UNIT 5
Reasoning

Knowledge Representation

- When we use search to solve a problem we must
  - Capture the knowledge needed to formalize the problem
  - Apply a search technique to solve problem
  - Execute the problem solution

Role of KR

- The first step is the role of “knowledge representation” in AI.
- Formally,
  - The intended role of knowledge representation in artificial intelligence is to reduce problems of intelligent action to search problems.

Limitations of Propositional Logic

- Propositional logic cannot express general-purpose knowledge succinctly
- We need 32 sentences to describe the relationship between wumpi and stenches
- We would need another 32 sentences for pits and breezes
- We would need at least 64 sentences to describe the effects of actions
- How would we express the fact that there is only one wumpus?
- Difficult to identify specific individuals (Mary, among 3)
- Generalizations, patterns, regularities difficult to represent (all triangles have 3 sides)

First-Order Predicate Calculus

- Propositional Logic uses only propositions (symbols representing facts), only possible values are True and False
- First-Order Logic includes:
  - Objects: peoples, numbers, places, ideas (atoms)
  - Relations: relationships between objects (predicates, T/F value)
    - Example: father(fred, mary)
    - Properties: properties of atoms (predicates, T/F value)
      Example: red(ball)
  - Functions: father-of(mary), next(3), (any value in range)
    - Constant: function with no parameters, MARY

Example

- Express “Socrates is a man” in
- Propositional logic
  - MANSOCRATES - single proposition representing entire idea
- First-Order Predicate Calculus
  - Man(SOCRATES) - predicate representing property of constant SOCRATES

DeMorgan Rules

-
- $\forall x \neg P \leftrightarrow \neg \exists x P$
- $\forall x P \leftrightarrow \neg \exists x \neg P$
- $\neg \forall x \neg P \leftrightarrow \exists x P$
- Example

$\neg \forall x P \leftrightarrow \exists x \neg P$

$$\forall x Loves Watermelon(x) \leftrightarrow \neg \exists x \neg Loves Watermelon(x)$$

Rules of Inference for Predicate Logic

- Modus ponens $\frac{A, A \rightarrow B}{B}$

  All men are mortal (Man -> Mortal)
  Socrates is a man (Man)

  -
  Socrates is mortal (Mortal)
- And introduction

  $\frac{A, B}{A \wedge B}$
- Or introduction $\frac{A}{A \vee B \vee C \vee D \vee \ldots}$
- And elimination $\frac{A \wedge B \wedge C \wedge \ldots \wedge Z}{A}$

- Double-negation elimination $\frac{--A}{A}$
- Unit resolution

  $\frac{A \vee B, -B}{A}$

  Today is Tuesday or Thursday
  Today is not Thursday

  Today is Tuesday
- Resolution

  $\frac{A \vee B, -B \vee C}{A \vee C}$

  Today is Tuesday or Thursday
  Today is not Thursday or tomorrow is Friday

  Today is Tuesday or tomorrow is Friday

Inference in First-Order Logic: First-Order Deduction

- Want to be able to draw logically sound conclusions from a knowledge-base expressed in first-order logic.

•Several styles of inference:

-Forward chaining

-Backward chaining

-Resolution refutation

•Properties of inference procedures (Entail):

-Soundness: If A |- B ("If A, then B" asserts that if A is true, then B must be true also), then A |= B (A is equal to B)

-Completeness: If A |= B (A is equal to B), then A |- B ("If A, then B" asserts that if A is true, then B must be true also)

•Forward and Backward chaining are sound and can be reasonably efficient but are incomplete.

•Resolution is sound and complete for FOPC but can be very inefficient.

Inference Rules for Quantifiers

Let SUBST(θ, α) denote the result of applying a substitution or binding list θ to the sentence α.
SUBST({x/Tom, y,/Fred}, Uncle(x,y)) = Uncle(Tom, Fred)

Inference rules

Universal Elimination: ∀*v* α |- SUBST({*v*/*g*},α)
for any sentence α, variable *v*, and ground term *g*
∀x Loves(x, FOPC) |- Loves(Ray, FOPC)

Existential Elimination: ∃*v* α |- SUBST({*v*/*k*},α)
for any sentence α, variable *v*, and constant symbol *k,*
that doesn't occur elsewhere in the KB (**Skolem constant**)
∃x (Owns(Mary ,x) ∧ Cat(x)) |- Owns(Mary, MarysCat) ∧
Cat(MarysCat)

–**Existential Introduction**: α |- ∃v SUBST({*g*/*v*},α)
for any sentence α, variable,*v*, that does not occur in α, and ground term *g*, that does occur in α
Loves(Ray, FOPC) |- ∃x Loves(x, FOPC)

Inference Rules for Quantifiers

1) $\forall x,y(\text{ Parent }(x,y) \wedge \text{ Male }(x) \Rightarrow \text{ Father }(x,y))$

2) Parent(Tom, John)

3) Male(Tom)

Using Universal Elimination from 1)

4) $\forall y(\text{Parent(Tom, } y) \wedge \text{Male(Tom)} \Rightarrow \text{Father(Tom, } y))$

Using Universal Elimination from 4)

5) $\text{Parent(Tom, John)} \wedge \text{Male(Tom)} \Rightarrow \text{father (Tom, John)}$

Using And Introduction from 2) and 3)

6) $\text{Parent(Tom, John)} \wedge \text{Male(Tom)}$

Using Modes Ponens from 5) and 6)

7) Father(Tom, John)

Generalized Modus Ponens

- Combines three steps of “natural deduction”
- (Universal Elimination, And Introduction, Modus Ponens) into one.
- Provides direction and simplification to the proof process for standard inferences.

Generalized Modus Ponens:

$p_1', p_2', \ldots p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q) \vdash \mathrm{SUBST}(\theta, q)$

where θ is a substitution such that for all *i*
$\mathrm{SUBST}(\theta, p_i') = \mathrm{SUBST}(\theta, p_i)$

1) ∀x,y(Parent(x,y) ∧ Male(x) ⇒ Father(x,y))

2) Parent(Tom,John)

3) Male(Tom)

θ={x/Tom, y/John)

4) Father(Tom,John)

Propositional vs. Predicate Logic

- In propositional logic, each possible atomic fact requires a separate unique propositional symbol.
- If there are *n* people and *m* locations: representing the fact that some person moved from one location to another requires nm^2 separate symbols.
- Predicate logic includes a richer **ontology.**
- **Ontology:** A rigorous and exhaustive organization of some knowledge domain that is usually hierarchical and contains all the relevant entities and their relations.

