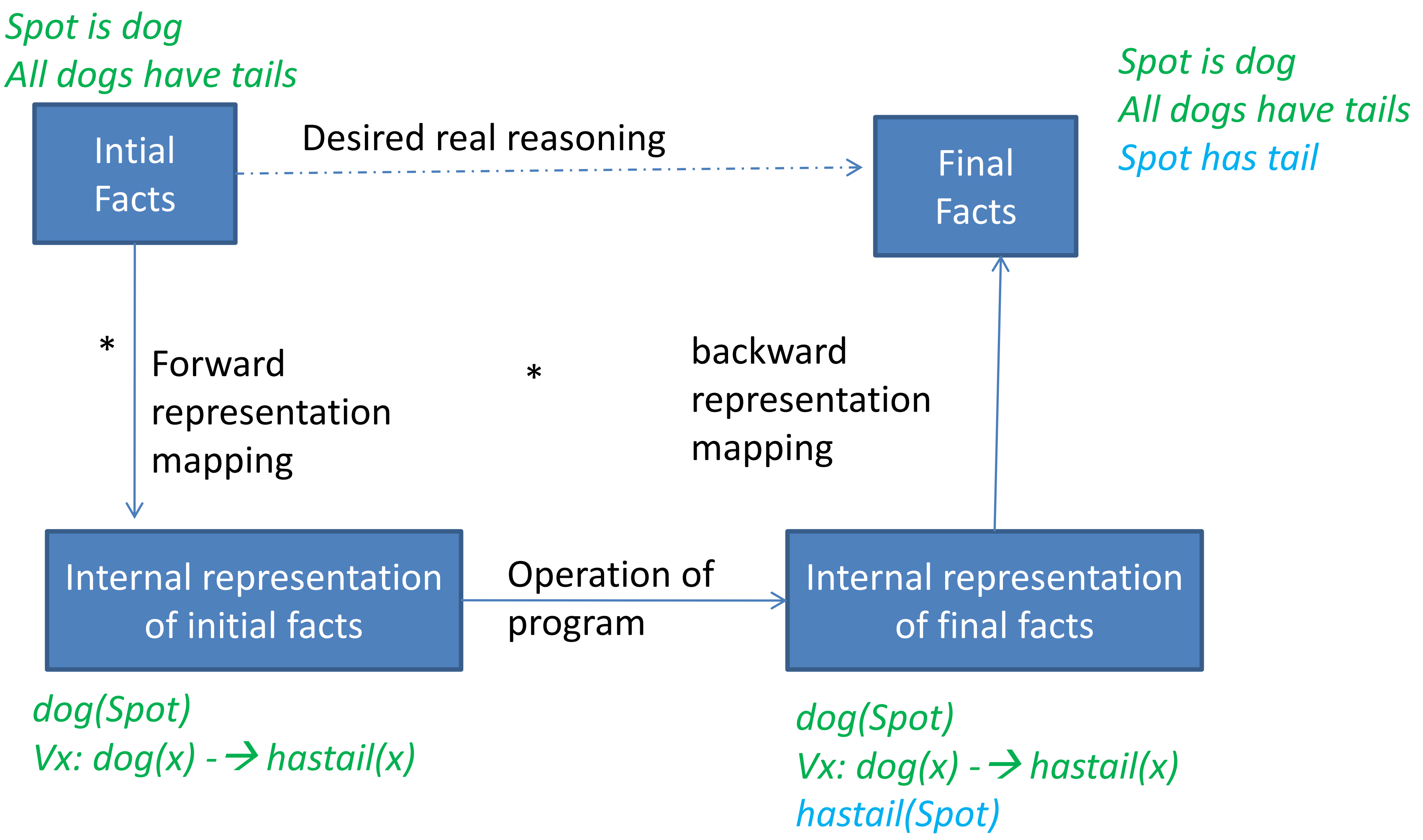
Representation mappings



Mapping between facts and representation
Two way mapping :

- From facts to internal representation…*forward*
- From internal representation to facts…*backward*
- *Represented by * in above diagram*

Representation of facts



Spot is dog
All dogs have tails
Intial Facts
Desired real reasoning
Final Facts
Spot is dog
All dogs have tails
Spot has tail
*
Forward representation mapping
*
backward representation mapping
Internal representation of initial facts
Operation of program
Internal representation of final facts
dog(Spot)
Vx: dog(x) -→ hastail(x)
dog(Spot)
Vx: dog(x) -→ hastail(x)
hastail(Spot)

Properties of KR

- **Representational adequacy** Ability to represent all kinds of knowledge that are needed in the domain
- **Inferential Adequacy** ability to manipulate the representational structures in such a way to derive new structures corresponding to new knowledge derived from old.
- **Inferential efficiency** ability to incorporate into the knowledge structure additional information that can focus the attention of inference mechanism in more promising direction
- **Acquisitional Efficiency**: ability to acquire new information easily. Either by direct insertion or by program itself

Relational knowledge

Inheritable

Inferential

Procedural/declarative

TYPES OF KNOWLEDGE

Types of knowledge

- Simple relational knowledge
  - Represented in tabular form
  - Simple
  - Weak inferential capabilities
  - Similar to databases

