Inference in First Order Logic

DR. SUNIL SURVE

FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

Inference rules for quantifiers

- Suppose our knowledge base contains the standard folkloric axiom stating that all greedy kings are evil:
  $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
- Then it seems quite permissible to infer any of the following sentences:
  $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
  $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
  $King(Father\ (John)) \wedge Greedy(Father\ (John)) \Rightarrow Evil(Father\ (John))$ …

Inference rules for quantifiers

- The rule of **Universal Instantiation** UNIVERSAL (**UI** for short) says that we can infer any sentence obtained by substituting a **ground term** (a term without variables) for the variable.
- Let SUBST(θ,α) denote the result of applying the substitution θ to the sentence α. Then the rule is written $\frac{\forall v, \alpha}{SUSBT(\{\frac{v}{g}\}, \alpha)}$

  any variable v and ground term g.
- For example, the three sentences given earlier are obtained with the substitutions {x/John}, {x/Richard }, and {x/Father(John)}.

Inference rules for quantifiers

- In the rule for **Existential Instantiation**, the variable is replaced by a single *new constant symbol*.
- The formal statement is as follows: for any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base,
- $$\frac{\exists v\ ,\alpha}{SUSBT(\{\frac{v}{k}\},\alpha)}$$
- For example, from the sentence
  ∃x Crown(x) ∧ OnHead(x, John)
- we can infer the sentence
  Crown(C1) ∧ OnHead(C1, John)
  as long as C1 does not appear elsewhere in the knowledge base

Inference rules for quantifiers

- 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
- Existential Instantiation can be applied once, and then the existentially quantified sentence can be discarded.
- For example, we no longer need ∃x Kill(x, Victim) once we have added the sentence Kill (Murderer, Victim).
- Strictly speaking, the new knowledge INFERENTIAL base is not logically equivalent to the old, but it can be shown to be **inferentially equivalent** in the sense that it is satisfiable exactly when the original knowledge base is satisfiable

Reduction to propositional inference

- An existentially quantified sentence can be replaced by one instantiation, a universally quantified sentence can be replaced by the set of *all possible* instantiations.
- For example, suppose our knowledge base contains just the sentences
  $\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)
  King(John)
  Greedy(John)
  Brother (Richard, John)
- Then we apply UI to the first sentence using all possible ground-term substitutions from the vocabulary of the knowledge base—in this case, {x/John} and {x/Richard }.
- We obtain
  King(John) $\wedge$ Greedy(John) $\Rightarrow$ Evil(John)
  King(Richard) $\wedge$ Greedy(Richard) $\Rightarrow$ Evil(Richard)

Reduction to propositional inference

- This technique of **propositionalization** can be made completely general, that is, every first-order knowledge base and query can be propositionalized in such a way that entailment is preserved.
- When the knowledge base includes a function symbol, the set of possible ground-term substitutions is infinite!
- For example, if the knowledge base mentions the Father symbol, then infinitely many nested terms such as Father (Father (Father (John))) can be constructed.
- Propositional algorithms will have difficulty with an infinitely large set of sentences
- Since any such subset has a maximum depth of nesting among its ground terms, we can find the subset by first generating all the instantiations with constant symbols (Richard and John), then all terms of depth 1 (Father(Richard) and Father(John)), then all terms of depth 2, and so on, until we are able to construct a propositional proof of the entailed sentence.

A first-order inference rule

- If there is some substitution θ that makes each of the conjuncts of the premise of the implication identical to sentences already in the knowledge base, then we can assert the conclusion of the implication, after applying θ.
- In this case, the substitution θ ={x/John} achieves that aim
- Suppose that instead of knowing Greedy(John), we know that *everyone* is greedy: ∀y Greedy(y)
- Then we would still like to be able to conclude that Evil(John)
- What we need for this to work is 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

A first-order inference rule

- 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
- This inference process can be captured as a single inference rule that we call **Generalized Modus Ponens**
- For atomic sentences pi, pi, and q, where there is a substitution θ
- such that SUBST(θ, pi)= SUBST(θ, pi), for all i,