Unification

- In order to match antecedents /premise to existing literals in the KB, need a pattern matching routine.
- UNIFY(p, q) takes two atomic sentences and returns a substitution that makes them equivalent.
- UNIFY(p, q)=θ where SUBST(θ, p)=SUBST(θ, q)

  θ is called a **unifier**.
- Examples
- UNIFY(Parent(x,y), Parent(Tom, John)) = {x/Tom, y/John}
- UNIFY(Parent(Tom, x), Parent(Tom, John)) = {x/John})
- UNIFY(Likes(x,y), Likes(z, FOPC)) = {x/z, y/FOPC}
- UNIFY(Likes(Tom, y), Likes(z,FOPC)) = {z/Tom, y/FOPC}
- UNIFY(Likes(Tom, y), Likes(y, FOPC)) = fail
- UNIFY(Likes(Tom, Tom), Likes(x, x)) = {x/Tom}
- UNIFY(Likes(Tom, Fred), Likes(x, x)) = fail

Unification

- Exact variable names used in sentences in the KB should not matter.
- But if Likes(x, FOPC) is a formula in the KB, it does not unify with Likes(John, x) but does unify with Likes(John, y).
- To avoid such conflicts, one can **standardize apart** one of the arguments to UNIFY to make its variables unique by renaming them.

  Likes(x ,FOPC) -> Likes(x_1, FOPC)
- UNIFY(Likes(John ,x), Likes(x_1,FOPC)) = {x/John, x/FOPC}
- There are many possible unifiers for some atomic sentences.
- UNIFY(Likes(x,y),Likes(z,FOPC)) = {x/z, y/FOPC}

  {x/John, z/John, y/FOPC}

  {x/Fred, z/Fred, y/FOPC} ......
- UNIFY should return the **most general unifier** which makes the least commitment to variable values.

Unification

We sometimes want to “match” statements

- -dog(?x) v feathers(?x)
- Feathers(Tweety)
- Dog(Rufus)
- (?x Tweety)

The match needs to be consistent

During the match we build a binding list

Example

- -hold(P1, ?card) ^ -hold(P2, ?card) ^ -hold(P3, ?card) -> solution(?card)
- -hold(P1, Rope) ^ -hold(P2, Rope) ^ -hold(P3, Rope)

If we substitute ?card with Rope everywhere (?card Rope) then statement is equivalent to the left-hand side of the rule

Two expressions are unifiable iff there exists a substitution list (binding list) that, when applied to both expressions, makes them the same

Unification

We sometimes want to "match" statements

- -dog(?x) v feathers(?x)
- Feathers(Tweety)
- Dog(Rufus)
- (?x Tweety)

The match needs to be consistent

During the match we build a binding list

Example

- -hold(P1, ?card) ^ -hold(P2, ?card) ^ -hold(P3, ?card) -> solution(?card)
- -hold(P1, Rope) ^ -hold(P2, Rope) ^ -hold(P3, Rope)

If we substitute ?card with Rope everywhere (?card Rope) then statement is equivalent to the left-hand side of the rule

Two expressions are unifiable iff there exists a substitution list (binding list) that, when applied to both expressions, makes them the same

Unification

- Three valid types of substitutions:

  variable -> constant

  variable1 -> variable2

  variable -> function, if function doesn't contain variable

Unification Code

// Determine whether two expressions can be unified. If yes, return bindings. If no, return FAIL

```
Function unify(p1, p2, bindings)
    If (p1 = p2) return bindings
    If var(p1) try to add (p1 p2) to list of bindings
    If var(p2) try to add (p2 p1) to list of bindings
    If p1&p2 are length 1 return FAIL
    If (length(p1) != length(p2)) return FAIL
    Recursively unify each term pair in p1 and p2
    Return binding list
```

Examples

- f(?x, ?x) and f(?y, ?z)
  - ?
- f(?x, ?x) and f(John, Fred)
  - ?
- f(?x, ?y, ?z) and f(?y, John, Fred)
  - ?
- f(?x, ?y, ?z) and f(?y, ?z, Fred)
  - ?
- p(?x, ?x) and p(cook,henderson)

  ?
- p(?x, ?x) and p(cook, ?y) ?

Examples

- f(?x, ?x) and f(?y, ?z)
  - OK ((?x ?y) (?y ?z))
- f(?x, ?x) and f(John, Fred)
  - ?
- f(?x, ?y, ?z) and f(?y, John, Fred)
  - ?
- f(?x, ?y, ?z) and f(?y, ?z, Fred)
  - ?

- p(?x, ?x) and p(cook,henderson) ?
- p(?x, ?x) and p(cook, ?y) ?

Examples

- f(?x, ?x) and f(?y, ?z)
  - OK ((?x ?y) (?y ?z))
- f(?x, ?x) and f(John, Fred)
  - NO
- f(?x, ?y, ?z) and f(?y, John, Fred)
  - ?
- f(?x, ?y, ?z) and f(?y, ?z, Fred)
  - ?

- p(?x, ?x) and p(cook,henderson) ?
- p(?x, ?x) and p(cook, ?y) ?

Examples

- f(?x, ?x) and f(?y, ?z)
  - OK ((?x ?y) (?y ?z))
- f(?x, ?x) and f(John, Fred)
  - NO
- f(?x, ?y, ?z) and f(?y, John, Fred)
  - NO
- f(?x, ?y, ?z) and f(?y, ?z, Fred)
  - ?

- p(?x, ?x) and p(cook,henderson) ?
- p(?x, ?x) and p(cook, ?y) ?

Examples

- f(?x, ?x) and f(?y, ?z)
  - OK ((?x ?y) (?y ?z))
- f(?x, ?x) and f(John, Fred)
  - NO
- f(?x, ?y, ?z) and f(?y, John, Fred)
  - NO
- f(?x, ?y, ?z) and f(?y, ?z, Fred)
  - OK ((?x ?y) (?y ?z) (?z Fred))

- p(?x, ?x) and p(cook,henderson) ?
- p(?x, ?x) and p(cook, ?y) ?

Examples

- f(?x, ?x) and f(?y, ?z)
  - OK ((?x ?y) (?y ?z))
- f(?x, ?x) and f(John, Fred)
  - NO
- f(?x, ?y, ?z) and f(?y, John, Fred)
  - NO
- f(?x, ?y, ?z) and f(?y, ?z, Fred)
  - OK ((?x ?y) (?y ?z) (?z Fred))

- p(?x, ?x) and p(cook,henderson)

  NO
- p(?x, ?x) and p(cook, ?y) ?