Player	Height	weight

Types of knowledge

- Inheritable knowledge
  - Uses objects , classes and attributes
  - Property inheritance
  - Slot and filler structure
  - Frames and semantic net

Types of knowledge

- Inferential knowledge
  - All properties or knowledge using logic
  - An efficient inferential procedure
  - Inference rules
- Procedural knowledge
  - Uses procedure
  - If the else constructs

Logic (Prepositional and Predicate)
Semantic Network
Frames
Scripts
Conceptual Dependencies

KNOWLEDGE REPRESENTATION TECHNIQUES

Knowledge representation mechanism

1. Logic
   - Propositional
     - Syntax and semantics of PL
     - Properties of sentence
     - Inference Rules
     - Formal System
   - Predicate
     - First Order Predicate Logic (FOPL)
     - Syntax and semantics of FOPL
     - Properties of well formed formulae(wff)
     - Inference mechanism
       - Backward chaining
       - Forward chaining
       - Resolution Principle in PL and FOPL
       - Unification Algorithm
       - Conversion to clausal form (Skolemnization)
       - Examples
       - Question-answer system using resolution
       - Advantages
       - Drawback
2. Semantic net
3. Frames
4. Scripts
5. Conceptual dependencies

LOGIC: Propositional logic

- Propositions are atomic sentences
- Syntax governs the combination of building blocks such as propositions and connectives
- In place of sentences term *formulae or well formed formulas (wff)* is used
- Connectives

  ($\sim$ $\vee$ & $\leftrightarrow$ $\rightarrow$)

  (not or and double implication implication

Syntax of PL

- Example
  - Its raining RAIN
  - Its windy WINDY
  - Its sunny SUNNY
  - Its windy but not sunny WINDY $\wedge \sim$ SUNNY
- Syntax
  - If P and Q are formulae then following are formula
    - ~P
    - (P&Q)
    - (PVQ)
    - (P → Q)
    - (P ←→ Q)

Semantics of PL

- Meaning of the sentence is just true or false
- Assignment of truth value to the sentence
- An interpretation for the sentence or group of sentence is an assignment of truth value to each propositional symbol
  - Example P&~Q
  - Interpretation

	P	Q	~Q	P&~Q
I1	T	T	F	F
I2	T	F	T	T

  - Once interpretation are given truth value of sentence can be determined

Example

- $((P \& \sim Q) \rightarrow R) V Q$
  - If P=t , Q= f , R=f
- Find truth value of sentence

Properties of sentence

- Satisfiable
  - A sentence is satisfiable if there is some interpretation for which it is true
- Contradiction
  - A sentence is contradictory (unsatisfiable) if there is no interpretation for which it is true
- Valid
  - A sentence is valid if it is true for every interpretation. They are also called tautologies
- Equivalence
  - Two sentences are equivalent if they have same truth value under every interpretation
- Logic Consequences
  - A sentence is logical consequence of other if it is satisfiable by all interpretations which satisfy the first

Examples

- A valid sentence is satisfiable?
- A contradictory statement is invalid
- P is satisfiable
- P V~P is Valid
- P&~P is contradictory
- P and ~(~P)
- P is logical consequence of (P&Q)

Example

- To prove P→Q is equivalent to ~PVQ

P	Q	~P	~PvQ	P→Q
T	T	F	T	T
T	F	F	F	F
F	T	T	T	T
F	F	T	T	T

Inference Rules

- Inference Rule
  - Provide means to perform logical proofs or deductions
  - Gives a set of sentence S={s1,s2,..........,sn} , prove the truth of s(conclusion)
  - Syntactic methods of inference of deduction can be used
  - They do not depend on truth assignment , but on syntactic relations

Inference rules

- Modus Pones Rule
  - From P and P→Q infer Q

Example	
given	*Joe is intelligent*
and	*Joe is intelligent -→Joe will pass exams*
Conclude	*Joe will pass exam*

- Chain Rule
  - From P→Q and Q→R infer P→R

Inference rules

- Substitution
  - If s is valid ,s’ derived from s by substitution of propositions in s , then s’ is also valid
  - PV~P is valid then QV~Q is also valid
- Simplification (And elimination)
  - From P&Q infer P
- Conjunction
  - From P and from Q infer P&Q (And Introduction)
- Transposition
  - From P$\rightarrow$Q , infer ~Q$\rightarrow$~P

Inference mechanism

- Backward chaining
  - Facts are given in form of rules
  - To prove a statement(goal) , match the goal in right side of rule in knowledge base
  - Unify the predicates by performing substitution during matching
  - repeat the process till all predicates in LHS match with given facts
- Forward chaining
- Resolution
  - To prove a fact , negation of fact is inserted into the system
  - Then given clauses are resolved
  - If contradiction is encountered then negation of fact is false
  - Fact is true

Formal system

- A formal system is a set of axioms S and a set of inference rules L from which new statements can be logically derived. System <S,L> is also referred as knowledge base(KB)
  - Soundness
    - If <S,L> is a formal system , we say inference procedure is sound if and only if any statement s can be from <S,L>
  - Completeness
    - Let <S,L> be a formal system , then inference procedure L is complete if and only if any sentence s is logically implied by<S,L> can be derived using that procedure