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

- There are n+1 premises to this rule: the n atomic sentences pi and the one implication

A first-order inference rule

- The conclusion is the result of applying the substitution θ to the consequent q.
- For our example:

  p1 is King(John) p1 is King(x)
  p2 is Greedy(y) p2 is Greedy(x)
  θ is {x/John, y/John} q is Evil(x)
  SUBST(θ, q) is Evil(John)
- First, we observe that, for any sentence p (whose variables are assumed to be universally quantified) and for any substitution θ,

  p |= SUBST(θ, p)
  holds by Universal Instantiation

A first-order inference rule

- It holds in particular for a θ that satisfies the conditions of the Generalized Modus Ponens rule.
- Thus, from p1 , . . . , pn we can infer
  SUBST(θ, p1) ∧ . . . ∧ SUBST(θ, pn)
  and from the implication p1 ∧ . . . ∧ pn ⇒ q we can infer
  SUBST(θ, p1) ∧ . . . ∧ SUBST(θ, pn) ⇒ SUBST(θ, q) .
- Now, θ in Generalized Modus Ponens is defined so that SUBST(θ, pi)= SUBST(θ, pi), for all i; therefore the first of these two sentences matches the premise of the second exactly.
- Hence, SUBST(θ, q) follows by Modus Ponens

Unification

- Lifted 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)
- Suppose we have a query AskVars(Knows(John, x)): whom does John know?
  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(y,Mother (y))) = {y/John, x/Mother (John)}
  UNIFY(Knows(John, x), Knows(x, Elizabeth)) = fail
- The last unification fails because x cannot take on the values John and Elizabeth at the same time

Unification

- UNIFY(Knows(John, x), Knows(x17, Elizabeth)) = {x/Elizabeth, x17/John}
- UNIFY should return a substitution that makes the two arguments look the same. But there could be more than one such unifier.
- For example, UNIFY(Knows(John, x), Knows(y, z)) could return {y/John, x/z} or {y/John, x/John, z/John}.

Unification

function UNIFY(x , y, θ) **returns** a substitution to make x and y identical
inputs: x , a variable, constant, list, or compound expression
y, a variable, constant, list, or compound expression
θ, the substitution built up so far (optional, defaults to empty)

if θ = failure **then return** failure
else if x = y **then return** θ
else if VARIABLE?(x) **then return** UNIFY-VAR(x , y, θ)
else if VARIABLE?(y) **then return** UNIFY-VAR(y, x , θ)
else if COMPOUND?(x) **and** COMPOUND?(y) **then**
return UNIFY(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP, θ))
else if LIST?(x) **and** LIST?(y) **then**
return UNIFY(x .REST, y.REST, UNIFY(x .FIRST, y.FIRST, θ))
else return failure

Unification

function UNIFY-VAR(var, x , θ) **returns** a substitution

if {var/val} ∈ θ **then return** UNIFY(val , x , θ)

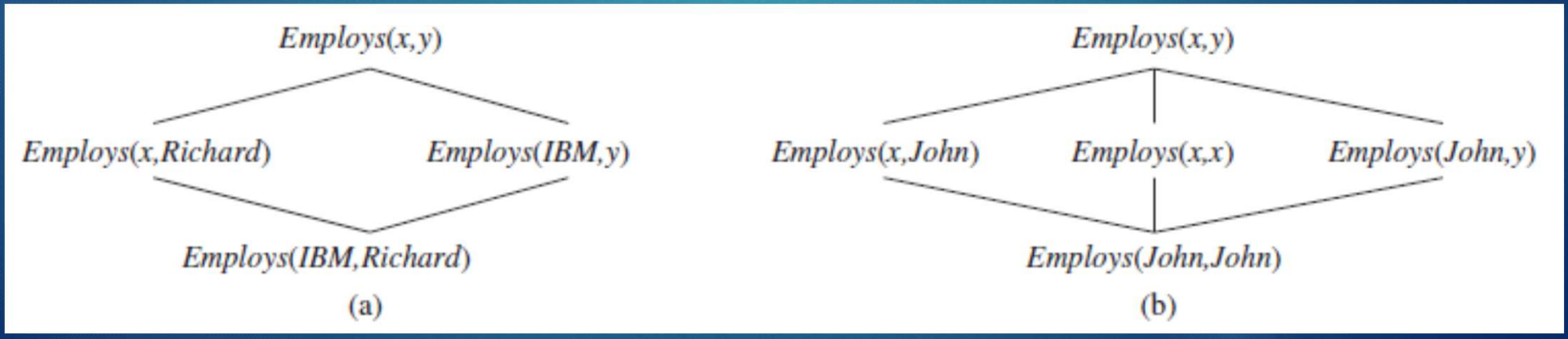
else if {x/val} ∈ θ **then return** UNIFY(var, val , θ)