Examples

- f(?x, ?x) and f(?y, ?z)
  - OK ((?x ?y) (?y ?z))
- f(?x, ?x) and f(John, Fred)
  - NO
- f(?x, ?y, ?z) and f(?y, John, Fred)
  - NO
- f(?x, ?y, ?z) and f(?y, ?z, Fred)
  - OK ((?x ?y) (?y ?z) (?z Fred))

- p(?x, ?x) and p(cook,henderson)

  NO
- p(?x, ?x) and p(cook, ?y) OK ((?x cook) (?y cook))

Unifiers

- Note that there can be more than one binding list that unifies two expressions.
    - (f ?x) (f ?y)
    - binding list = ((?x ?y)) or ((?x foo) (?y foo))
- In general, we prefer to not overly constrain the substitutions.
- If keep general, result can apply to greater number of future situations.
- The less constraints placed, the more general the substitution (the binding list, or unifier)
- Given expressions p and q, a **unifier** of p and q is any binding list b such that pb = qb (pb means binding list b is applied to expression p)

Most General Unifier

- Unifier b_1 is more general than unifier b_2 if for every expression p, pb_2 is an instance of pb_1 (or $pb_1b_3 = pb_2$)
- ((?x ?y)) is more general than ((?x foo) (?y foo))
  - because (f ?x) ((?x ?y)) ((?y foo)) = (f ?x) ((?x foo) (?y foo))
- If two expressions are unifiable, then there exists an mgu (most general unifier)
- The code we designed returns an mgu.

Q(F(?x), ?z, A) and Q(?a, ?z, ?y)

Unifiable?

Yes! substitution list = ((?a F(?x)) (?y A))

P(?x) and P(A)?

P(F(?x, G(A,?y)), G(A,?y)) and P(F(?x,?z), ?z)?

Q(?x, ?y, A) and Q(B, ?y, ?z)?

R(?x) and R(F(?x))?

Find the MGU of {p(f(a), g(Y)) and p(X, X)}

Sol: S_0 => Here, L_1 = p(f(a), g(Y)), and L_2 = p(X, X)

SUBST θ= {f(a) / X}

S1 => L_1 = p(f(a), g(Y)), and L_2 = p(f(a), f(a))

SUBST θ= {f(a) / g(y)}, **Unification failed**.

First order Inference

1. Resolution
2. Forward Chaining
3. Backward Chaining

Converting To Clausal Form

Two benefits of seeing this process:

- Learn sound inference rules
- Convert FOPC to clausal form for use in resolution proofs

First Order Logic: Conversion to CNF

1. Eliminate biconditionals and implications:

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.

2. Move ¬ inwards:

- $\neg(\forall\ x\ p) \equiv \exists\ x\ \neg p$,
- $\neg(\exists\ x\ p) \equiv \forall\ x\ \neg p$,
- $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$,
- $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$,
- $\neg\neg\alpha \equiv \alpha$.

3. Standardize variables apart by renaming them: each quantifier should use a different variable.

First Order Logic: Conversion to CNF

4. Skolemize: each existential variable is replaced by a Skolem constant or Skolem function of the enclosing universally quantified variables.

- For instance, ∃x Rich(x) becomes Rich(G1) where G1 is a new Skolem constant.
- “Everyone has a heart”

  ∀ x P erson(x) ⇒ ∃ y Heart(y) ∧ Has(x, y) becomes ∀ x P erson(x) ⇒ Heart(H(x)) ∧ Has(x, H(x)), where H is a new symbol (Skolem function).

5. Drop universal quantifiers

- For instance, ∀ x P erson(x) becomes Person(x).

6. **Distribute ∧ over ∨:**

- (α ∧ β) ∨ γ ≡ (α ∨ γ) ∧ (β ∨ γ).

Examples

- P(A) -> Q(B,C)
  ~P(A) v Q(B,C)
- ~(P(A) -> Q(B,C))
  P(A), ~Q(B,C)
- P(A) ^ (Q(B,C) v R(D))
  P(A), Q(B,C) v R(D)
- P(A) v (Q(B,C) ^ R(D))
  P(A) v Q(B,C)
  P(A) v R(D)

- FORALL x P(x)
  P(x)
- FORALL x P(x) -> Q(x,A)
  ~P(x) v Q(x,A)
- EXISTS x P(x)
  P(E), where E is a new constant
- P(A) -> EXISTS x Q(x)
  ~P(A) v Q(F)
- FORALL x P(x)
  ~P(G)

Inference in First Order Logic

Logical inference algorithms use forward and backward chaining approaches, which require KB in the form of the **first-order definite clause**.

Definite clause: A clause which is a disjunction of literals with **exactly one positive literal** is known as a definite clause or strict horn clause.

Horn clause: A clause which is a disjunction of literals with **at most one positive literal** is known as horn clause. Hence all the definite clauses are horn clauses.

Example: (¬ p V ¬ q V k). It has only one positive literal k.

It is equivalent to p ∧ q → k.

Forward Chaining

- Forward chaining is also known as a forward deduction or forward reasoning method when using an inference engine. Forward chaining is a form of reasoning which start with atomic sentences in the knowledge base and applies inference rules (Modus Ponens) in the forward direction to extract more data until a goal is reached.
- The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.

Forward Chaining

Properties of Forward-Chaining:

- It is a down-up approach, as it moves from bottom to top.
- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.
- Forward -chaining approach is commonly used in the expert system, such as CLIPS, business, and production rule systems.

Forward Chaining

Example:

"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."

Prove that **"Robert is criminal."**

To solve the above problem, first, we will convert all the above facts into first-order definite clauses, and then we will use a forward-chaining algorithm to reach the goal.

Forward Chaining

Facts Conversion into FOL:

- It is a crime for an American to sell weapons to hostile nations. (Let's say p, q, and r are variables)

  American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) → Criminal(p) ...(1)
- Country A has some missiles.

Owns(A, p) ∧ Missile(p).

It can be written in two definite clauses by using Existential Instantiation, introducing new Constant T1.

Owns(A, T1) ......(2)

Missile(T1) ....(3)

Forward Chaining

Facts Conversion into FOL:

- All of the missiles were sold to country A by Robert.

  ∀p **Missiles(p) ∧ Owns (A, p) → Sells (Robert, p, A)** **......(4)**

- Missiles are weapons.

  Missile(p) → Weapons (p) **.......(5)**

- Enemy of America is known as hostile.

  Enemy(p, America) →Hostile(p) **........(6)**

- Country A is an enemy of America.