Predicate Logic
First Order Predicate Logic (FOPL)

- Syntax and semantics of FOPL
- Drawbacks of PL
    - Too “coarse” to describe properties of an object
    - Lack structure to represent relations among two objects
    - Does allow to construct generalized statements about classes of similar objects
    - Limitation while reasoning with the system
- FOPL
    - was developed to extend expressiveness of PL
    - Is a generalisation of PL that allows reasoning with world objects as entities , classes etc
    - Generalisation is possible by using predicates in place of proposition, use of variable , function etc

Syntax of FOPL

- Syntax of FOPL
  - Is determined by the allowable symbols and rules of combinations

- Semantics of FOPL
  - Are determined by the interpretation assigned to predicates instead of proposition
  - Interpretations are also assigned to constant , variables , function and arguments

Syntax of FOPL

- Connectives

  ~ & V → ↔
  NOT AND OR Implication double implications
- Quantifiers
  - Ǝ existential quantifier
    - $\exists x : P(x)$ should be true for atleast one value of x
  - V universal quantifier (for all)
    - $\forall x : Q(x)$ should be true for all values of x
- Constants
  - Fixed value term denoting number terms, words etc
- Variables
  - it can assume different values over given domain x,y,z

Syntax of FOPL

- Function
  - Function symbols denote relation define on a domain D
  - They map n element to single element
  - age-of(x) , F((t1,t2,t3......tn) where ti can be constant, variable and function
- Predicates
  - Predicate symbol denote the relations or functional mapping from elements of domain D to values true or false
  - P(t1,t2,------tn) can take constant , variable or function as their argument
  - 0-ary predicate is proposition or constant predicate
  - A predicate with no variables is called ground atom
- Constant , variables and function are called terms
- Predicate is referred as atom
- Also atom and its negation is called literal

First order predicate logic

- Why it is called first order ?
  - If quantification is on variables then its is first order
  - If quantification is on predicate and function then it is second order
- Interpretation of wff in FOPL
  - it refers to assignment of truth values to atoms in wff

Semantics of FOPL

- Wffs are implemented in domain D
- When assignment of values is given to each predicate and each symbol , we say interpretation is given to wff
- Vx: ((A(a,x) V B(f(x))) & C(x))→D(x)

  Given D={1,2}

  a=2

  f(1)=2 f(2)=1

  A(2,1)=true A(2,2)=false

  B(1)=true B(2)=false

  C(1)=true C(2)= false

  D(1)=false D(2)=true

  For x=1 find the truth value of sentence

  For x=2 find the truth value of sentence

Properties of wff

- Valid
- Inconsistent (un satisfiable/contradictory)
- Satisfiable
- Equivalence
- Logical consequence

Properties of wff

- Equivalent logical expression
  - $\sim(\sim F)=F$
  - $F\&G=G\&F$ Commutative
  - $(F\&G)\&H=F\&(G\&H)$ Associative
  - $F\&(GVH)=(F\&G)V(F\&H)$ distributive
  - $\sim(F\&G)=\sim FV\sim G$ De Morgan’s law
  - $F\rightarrow G=\sim FVG$
  - $F\leftarrow\rightarrow G=(\sim FVG)\&(\sim GVF)$
  - $\sim(Vx)F(x)=\exists x \sim F(x)$

FOPL

- *Bounded and free variable*
  - A variable of predicate is **bounded** if it is associated with quantifiers , otherwise it is termed as **free variable**
  - Vx (P(x) -→ Q(x,y))
  - x is bounded and y is free variable
- CNF and DNF
  - Given F1, F2, F3, -----Fn as wff containing disjunct of literals only then
    - F1&F2&F3------Fn is *conjuctive normal form (CNF)*
  - Given F1, F2, F3, -----Fn as wff containing conjunct of literals only then
    - F1 V F2 V F3------Fn is *disjunct normal form (DNF)*

Inferencing (Reasoning) Mechanism

- Inferencing can be done in two ways
- Backward chaining
  - Statement of the form A(x) & B(y) $\rightarrow$C(x,y) is used
  - Where A(x) and B(y) are independent sub goals
  - Substitutions are used while proving the result
  - Proof ends in nil , no more sub goals left for proving
  - Implement the rules from right and try to prove left part
- Resolution Principle

Example-1

1. Marcus was man

   man(Marcus)

2. Marcus was Pompeian

   pompeian(Marcus)

3. All Pompeian's were Roman

   (∀x) pompeian(x) → roman(x)

4. Caesar was ruler

   ruler(caesar)

5. All Romans were either loyal to Caesar or hated him

   (∀x) roman(x) Λ ~loyalto(x,Caesar) → hate(x,Caesar)

6. Everyone is loyal to someone

   (∀x)(∃y) loyalto(x,y)

7. People try to assassinate rulers they are not loyal to.)

   ∀x ∀y person(x) Λ ruler(y) Λ tryassasinate (x,y) → ~loyalto(x,y)

8. Marcus tried to assassinate Caesar

   tryassasinate(Marcus,Caesar)

9, All men are people

   (∀x) man(x)→ person(x)

Example-1

1. Marcus was man.
2. Marcus was Pompeian
3. All Pompeian's were Roman
4. Caesar was ruler
5. All Romans were either loyal to Caesar or hated him
6. Everyone is loyal to someone
7. People try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar
9. All men are people