else if OCCUR-CHECK?(var, x) **then return** failure

else return add {var/x } to θ

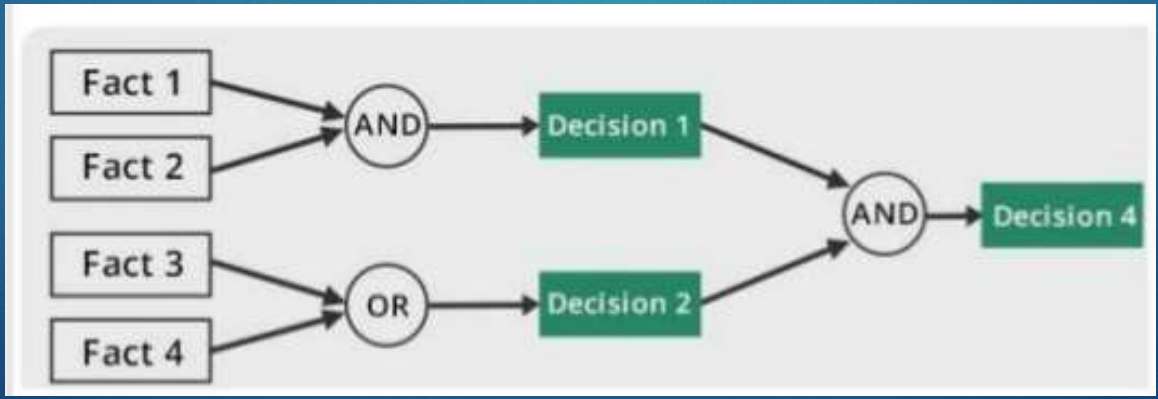
Unification

- For the fact Employs(IBM ,Richard), the queries are
- Employs(IBM ,Richard) Does IBM employ Richard?
- Employs(x,Richard) Who employs Richard?
- Employs(IBM , y) Whom does IBM employ?
- Employs(x, y) Who employs whom?