  Enemy (A, America) **.........(7)**

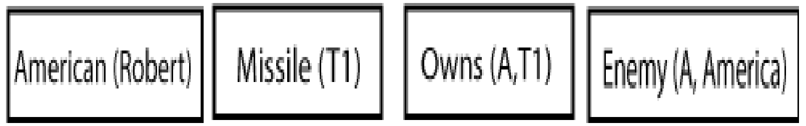
- Robert is American

  American(Robert). **..........(8)**

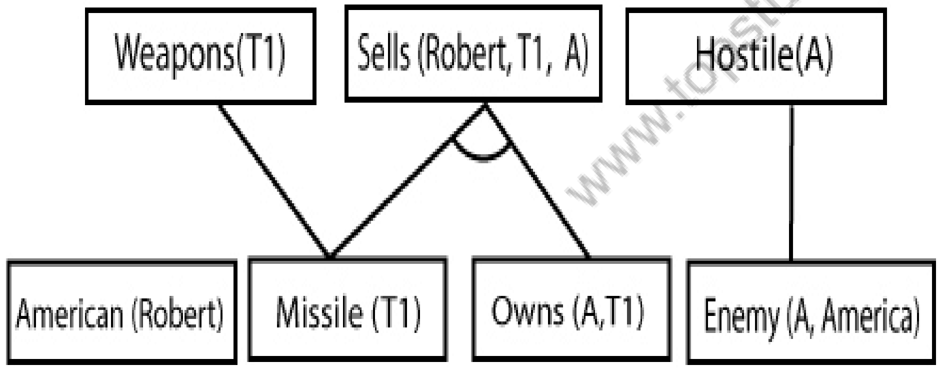
Forward Chaining

Forward chaining proof:

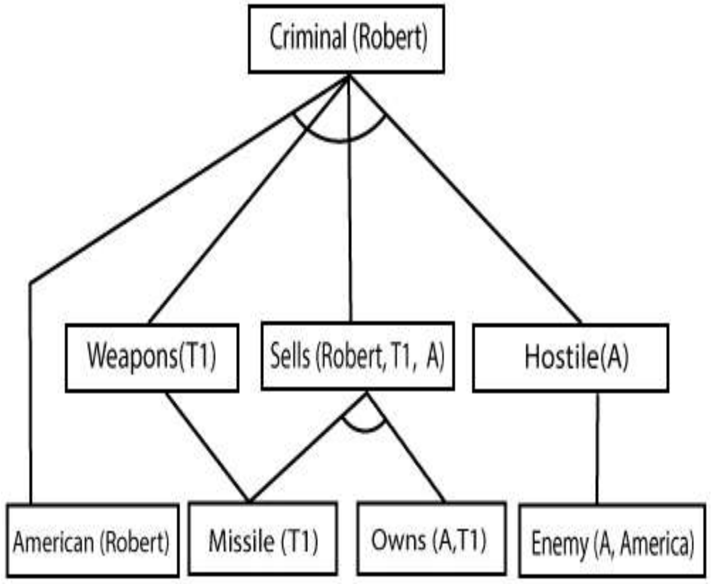
Step 1



Step 2



Step 3



Backward Chaining:

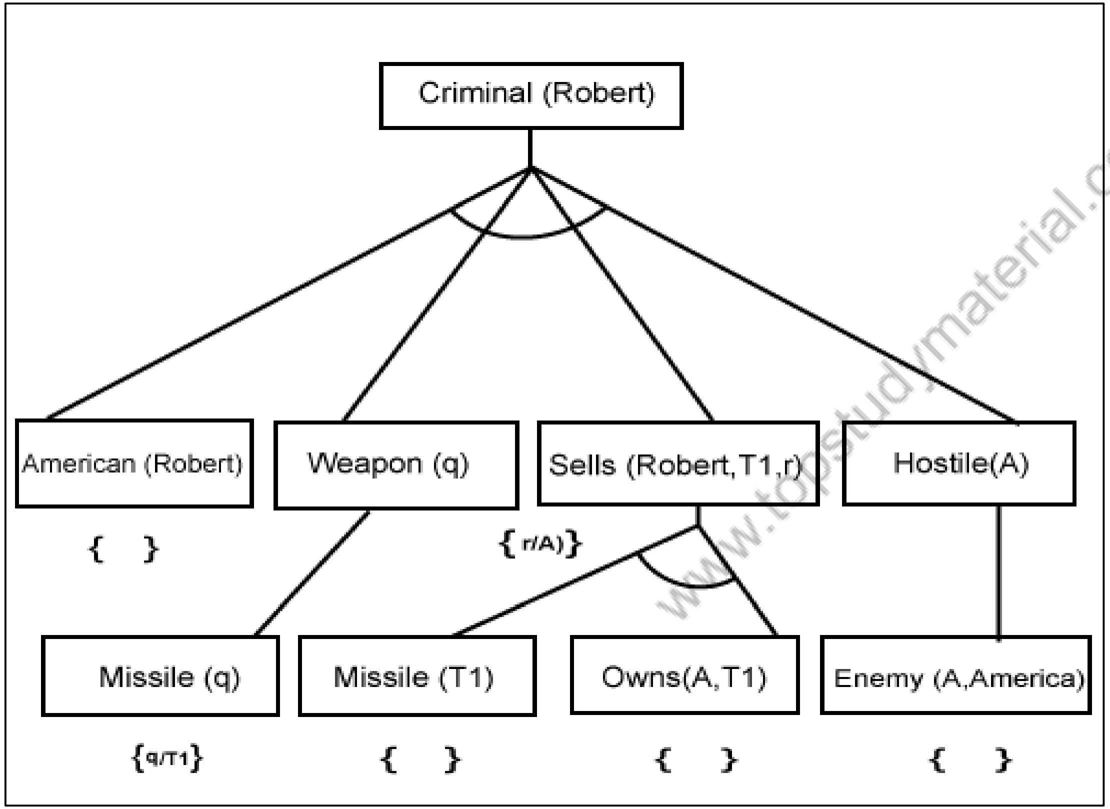
- Backward-chaining is also known as a backward deduction or backward reasoning method when using an inference engine.
- A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.

Backward Chaining

Properties of backward chaining:

- It is known as a top-down approach.
- Backward-chaining is based on modus ponens inference rule.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.
- Backward -chaining algorithm is used in game theory, automated theorem proving tools, inference engines, proof assistants, and various AI applications.
- The backward-chaining method mostly used a **depth-first search** strategy for proof.

Backward Chaining



In Backward chaining, we will start with our goal predicate, which is **Criminal(Robert)**, and then infer further rules.

Resolution

- Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions.
- Resolution is used, if there are various statements are given, and we need to prove a conclusion of those statements. Unification is a key concept in proofs by resolutions. Resolution is a single inference rule which can efficiently operate on the **conjunctive normal form or clausal form**.
- **Clause**: Disjunction of literals (an atomic sentence) is called a **clause**. It is also known as a unit clause.
- **Conjunctive Normal Form**: A sentence represented as a conjunction of clauses is said to be **conjunctive normal form** or **CNF**.

Resolution

Resolution can resolve two clauses if they contain complementary literals, which are assumed to be standardized apart so that they share no variables.

This rule is also called the **binary resolution rule** because it only resolves exactly two literals.

Example:

We can resolve two clauses which are given below:

[Animal (g(x) V Loves (f(x), x)]   and   [¬ Loves(a, b) V ¬Kills(a, b)]

Where two complementary literals are:

Loves (f(x), x) and ¬ Loves (a, b)

These literals can be unified with unifier **θ= [a/f(x), and b/x]** ,

and it will generate a resolvent clause:

Resolution

Steps for Resolution:

1. Conversion of facts into first-order logic.
2. Convert FOL statements into CNF
3. Negate the statement which needs to prove (proof by contradiction)
4. Draw resolution graph (unification).