Backward chaining

To prove ~loyalto(Marcus,Caesar)

$$\begin{tikzcd}
\text{\textasciitilde loyalto(Marcus,Caesar)} \arrow[d, "\text{7, x|marcus , y|caesar}"] \\
\text{person(Marcus)} \land \text{ruler(caesar)} \land \text{tryassasinate (Marcus,Caesar)} \arrow[d, "4"] \\
\text{person(Marcus)} \land \text{tryassasinate (Marcus,Caesar)} \arrow[d, "8"] \\
\text{person(Marcus)} \arrow[d, "\text{9 x| Marcus}"] \\
\text{Man(Marcus)} \arrow[d, "1"] \\
\textbf{nil}
\end{tikzcd}$$

Example 2

1. Marcus was man
2. Marcus was Pompeian
3. Marcus was born in 40 AD
4. All men are mortal
5. All Pompeian died when volcano erupted in 79 AD
6. No mortal lives longer than 150 years
7. It is now 1991
8. Alive means not dead
9. If someone dies then he is dead for all times

Example 2

1. Marcus was man
   - Man(Marcus)
2. Marcus was Pompeian
   - Pompeian(Marcus)
3. Marcus was born in 40 AD
   - born(Marcus,40)
4. All men are mortal
   - Vx man(x) → mortal(x)
5. All Pompeian died when volcano erupted in 79 AD
   - erupted (volcano,79) & [Vx : Pompeian (x)→dead(x,79)]
   - 5(a) erupted (volcano,79)
   - 5(b) [Vx : Pompeian(x)→dead(x,79)]

Example 2

6. No mortal lives longer than 150 years
   - mortal(x) & born(x,t1) & gt(t2-t1,150) → dead(x,t2)
7. It is now 2015
   - Now=2015
8. Alive means not dead
   - alive(x,t)→~dead(x,t)
   - dead(x,t)→ ~alive(x,t)   [Transposition]
9. If someone dies then he is dead for all times
   - Vx: dead(x,t1) & gt(t2,t1)→ dead(x,t2)

Backward chaining

Prove ~alive(marcus,now)

$$\begin{tikzcd}
\text{\textasciitilde alive(marcus,now)} \arrow[d, "8(b)"] \\
\text{Dead(marcus,now)} \arrow[d, "6"] \\
\text{Mortal(Marcus) \& born(Marcus,t1)\& gt(now –t1,150)} \arrow[d, "3"] \\
\text{Mortal(Marcus) \& gt(now –40,150)} \arrow[d] \\
\text{Mortal(Marcus)} \arrow[d, "4"] \\
\text{man(Marcus)} \arrow[d, "1"] \\
\text{nil}
\end{tikzcd}$$

$$\begin{tikzcd}
\text{\textasciitilde alive(marcus,now)} \arrow[d, "8(b)"] \\
\text{Dead(marcus,now)} \arrow[d, "9"] \\
\text{dead(marcus,t1) \& gt(now,t1)} \arrow[d, "5(b)"] \\
\text{Pompeian(Marcus) \& gt(2015,79)} \arrow[d, "2"] \\
\text{nil}
\end{tikzcd}$$

Backward chaining

- Drawbacks of backward chaining
  - Even simple conclusion require many steps
  - Many processes like *matching*, *substitution*, *modus ponens* are required in production of proof
  - Statement with implications having more than one term on right becomes complex to use or having terms with *and's* or *or's* on left side
  - It is applicable to only Horn clauses (*clause with at least one positive literal)*

1. John likes all kinds of food

$$\forall x: food(x) \rightarrow likes(John, x)$$

2. Apple is a food 3. Chicken is a food

$$food(Apple) \qquad food(Chicken)$$

4. Anything anyone eat and is'nt killed by is a food

$$\forall x \forall y: eat(x, y) \wedge \sim killed(x) \rightarrow food(y)$$

5. Bill eats peanuts and is still alive

$$eats(Bill, peanuts) \wedge \sim killed(Bill)$$
$$5(a) eats(Bill, peanuts)$$
$$5(b) \sim killed(Bill)$$

6. Sue eats everything Bill eats

$$\forall x: eats(Bill, x) \rightarrow eats(Sue, x)$$

To Prove : **like (John,peanuts)**

Forward chaining

$$eats(Bill, peanuts) and \sim killed(Bill)$$

$$\begin{tikzcd} eats(Bill, peanuts) \wedge \sim killed(Bill) \arrow[d, "\text{Using 4 x|Bill y|peanuts}"] \\ food(peanut) \arrow[d, "\text{Using 1 x|peanuts}"] \\ likes(John, peanuts) \end{tikzcd}$$

Basis of Resolution

- From computational perspective it would be better to have proof procedure that uses various process in a single operation

- Resolution is such a procedure, but it requires that statement should be converted into simple standard form called **clausal form**

Basis of resolution

- Simple iterative process
- At each step two clauses called parent clause are compared (resolved)
- Yielding a new clause (inferred clause)
- Which represent the way two parent clause interact with each other
- **Winter** V **Summer** **~Winter** V **Cold**
- Both the clauses must be true
- Out of **~winter** and **winte**r only one can be true
- If **winte**r =true then **cold** must be true to guarantee the truth of second clause
- If **winter** is false then **summe**r must be true