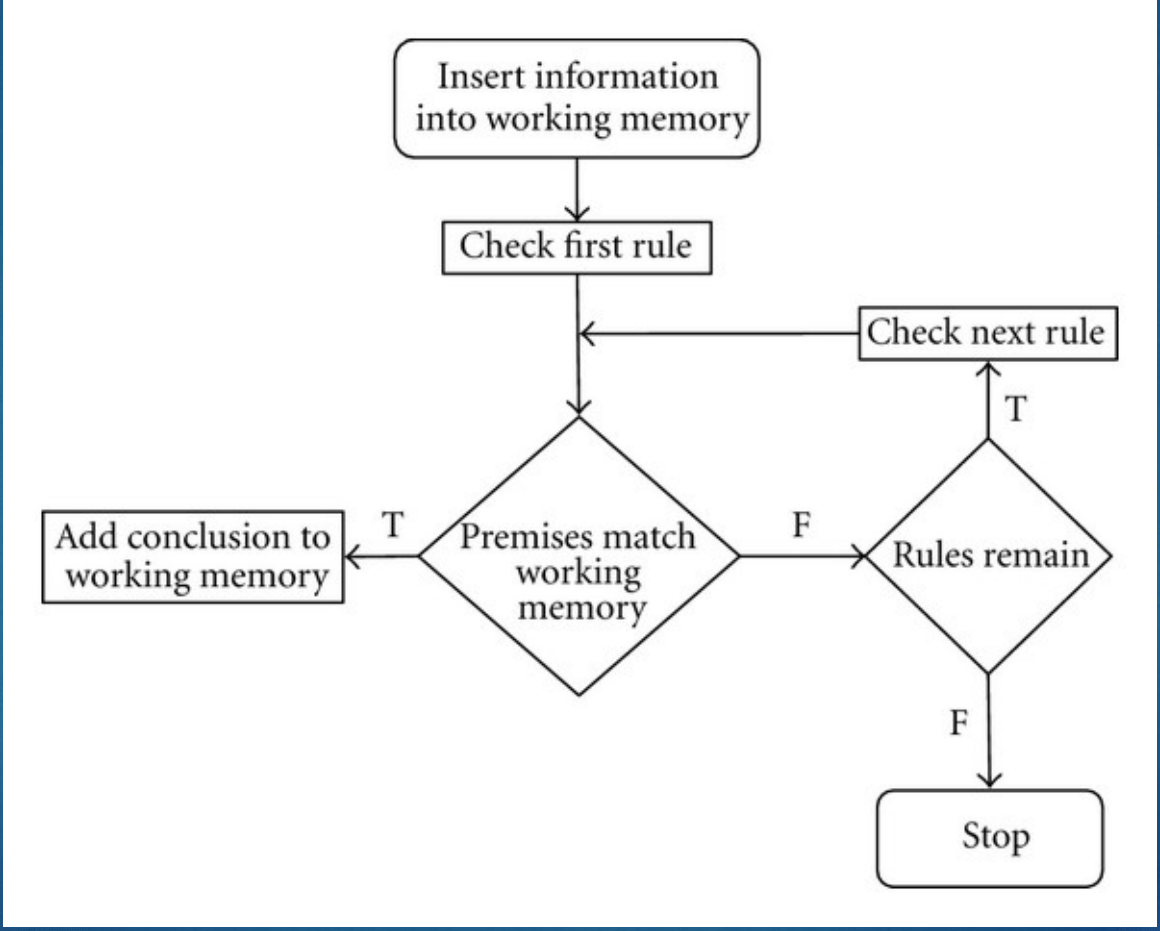


Forward Chaining

- It is a strategy of an expert system to answer the question, “What can happen next?”
- The inference engine follows the chain of conditions and derivations and finally deduce the outcome.
- It considers all the facts and rules, and sorts them before concluding to a solution
- This strategy is followed for working on conclusion, result, or effect
- For example, prediction of share market status as an effect of changes in interest rates



Forward Chaining



Forward Chaining

- Start with the atomic sentences in the knowledge base and apply Modus Ponens in the forward direction, adding new atomic sentences, until no further inferences can be made
- Definite clauses such as **Situation ⇒ Response** are especially useful for systems that make inferences in response to newly arrived information

Forward Chaining

- First-order definite clauses closely resemble propositional definite: they are disjunctions of literals of which *exactly one is positive*.
- A definite clause either is atomic or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal.
- The following are first-order definite clauses:
  $\text{King}(x) \land \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
  $\text{King}(\text{John})$
  $\text{Greedy}(y)$
- First-order literals can include variables, in which case those variables are assumed to be universally quantified
- Not every knowledge base can be converted into a set of definite clauses because of the single-positive-literal restriction

First-order definite clauses

- Consider the following problem:
  The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.
- Prove that West is a criminal.
- Represent these facts as first-order definite clauses
- ". . . it is a crime for an American to sell weapons to hostile nations":
  $American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$ (1)
- "Nono . . . has some missiles."
  $\exists x\ Owns(Nono, x) \land Missile(x)$
  $Owns(Nono, M1)$ (2)
  $Missile(M1)$ (3)