Resolution

Example:

a. **John likes all kind of food.**

b. **Apple and vegetable are food**

c. **Anything anyone eats and not killed is food.**

d. **Anil eats peanuts and still alive**

e. **Harry eats everything that Anil eats.**

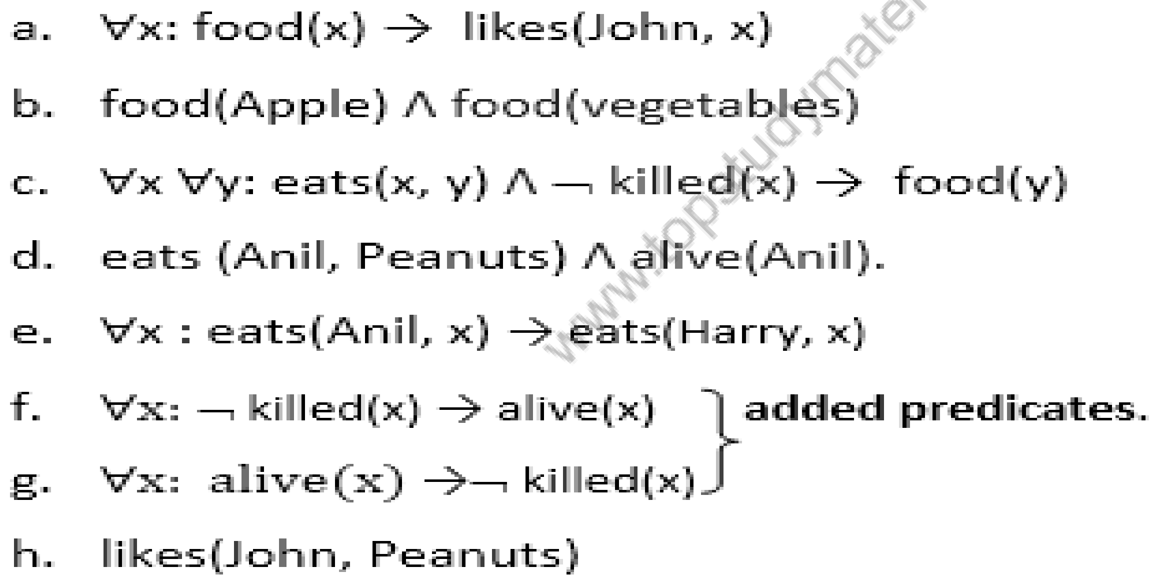
Prove by resolution that:

f. **John likes peanuts.**

Resolution

Conversion of Facts into FOL

In the first step we will convert all the given statements into its first order logic.



a. $\forall x: \text{food}(x) \rightarrow \text{likes}(\text{John}, x)$

b. $\text{food}(\text{Apple}) \land \text{food}(\text{vegetables})$

c. $\forall x\, \forall y: \text{eats}(x, y) \land \neg\, \text{killed}(x) \rightarrow \text{food}(y)$

d. $\text{eats}\,(\text{Anil}, \text{Peanuts}) \land \text{alive}(\text{Anil})$.

e. $\forall x : \text{eats}(\text{Anil}, x) \rightarrow \text{eats}(\text{Harry}, x)$

f. $\forall x: \neg\, \text{killed}(x) \rightarrow \text{alive}(x)$ **added predicates.**

g. $\forall x: \text{alive}(x) \rightarrow \neg\, \text{killed}(x)$

h. $\text{likes}(\text{John}, \text{Peanuts})$

Resolution

Conversion of FOL into CNF

In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.

a. ¬ food(x) V likes(John, x)
b. food(Apple)
c. food(vegetables)
d. ¬ eats(y, z) V killed(y) V food(z)
e. eats (Anil, Peanuts)
f. alive(Anil)
g. ¬ eats(Anil, w) V eats(Harry, w)
h. killed(g) V alive(g)
i. ¬ alive(k) V ¬ killed(k)
j. likes(John, Peanuts).

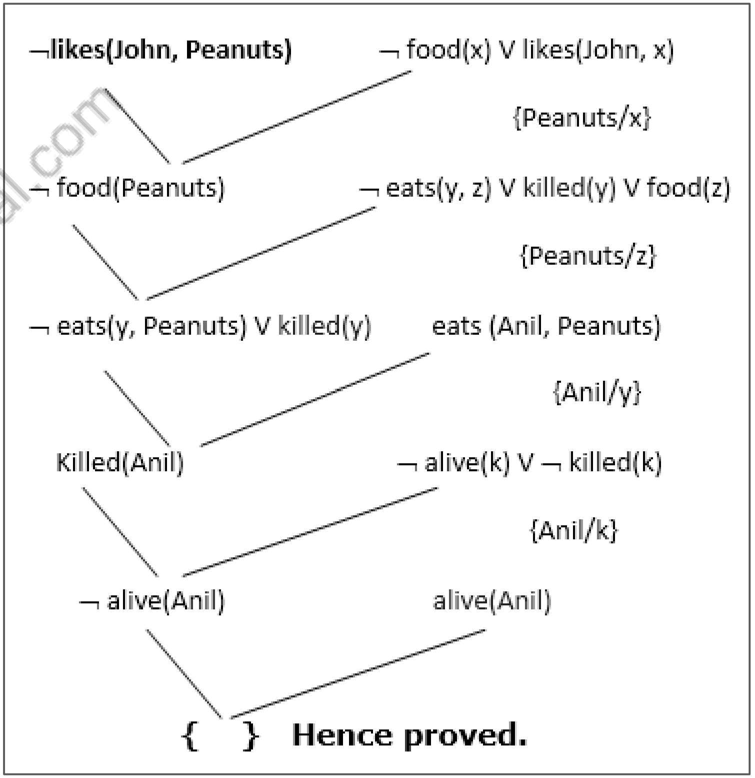
Resolution

Negate the statement to be proved

In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)

Draw Resolution graph:

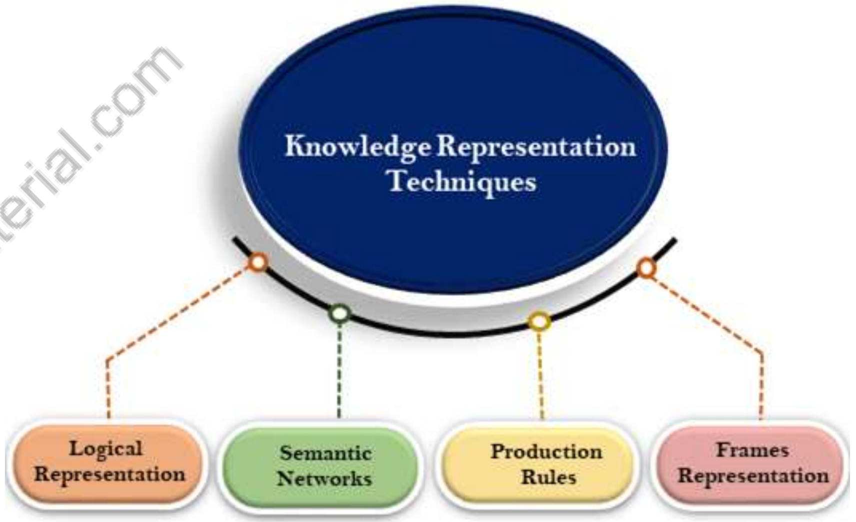
Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



Knowledge Representation

There are mainly four ways of knowledge representation which are given as follows:

1. Logical Representation
2. Semantic Network Representation
3. Frame Representation
4. Production Rules



Knowledge Representation

Logical Representation

Logical representation is a language with some concrete rules which deals with propositions and has no ambiguity in representation. Logical representation means drawing a conclusion based on various conditions.