Basis of resolution

$$\begin{tikzcd}[column sep=small] & \text{winter V summer} \arrow[dr] & & \sim\text{winter V cold} \arrow[dl] \\ \textit{Resolvent}\text{----} & & \text{summer V cold} & \end{tikzcd}$$

- This deduction is made by **resolution principle**
- Resolution takes two clauses such that each contain same literal
- In positive form in one clause and negative form in other
- Resolvent is obtained by combining all the literal of two parent clause except the one that cancel out
- If resolvent is empty clause then **contradiction** has been found

Resolution

- It produces proof by refutation
- To prove a statement (show that it is valid)
- Resolution attempts to show that negation of that statement produces contradiction with known statements.
- But it requires FOPL statement to be presented in simple standard form called clausal from which do not include
  - Universal quantifier
  - Implications
- All facts are converted into Conjective Normal Form
- Facts in CNF are then broken into individual clauses

Clause: is a wff in CNF with no instances of connector &

Resolution in propositional logic

1. Convert all proposition to clausal form
2. Negate P and convert the result to clausal form
3. Repeat until a contradiction is found or no progress can be made
   a. Select two clauses- **parent clause**
   b. Resolve them together
      I. Resolvent will be disjunct of literals of parent clause
      II. With exceptions that is there are any pair of literals L and ~L
      III. Such that one parent contain L and other contain ~L
      IV. Select one such pair and eliminate both from resolvent
   c. If resolvent = empty ; contradiction has been found
   d. If not, then add resolvent to set of clauses

Example

- Given
  - P
  - (P&Q)→R
  - (SVT)→Q
  - T
- Convert them into clausal form
- Prove R by resolution

Unification algorithm

- In PL it is easy to determine that two literals cannot be true at the same time
- In FOPL it is more difficult perform this match
- As arguments should be considered during matching process
- Example
  - *man(John)* and *~man(John)* is a contradiction
  - *Man(John)* and *~man(Spot) is not*
- Hence in order to determine contradiction we need matching process that compare two literals and discovers whether there exist a substitution that makes them identical
- Recursive process which performs this is called **unification algorithm**

Unification algorithm

- Following checks are made
  - Initial predicate symbol are same
  - Check the arguments , one pair at a time
  - To test each pair we can call unification alg recursively
- Problem
  - Find substitution list for
  - P(x,x) and P (y,z)
  - Solution (z/y,y/x)

Unification algorithm

- If L1 and L2 are variable or constant then
  - If L1 and L2 are identical return Nil
  - If L1 is a variable and L1 occurs in L2 return fail else return (L2/L1)
  - If L2 is a variable and L2 occurs in L1 return fail else return (L1/L2)
  - Else return fail
- If initial predicate symbol is not identical return fail
- If L1 and L2 have different number of arguments the return fail
- Set SUBST=Nil
- For i= 1 to no of arguments in L1
  - Call unify with i^{th} argument of L1 and L2, putting result in S
  - If S contains FAIL return FAIL
  - IF S is not equal to Nil
    - Apply S to remainder of L1 and L2
    - SUBST:=Append(S,SUBST)
- Return SUBST

Resolution in FOPL

1. Convert all statement to clausal form
2. Negate P and convert the result to clausal form. Add it to set of clauses obtained in 1
3. Repeat until a contradiction is found or no progress can be made
    a. Select two clauses - **parent clause**
    b. Resolve them together
        I. Resolvent is disjunct of literals of parent clauses
        II. With proper substitution and with one exception
            i. That if a pair T and ~T exist in such a way that T is in one parent and ~T is in another
            ii. T and ~T are unifiable then neither T and ~T should appear in resolvent
            iii. T and ~T are called complementary literal
            iv. Use substitution produced by unification algorithm to create resolvent
            v. Remove only one pair of complementary literals
    c. If resolvent is empty contradiction has been found, if not add it to the set of clauses

Resolution in FOPL

- Choices of clauses should be systematic
  - Resolve pair of clauses that contain complementary literals
  - Eliminate certain clauses as soon as they are generated. Two types can be eliminated
    - Tautologies (which can never be satisfied)
    - Clauses that are subsumed by other
      - P V Q is subsumed by P
  - ***Set_of_support strategy :*** Resolve with one of the statement we are trying to prove or by clause generated by resolution with such a clause.
  - ***Unit_preference strategy :***Resolve with clause having single literal. It generated clauses with fewer literals. Closer to goal of resolvent with zero term.