First-order definite clauses

- "All of its missiles were sold to it by Colonel West":
  $\text{Missile}(x) \land \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$ (4)
- Missiles are weapons:
  $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$ (5)
- An enemy of America counts as "hostile":
  $\text{Enemy}(x,\text{America}) \Rightarrow \text{Hostile}(x)$ (6)
- "West, who is American . . .":
  $\text{American}(\text{West})$ (7)
- "The country Nono, an enemy of America . . .":
  $\text{Enemy}(\text{Nono},\text{America})$ (8)
- **Datalog** knowledge bases – No function in knowledge base

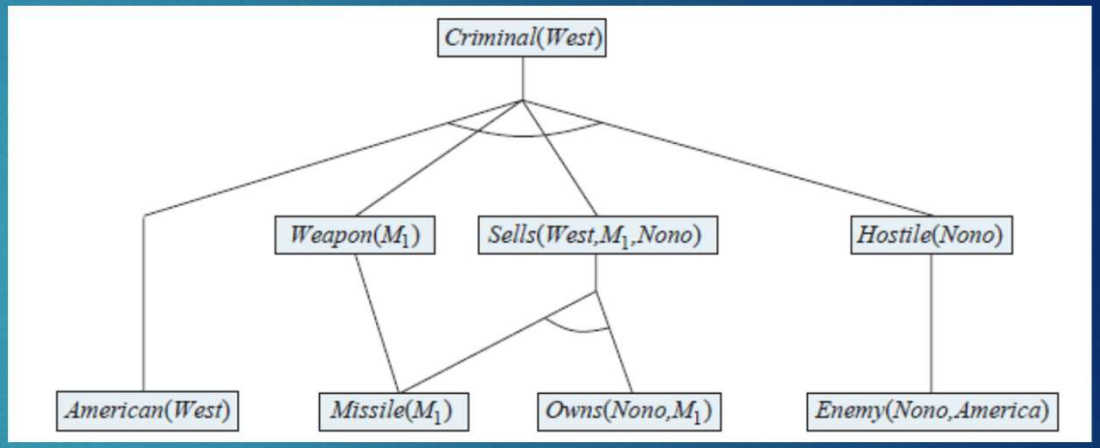
A simple forward-chaining algorithm

function FOL-FC-ASK(KB,α) **returns** a substitution or false
inputs: KB, the knowledge base, a set of first-order definite clauses
α, the query, an atomic sentence
local variables: new, the new sentences inferred on each iteration

repeat until new is empty
    new ←{}
    for each rule **in** KB **do**
        (p1 ∧ . . . ∧ pn ⇒ q) ← STANDARDIZE-VARIABLES(rule)
        for each θ such that SUBST(θ, p1 ∧ . . . ∧ pn) = SUBST(θ, p1 ∧ . . . ∧ pn) for some p1 , . . . , pn in KB
        q ← SUBST(θ, q)
        if q does not unify with some sentence already in KB or new **then** add q to new
            ϕ←UNIFY(q,α)
            if ϕ is not fail **then return** ϕ
    add new to KB
return false

A simple forward-chaining algorithm

- The implication sentences are (1), (4), (5), and (6).
- On the first iteration, rule (1) has unsatisfied premises.
  Rule (4) is satisfied with {x/M1}, and Sells(West,M1, Nono) is added.
  Rule (5) is satisfied with {x/M1}, and Weapon(M1) is added.
  Rule (6) is satisfied with {x/Nono}, and Hostile(Nono) is added.
- On the second iteration, rule (9.3) is satisfied with {x/West, y/M1, z/Nono}, and Criminal (West) is added.