Logical representation can be categorised into mainly two logics:

a. Propositional Logics
b. Predicate logics

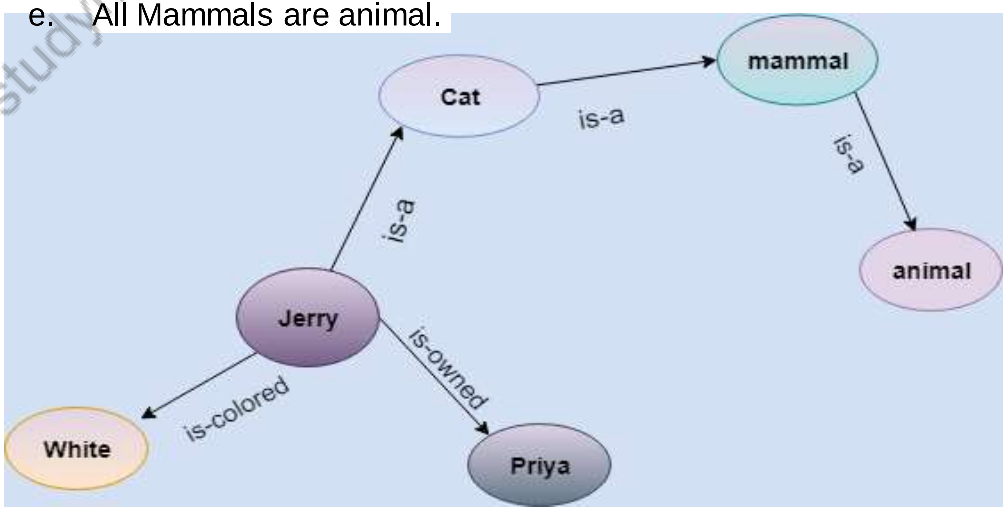
Knowledge Representation

Semantic Network Representation

Semantic networks are alternative of predicate logic for knowledge representation. In Semantic networks, we can represent our knowledge in the form of graphical networks. This network consists of nodes representing objects and arcs which describe the relationship between those objects.

Statements:

a. Jerry is a cat.
b. Jerry is a mammal
c. Jerry is owned by Priya.
d. Jerry is brown colored.
e. All Mammals are animal.



Knowledge Representation

Frame Representation

- A frame is a record like structure which consists of a collection of attributes and its values to describe an entity in the world. Frames are the AI data structure which divides knowledge into substructures by representing stereotypes situations.

Slots	Filters
Title	Artificial Intelligence
Genre	Computer Science
Author	Peter Norvig
Edition	Third Edition
Year	1996
Page	1152

Knowledge Representation

Production Rules

Production rules system consist of (**condition, action**) pairs which mean, "If condition then action". It has mainly three parts:

- The set of production rules
- Working Memory
- The recognize-act-cycle

In production rules agent checks for the condition and if the condition exists then production rule fires and corresponding action is carried out.

```
If Signal== RED
Action Stop
If Signal== GREEN
Action Go
If Signal== YELLOW
Action Wait
```

Ontological Engineering

- In toy domains, the choice of representation is not important.
- It is easy to come up with a consistent vocabulary.
- Complex domains require more general and flexible representations:
- shopping on the Internet
- controlling a robot in a changing environment
- diagnosing problematic situations in wastewater management

Ontological Engineering

- Ontological engineering consists of representing the abstract concepts that occur in many different domains:

Actions
Time
Physical Objects
Beliefs
...

- It is related to knowledge engineering, but operates on a larger scale.

Ontological Engineering

- Representing everything in the world can be a problem.
- We leave placeholders where new knowledge can fit in.
- For example, we can define what it means to be a physical object.
- The different types of object can be filled later:
  robots
  televisions
  books

Ontological Engineering

- The general framework of concepts is called an upper ontology,
- because of the convention of drawing graphs with:

  the general concepts at the top

  the more specific concepts below them

Ontological Engineering

- Certain aspects of the real world are hard to capture in FOL:
- Almost all generalizations have exceptions
- They hold only to a degree

  Example:

  “Tomatoes are red” is a useful rule, but...
- Some tomatoes are green, yellow or orange.
- The ability to handle exceptions and uncertainty is extremely important.

Ontological Engineering

- For any special-purpose ontology, it is possible to move toward greater generality.
- Do all ontologies converge on a general-purpose ontology?

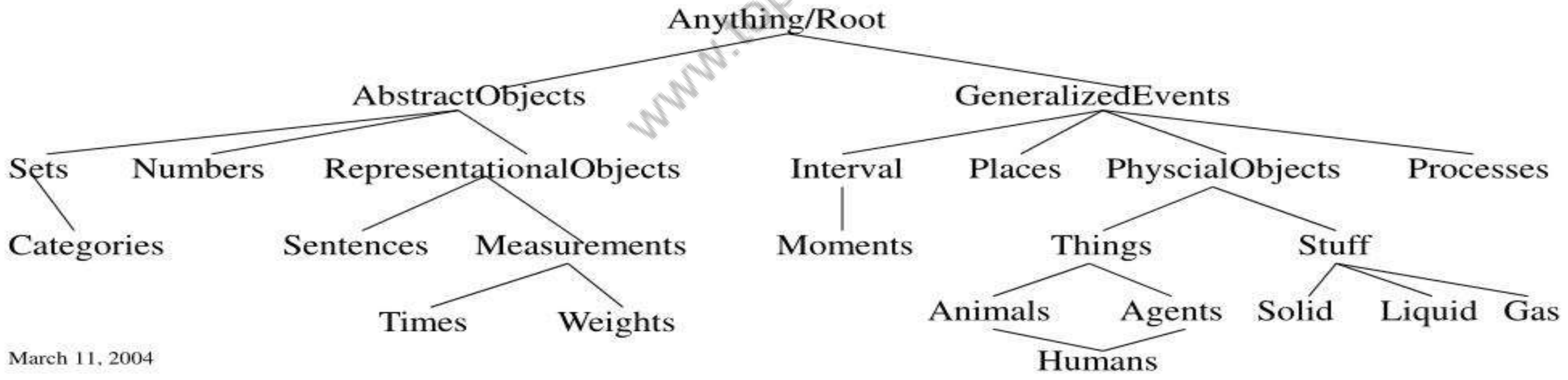
    Possibly

Ontological Engineering

- Main characteristics of general-purpose ontologies that distinguish them from collections of special-purpose ontologies:
- A general-purpose ontology should be applicable in any special-purpose domain.
- In any sufficiently demanding domain, different areas of knowledge have to be unified.
- Reasoning and problem solving could involve several areas simultaneously.