Conversion to Clausal Form

1. Remove implications :
   - Eliminate all <=> connectives by replacing each instance of the form (P <=> Q) by the equivalent expression ((P => Q) ^ (Q => P))
   - Eliminate all => connectives by replacing each instance of the form (P => Q) by (~P v Q)

2. Move ~ negation to innermost atom
   - Reduce the scope of each negation symbol to a single predicate by applying equivalences such as converting
   - ~(~P to P); ~(P v Q) to ~P ^ ~Q;
     ~(P ^ Q) to ~P v ~Q, $\sim(\forall x)\boldsymbol{P}\ \textbf{to}\ \exists \boldsymbol{x} \sim \boldsymbol{P}$ and $\sim(\exists x)P\ \ to\ \ \forall x\sim P$

3. Rename variable so that every bounded variable is unique
   - Standardize variables: rename all variables so that each quantifier has its own unique variable name. For example,

$$\forall x\ P(x)\ \ \forall x Q(x)\ with\ \ \forall x_1\ P(x_1)\ \ \forall x_2\ Q(x_2)$$

Conversion to Clausal Form

4. Move all quantifiers to left of wff (prenex normal form

5. Remove existential quantifiers(**Skolemnisation**)
   - Eliminate existential quantification by introducing **Skolem functions**.
   - For example, convert $(\exists x)P(x)$ to $P(c)$ where c is a brand new constant symbol that is not used in any other sentence. c is called a **Skolem constant**.
   - More generally, if the existential quantifier is within the scope of a universal quantified variable, then introduce a Skolem function that depend on the universally quantified variable.
   - For example, $(\forall x)(\exists y)P(x,y)$ is converted to $(\forall x)P(x, f(x))$.
   - f is called a **Skolem function**, and must be a brand new function name that does not occur in any other sentence in the entire KB.

Conversion to Clausal Form

6. Drop the prefix
   - Remove universal quantification symbols by first moving them all to the left end and making the scope of each the entire sentence, and then just dropping the "prefix" part. For example, convert ($\forall$ x)P(x) to P(x).

7. Convert into CNF
   - Distribute "and" over "or" to get a conjunction of disjunctions called **conjunctive normal form**. Convert (P ^ Q) v R to (P v R) ^ (Q v R), and convert (P v Q) v R to (P v Q v R).

8. Break into independent clause
   - Split each conjunct into a separate **clause**, which is just a disjunction ("or") of negated and un-negated predicates, called **literals**.

9. Standardize variable
   - Standardize variables apart again so that each clause contains variable names that do not occur in any other clause.

Clausal form

- Expression becomes simple
  - If there are less embedding of components
  - Quantifiers are removed
- Example " *All Romans who know Marcus either hate Ceasar or think that anyone who hates anyone is crazy"*
- FOPL:
  - (All romans) Λ (who knew Marcus) → (they hate Ceasar) V (they think y is crazy if y hates someone)

  - ∀x:[roman(x) Λ know(x,Marcus)] → [hate(x,ceasar) V (∀ y∃z: hate(y,z)→ thinkcrazy(x,y))]
  - ~roman(x) V ~know(x,Marcus) V hate(x,Ceasar) V ~hate(y,$\overset{f(x,y)}{\cancel{z}}$) V thinkcrazy(x,y)
- **Which is simpler?**

Example-1

1. Marcus was man
2. Marcus was Pompeian
3. All Pompeian's were Roman
4. Caesar was ruler
5. All Romans were either loyal to Caesar or hated him
6. Everyone is loyal to someone
7. People try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar
9. All men are people

Example-1

1. Marcus was man

   man(Marcus)

2. Marcus was Pompeian

   pompeian(Marcus)

3. All Pompeian's were Roman

   $(\forall x)$ pompeian(x) $\rightarrow$ roman(x)

4. Caesar was ruler

   ruler(caesar)

5. All Romans were either loyal to Caesar or hated him

   $(\forall x)$ roman(x) $\rightarrow$ loyalto(x,Caesar) $\vee$ hate(x,Caesar)

6. Everyone is loyal to someone

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

7. People try to assassinate rulers they are not loyal to.

   $(\forall y)$ person(x) $\wedge$ ruler(y) $\wedge$ tryassasinate(x,y) $\rightarrow$ ~loyalto(x,y)

8. Marcus tried to assassinate Caesar

   tryassasinate(Marcus,Caesar)