A simple forward-chaining algorithm

- No new inferences are possible at this point because every sentence that could be concluded by forward chaining is already contained explicitly in the KB
- Such a knowledge base is called a **fixed point** of the inference process
- Let k be the maximum **arity** (number of arguments) of any predicate, p be the number of predicates, and n be the number of constant symbols. Clearly, there can be no more than pn^k distinct ground facts, so after this many iterations the algorithm must have reached a fixed point

Efficient forward chaining

- There are three possible sources of inefficiency.
- **Pattern matching:** the “inner loop” of the algorithm involves finding all possible unifiers such that the premise of a rule unifies with a suitable set of facts in the knowledge base. It can be very expensive.
- Second, the algorithm rechecks every rule on every iteration to see whether its premises are satisfied, even if very few additions are made to the knowledge base on each iteration.
- Finally, the algorithm might generate many facts that are irrelevant to the goal

Matching rules against known facts

- The problem of matching the premise of a rule against the facts in the knowledge base might seem simple enough
  $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$.
- Find all the facts that unify with Missile(x); in a suitably indexed knowledge base,
  $\text{Missile}(x) \wedge \text{Owns}(\text{Nono}, x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono})$
- Find all the objects owned by Nono in constant time per object; then, for each object, check whether it is a missile
- **Conjunct ordering** problem: find an ordering to solve the conjuncts of the rule premise so that the total cost is minimized.
- It turns out that finding the optimal ordering is NP-hard, but good heuristics are available

Incremental forward chaining

- *Every new fact inferred on iteration* t *must be derived from at least one new fact inferred on iteration* t − 1
- An incremental forward-chaining algorithm where, at iteration t, check a rule only if its premise includes a conjunct pi that unifies with a fact pi' newly inferred at iteration t−1.
- The rule-matching step then fixes pi to match with pi', but allows the other conjuncts of the rule to match with facts from any previous iteration.
- With suitable indexing, it is easy to identify all the rules that can be triggered by any given fact, and indeed many real systems operate in an "update" mode wherein forward chaining occurs in response to each new fact that is TELLed to the system.
- Inferences cascade through the set of rules until the fixed point is reached, and then the process begins again for the next new fact.

Incremental forward chaining

- Only a small fraction of the rules in the knowledge base are actually triggered by the addition of a given fact
- A partial match is constructed on the first iteration between the rule
  $American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
  and the fact American(West)
- Better to retain and gradually complete the partial matches as new facts arrive, rather than discarding them
- If two literals in a rule share a variable, for example, $Sells(x, y, z) \land Hostile(z)$ in the crime example, then the bindings from each literal are filtered through an equality node.
- A variable binding reaching a node for an nary literal such as Sells(x, y, z) might have to wait for bindings for the other variables to be established before the process can continue

Irrelevant facts

- Forward chaining makes all allowable inferences based on the known facts, *even if they are irrelevant to the goal at hand*
- One way to avoid drawing irrelevant conclusions is to use backward chaining
- Another solution is to restrict forward chaining to a selected subset of rules
- A third approach has emerged in the field of **deductive databases**, which are large-scale databases, like relational databases, but which use forward chaining as the standard inference tool rather than SQL queries
- The idea is to rewrite the rule set, using information from the goal, so that only relevant variable bindings – those belonging to a so-called **magic set –** are considered during forward inference.
- For example, if the goal is Criminal (West), the rule that concludes Criminal (x) will be rewritten to include an extra conjunct that constrains the value of x:
  $Magic(x) \land American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$.

Example

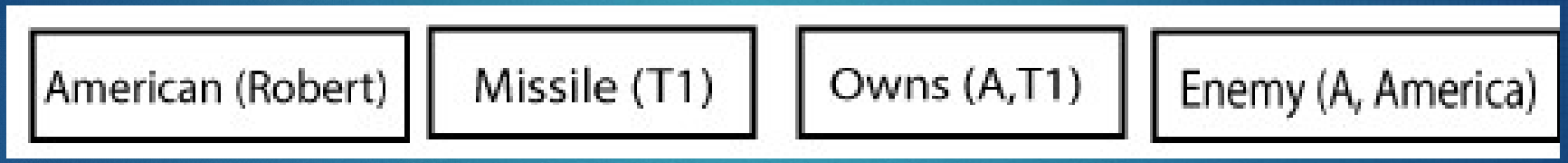
- **"As per the law, it is a crime for an American to sell weapons to hostile nations. Country A, an enemy of America, has some missiles, and all the missiles were sold to it by Robert, who is an American citizen."**
- Prove that **"Robert is criminal."**