Ontological Engineering

- Like knowledge engineering but applies to general-purpose knowledge bases
- Ultimate goal is to represent everything in the world!!
- Result is an upper ontology



Special- and General-purpose Ontologies

- Special-purpose ontology:
  - Designed to represent a specific domain of knowledge;
    - genetics (GO)
    - immune system (IMGT)
    - mathematics (Tom Gruber)
- General-purpose ontology:
  - Should be applicable in any special-purpose domain
  - Unifies different domains of knowledge
- Upper ontology provides highest level framework - all other concepts follow

Cyc Upper Ontology

- Cycorp released 3,000 upper-level concepts into public domain
- Cyc Upper Ontology satisfies two important criteria;
  - It is universal: Every concept can be linked to it
  - It is articulate: Distinctions are necessary and sufficient for most purposes

Categories and Objects

- Concepts are organized in taxonomies, linked by relations and conforming to axioms.
- Axioms can be very general, e.g.: “a fox is not a shifty deceptive person” or “a Firefox is not a fox”.
- The organization of objects into categories is a vital part of knowledge representation.

Categories and Objects

- Interaction with the world takes place at the level of individual objects, but:
- much reasoning takes place at the level of categories.
- Example:
- A shopper might have the goal of buying a mouse, rather than a particular mouse such as a V470 Cordless Laser Mouse.

Categories and Objects

- Categories can serve to make predictions about objects, once they are classified.
- Inferring the presence of certain objects from perceptual input
- Inferring category membership from the perceived properties of the object
- Example:
- From an object being a watermelon, one infers that it would be useful for fruit salad.

Categories - Representation

- Two choices for representation:
  - Predicate
    - Basketball(b)
  - Object
    - Basketballs
    - Member(b, Basketballs)
    - Subset(Basketballs, Balls)

Categories - Organizing

- Inheritance:
  - All instances of the category *Food* are edible
  - *Fruit* is a subclass of *Food*
  - *Apples* is a subclass of *Fruit*
  - Therefore, *Apples* are edible
- The Class/Subclass relationships among *Food*, *Fruit* and *Apples* is a taxonomy

Categories - Partitioning

- Disjoint: The categories have no members in common
- Exhaustive Decomposition: Every member of the category is included in at least one of the subcategories
- Partition: Disjoint exhaustive decomposition

Categories - Partitioning

Disjoint({Animals,Vegetables})

Disjoint(s) <=> ($\forall$c1,c2 c1$\in$s $\wedge$ c2$\in$s $\wedge$ c1$\neq$c2 $\Rightarrow$ Intersection(c1,c2) = { })

ExhaustiveDecomposition({Americans,Canadians ,Mexicans},NorthAmericans})

ExhaustiveDecomposition(s,c) $\Leftrightarrow$ ($\forall$i i$\in$c $\Leftrightarrow$ $\exists$c2 c2$\in$s $\wedge$ i$\in$c2)

Partition({Males,Females},Animals)

Parition(s,c) $\Leftrightarrow$ Disjoint(s) $\wedge$ ExhaustiveDecomposition(s,c)

Situation Calculus

- The states resulting from executing actions
- Ontology:
  - Situations: logical terms describing initial situation and all situations that result from executing actions on a given situation

    Result(a,s)
  - Fluents: functions and predicates that may be different in different situations

    Age(Wumpus,S0) is Wumpus age in situation S0
  - Atemporal or eternal: functions and predicates that are constant across all situations

    Gold(G1)

Situation Calculus – Actions

- Each action described by two axioms:
  - Possibility Axiom:

    Preconditions $\Rightarrow$ Poss(a,s)
  - Effect Axiom:

    Poss(a,s) $\Rightarrow$ changes that result from taking action

Situation Calculus - Example

Possibility Axioms:

$At(Agent,x,s) \wedge Adjacent(x,y) \Rightarrow Poss(Go(x,y),s).$

$Gold(g) \wedge At(Agent,x,s) \wedge At(g,x,s) \Rightarrow$
$Poss(Grab(g),s).$

$Holding(g,s) \Rightarrow Poss(Release(g),s).$

Effect Axioms:

$Poss(Go(x,y),s) \Rightarrow At(Agent,y,Result(Go(x,y),s).$

$Poss(Grab(g),s) \Rightarrow Holding(g,Result(Grab(g),s)).$

$Poss(Release(g),s) \Rightarrow$
$\neg Holding(g,Result(Grab(g),s)).$

Go for the Gold!

GOAL: Bring the gold from [1,2] to [1,1]

At(Agent,[1,1],S0) ∧ At(G1,[1,2],S0).

¬Holding(G1,S0).

Gold(G1).

Adjacent([1,1],[1,2]) ∧ Adjacent([1,2],[1,1]).

Do It:

Go([1,1],[1,2]).

Result:

At(Agent,[1,2],Result(Go([1,1],[1,2]),S0)).

Now, can I grab the gold?

Grab(G1).

The Frame Problem

Result:

At(Agent,[1,2],Result(Go([1,1],[1,2]),S0).

Now, can I grab the Gold?

Grab(G1).

What in the knowledge base allows me to go from my Result (above) to Grab(G1)?

nothing

Time and Event Calculus

- Event Calculus based on points in time
- Fluents hold at points in time as opposed to holding in situations

"A fluent is true at a point in time if the fluent was initiated by an event at some time in the past and was not terminated by an intervening event."

Event Calculus

- Initiates(e,f,t) and Terminates(w,f,t)
- Event Calculus Axiom:

  $T(f,t2) \Leftrightarrow \exists e,t \;\; Happens(e,t) \wedge Initiates(e,f,t) \wedge (t<t2) \wedge \neg Clipped(f,t,t2)$

  $Clipped(f,t,t2) \Leftrightarrow \exists e,t \;\; Happens(e,t1) \wedge Terminates(e,f,t1) \wedge (t < t1) \wedge (t1 < t2)$

Generalized Events

- Combines aspects of space and time calculus
- Allows representation of events occurring in a space-time continuum

World War II is an event that happened in various geographic locations during a specific period of time within the 20th century.