9, All men are people

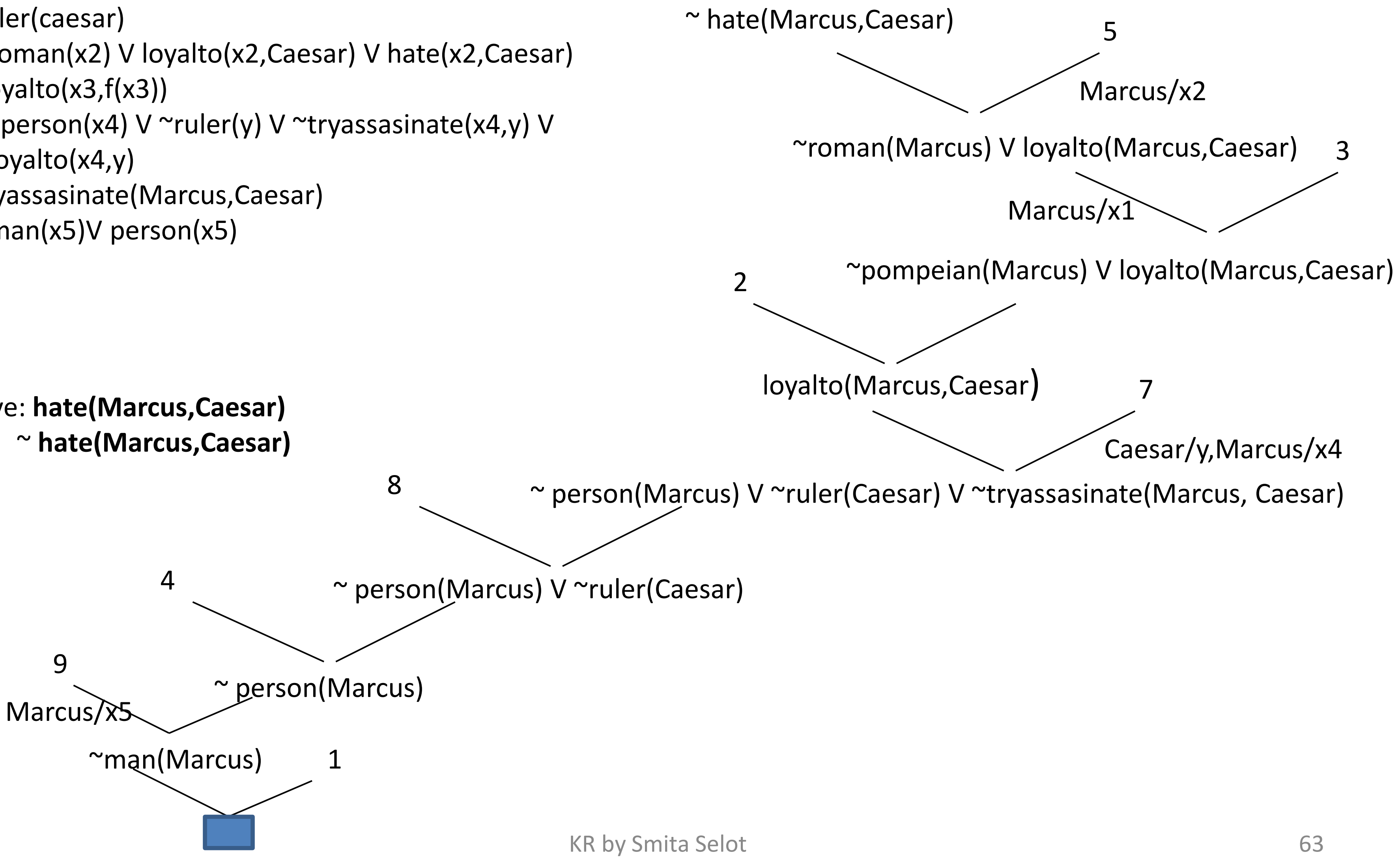
   $(\forall x)$ man(x)$\rightarrow$ person(x)

Example 1

1. man(Marcus)
2. pompeian(Marcus)
3. ~pompeian(x1) V roman(x1)
4. ruler(caesar)
5. ~roman(x2) V loyalto(x2,Caesar) V hate(x2,Caesar)
6. loyalto(x3,f(x3))
7. ~ person(x4) V ~ruler(y) V ~tryassasinate(x4,y) V ~loyalto(x4,y)
8. tryassasinate(Marcus,Caesar)
9. ~man(x5)V person(x5)

1.

To Prove: **hate(Marcus,Caesar)**
Add ~ **hate(Marcus,Caesar)**



~ hate(Marcus,Caesar)
5
Marcus/x2
~roman(Marcus) V loyalto(Marcus,Caesar)
3
Marcus/x1
~pompeian(Marcus) V loyalto(Marcus,Caesar)
2
loyalto(Marcus,Caesar)
7
Caesar/y,Marcus/x4
~ person(Marcus) V ~ruler(Caesar) V ~tryassasinate(Marcus, Caesar)
8
~ person(Marcus) V ~ruler(Caesar)
4
~ person(Marcus)
9
Marcus/x5
~man(Marcus)
1

Example 2

1. Marcus was man
2. Marcus was Pompeian
3. Marcus was born in 40 AD
4. All men are mortal
5. All Pompeian died when volcano erupted in 79 AD
6. No mortal lives longer than 150 years
7. It is now 1991
8. Alive means not dead
9. If someone dies then he is dead for all times

Convert them to clausal form

1. Man(Marcus)
2. Pompeian(Marcus)
3. born(Marcus,40)
4. Vx man(x) → mortal(x)
5. erupted (volcano,79) & [Vx : Pompeian (x)→dead(x,79)]
   a) erupted (volcano,79)
   b) Vx : Pompeian(x)→dead(x,79)]
6. mortal(x) & born(x,t1) & gt(t2-t1,150) → dead(x,t2)
7. Now=2015
   a) alive(x,t)→~dead(x,t)
   b) dead(x,t)→ ~alive(x,t)   [Transposition]
8. Vx: dead(x,t1) & gt(t2,t1)→ dead(x,t2)

1. Man(Marcus)
2. Pompeian(Marcus)
3. born(Marcus,40)
4. ~ man(x1) Vmortal(x1)
5. erupted (volcano,79)
6. ~Pompeian(x2)Vdead(x2,79)
7. ~mortal(x) V~ born(x,t1) V~ gt(t2-t1,150) V dead(x3,t2)
8. Now=2015
9. ~alive(x4,t3)Vdead(x4,t3)
10. ~dead(x5,t4)V ~alive(x5,t4)
11. ~dead(x6,t5) V ~gt(t2,t5)V dead(x6,t6)