Example - 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. **∀p 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)**
- 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)**

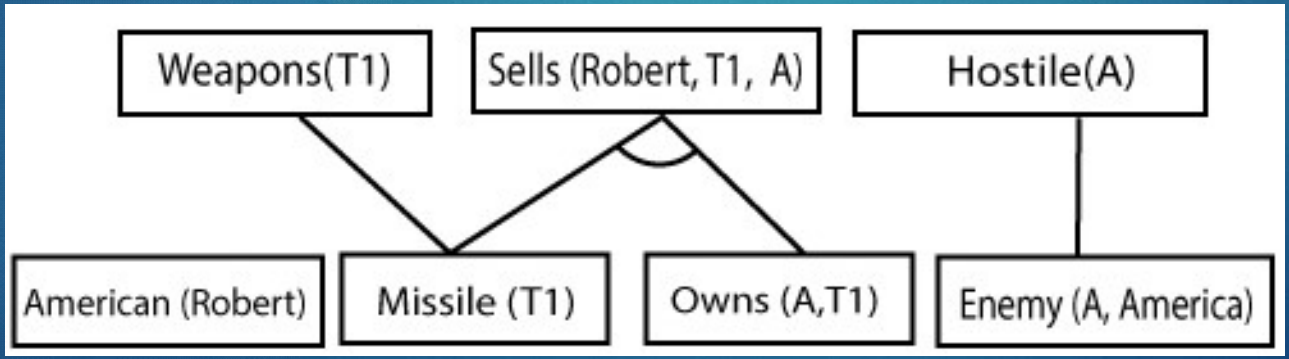
Example - Forward chaining proof

- **Step-1:** start with the known facts and will choose the sentences which do not have implications, such as:
  American(Robert), Enemy(A, America), Owns(A, T1), and Missile(T1).