Intervals

- Moment: has temporal duration of zero
- Extended Interval: has temporal duration of greater than zero

  Partition({Moments,ExtendedIntervals},Intervals)

  Member(i,Moments) $\Leftrightarrow$ Duration(i) = Seconds(0).

Intervals Ontology

$\text{Meet}(i,j) \Leftrightarrow \text{Time}(\text{End}(i)) = \text{Time}(\text{Start}(j)).$

$\text{Before}(i,j) \Leftrightarrow \text{Time}(\text{End}(i)) < \text{Time}(\text{Start}(j)).$

$\text{After}(j,i) \Leftrightarrow \text{Before}(i,j).$

$\text{During}(i,j) \Leftrightarrow \text{Time}(\text{Start}(j)) \leq \text{Time}(\text{Start}(i))$
$\quad \wedge \text{Time}(\text{End}(i)) \leq \text{Time}(\text{End}(j)).$

$\text{Overlap}(i,j) \Leftrightarrow \exists k \; \text{During}(k,i) \wedge \text{During}(k,j).$

Unit 5:Reasoning

Mental Objects and Modal Logic,

Reasoning Systems for Categories,

Reasoning with Default Information

Session Plan

Mental Event and Mental Objects

- A mental event is **any event that happens within the mind of a conscious individual**.
- Examples include thoughts, feelings, decisions, dreams, and realizations.
- Some believe that mental events are not limited to human thought but can be associated with animals and artificial intelligence as well.

Mental Event and Mental Objects

- A mental event is **any event that happens within the mind of a conscious individual**.
- Examples include thoughts, feelings, decisions, dreams, and realizations.
- Some believe that mental events are not limited to human thought but can be associated with animals and artificial intelligence as well.

Mental Events and Mental Objects

- Knowledge about beliefs, specifically about those beliefs held by an agent
  - "Which agent knows about the geography of Maine?"
- Provides an agent the ability to reason about beliefs of agents
- However, need to define propositional attitudes, such as *Believes*, *Knows* and *Wants* as relations where the second argument is referentially opaque (no substitution of equal terms)

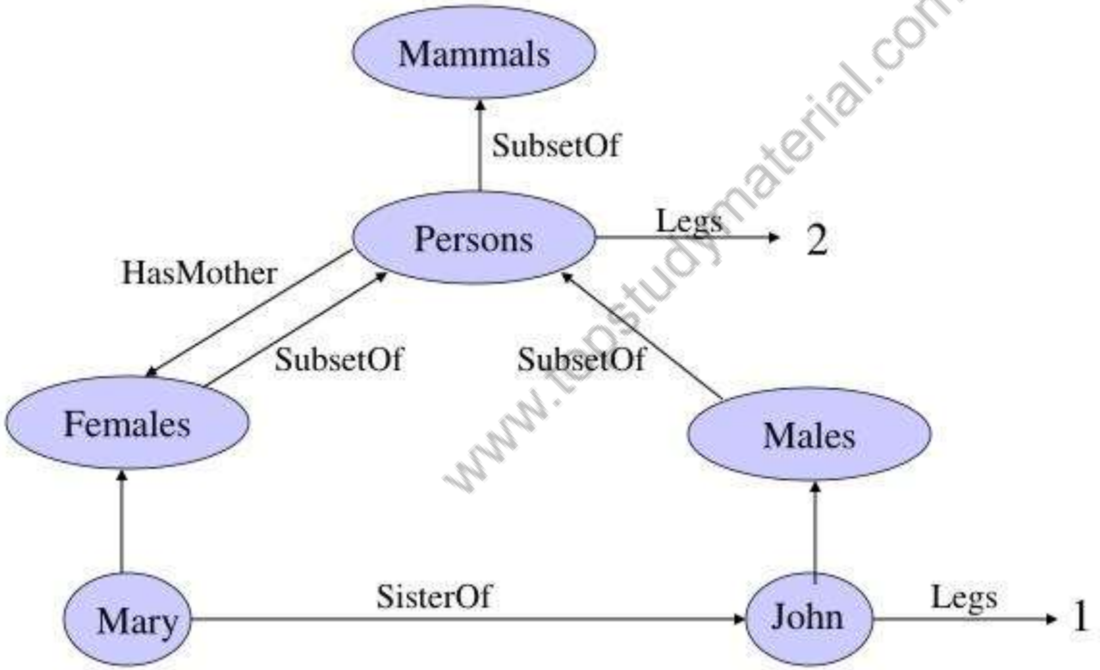
Reasoning Systems for Categories

- Categories are KR building blocks
- Two primary systems for reasoning:
  - Semantic Networks
    - Graphical aids for visualizing knowledge
    - Mechanisms for inferring properties of objects based on category membership
  - Description Logics
    - Formal language for constructing and combining category definitions
    - Algorithms for classifying objects and determining subsumption relationships

Semantic Networks

- Graphical notation with underlying logical representation
- A form of logic, but not FOL
- Capable of representing objects, relations, quantification, …
- Convenient representation of inheritance
- Multiple Inheritance (sometimes)
- Inverse links
- Extendable using procedural attachments

Semantic Networks



Description Logics

- Notations to make it easier to describe definitions and properties of categories
- Taxonomic structure is organizing principle
- Subsumption: Determine if one category is a subset of another
- Classification: Determine the category in which an object belongs
- Consistency: Determine if membership criteria are logically satisfiable

Description Logics

- CLASSIC was one of first languages (Borgida, et al, 1989)
- "All bachelors are unmarried adult males."
  - DL:

  Bachelor = And(Unmarried,Adult,Male).

  - FOL:

  $\text{Bachelor}(x) \Leftrightarrow \text{Unmarried}(x) \wedge \text{Adult}(x) \wedge \text{Male}(x)$

Description Logics

- What does this DL statement say?

```
And(Man,AtLeast(3,Son),
  AtMost(2,Daughter),
  All(Son,And(Unemployed,Married,
      All(Spouse,Doctor))),
  All(Daughter,And(Professor,
      Fills(Department,Physics,Math)))).
```

Reasoning with Default Information

- Open and Closed worlds
  - Open World: Information provided is not assumed to be complete, therefore inferences may result in sentences whose truth value is unknown
  - Closed World: Information provided is assumed complete, therefore ground sentences not asserted to be true are assumed false
  - Negation as Failure: A negative literal, *not P*, can be "proved" true if the proof of *P* fails

Nonmonotonic Logics: Default Logic

- Default rules express contingencies:

Bird(x) : Flies(x)/Flies(x)

- If Bird(x) is true, and Flies(x) consistent with KB, then conclude Flies(x) (by default)
- Default rule form is;

$P : J_1, \ldots, J_n/C$

- P = Prerequisite; J = Justifications; C = Conclusions
- If any J is false, then C is not true