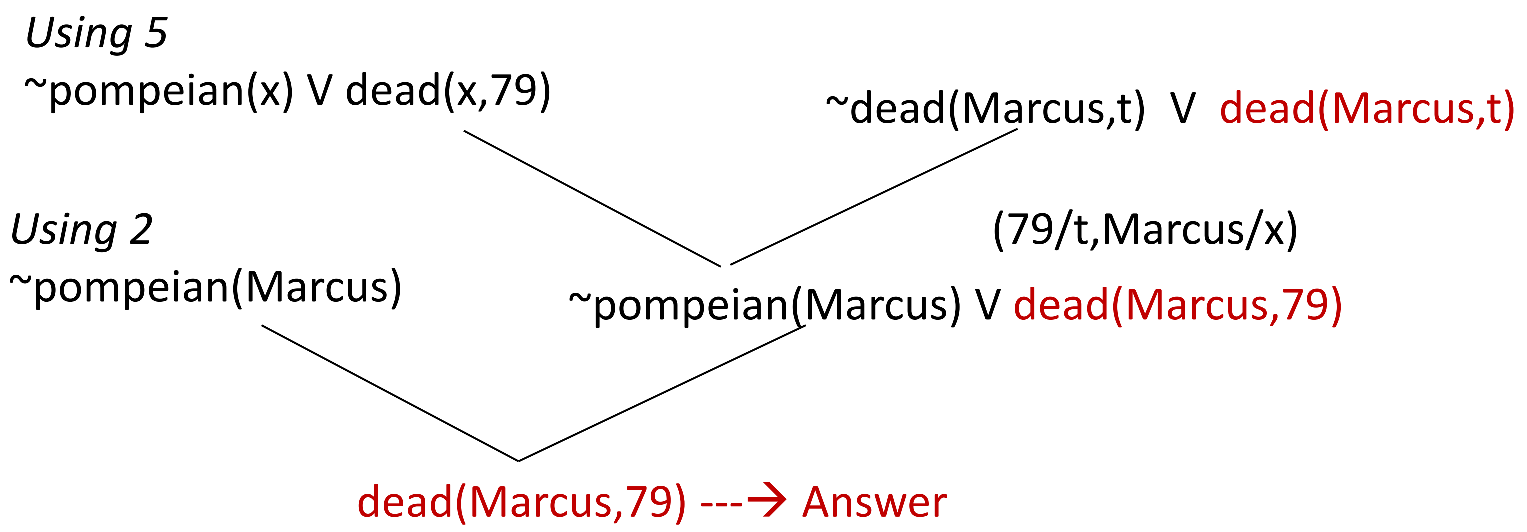
Prove ~alive(marcus,now)

$$\begin{tikzcd}
 & 10 \arrow[dr] & & \text{alive(marcus,now)} \arrow[dl, "\text{Marcus/x5 now/t4}"] \\
7 \arrow[dr] & & \sim\text{dead(Marcus,ow)} \arrow[dl] & \\
 & {} & &
\end{tikzcd}$$

Complete the proof

Answer extraction using Resolution

- Can be used to deduce a fact
- Can be used to answer a question
- Example
  - When did Marcus died?
  - dead(Marcus,t)   Use example 2
  - Add dead(Marcus,t) along with dead(Marcus,t)


Using 5
~pompeian(x) V dead(x,79)
~dead(Marcus,t) V dead(Marcus,t)
Using 2
~pompeian(Marcus)
(79/t,Marcus/x)
~pompeian(Marcus) V dead(Marcus,79)
dead(Marcus,79) ---→ Answer

Monkey banana problem

1. In_room(Banana)
2. In_room(Chair)
3. In_room(Monkey)
4. Dexterous (Monkey)
5. Tall(Chair)
6. ~close(Banana,floor)
7. Can_move(Monkey,Chair,Banana)
8. Can_climb(Monkey,Chair)
9. Dexterous(X) & close(x,y) → can-reach(x,y)
10. Get_on(x,y)&under(y,Banana)& tall(y)→ close(x,Banana)
11. In_room(x) & In_room(y) & In_room(z) & can_move(x,y,z) →close(z,floor)Vunder(y,z)
12. Can-climb(x,y) → get_on(x,y)

To prove can_reach(Monkey,Banana)

Advantages

- Captures generalization by allowing quantifiers
- It is **declarative** and **inferential** in nature
- Relationship amongst clauses (facts) are given in form of → and ←→
- Powerful way of deriving new knowledge as new facts are inferred from the given one by using strong inferential mechanism
  - Resolution and backward chaining
- Able to provide answer to question

Drawbacks of resolution

- When it will stop? If a contradiction is found.
- If contradiction is not found? Resolution proof is not **complete.** Needs an additional mechanism to implement completeness concept in resolution
- Inheritable knowledge is weakly captured
- Use of is-a or instance predicate allow inheritance but cannot give specific value under exceptions
- It cannot capture information like
  - *" It is very hot today"*—relative degrees of heat
  - *"Blond-haired people often have blue eyes"*—amount of certainty
  - *"I know Bill think Giants will win but I think they will loose"* -- How to represent several beliefs