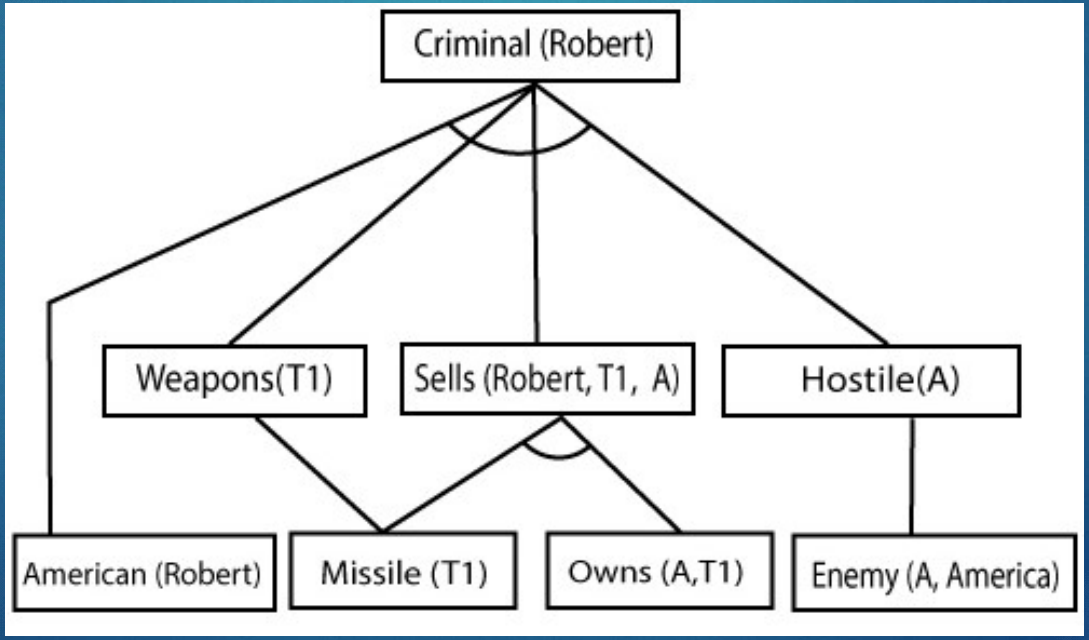
Example - Forward chaining proof

- **Step-2:** facts which infer from available facts and with satisfied premises.
- Rule-(1) does not satisfy premises, so it will not be added in the first iteration.
- Rule-(2) and (3) are already added.
- Rule-(4) satisfy with the substitution {p/T1}, **so Sells (Robert, T1, A)** is added, which infers from the conjunction of Rule (2) and (3).
- Rule-(6) is satisfied with the substitution(p/A), so Hostile(A) is added and which infers from Rule-(7).



Example - Forward chaining proof

- **Step-3:**
- Check Rule-(1) is satisfied with the substitution **{p/Robert, q/T1, r/A}, so we can add Criminal(Robert)** which infers all the available facts. And hence we reached our goal statement.



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.
- **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

- **function** FOL-BC-ASK(KB, query) **returns** a generator of substitutions
  return FOL-BC-OR(KB, query,{ })
- **generator** FOL-BC-OR(KB, goal , θ) **yields** a substitution
  for each rule (lhs ⇒ rhs) in FETCH-RULES-FOR-GOAL(KB, goal) **do**
  (lhs, rhs)←STANDARDIZE-VARIABLES((lhs, rhs))
  for each θ **in** FOL-BC-AND(KB, lhs, UNIFY(rhs, goal , θ)) **do**
  yield θ
- **generator** FOL-BC-AND(KB, goals, θ) **yields** a substitution
  if θ = failure **then return**
  else if LENGTH(goals) = 0 **then yield** θ
  else do
  first,rest ←FIRST(goals), REST(goals)
  for each θ **in** FOL-BC-OR(KB, SUBST(θ, first), θ) **do**
  for each θ **in** FOL-BC-AND(KB, rest , θ) **do**
  yield θ

Backward Chaining - Example

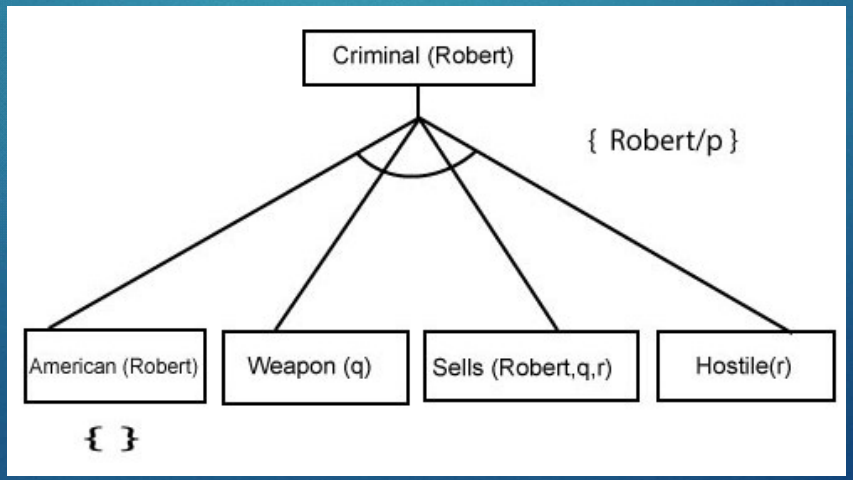
- American (p) ∧ weapon(q) ∧ sells (p, q, r) ∧ hostile(r) ⇒ Criminal(p) ...(1)
- Owns(A, T1) ...(2)
- Missile(T1) ...(3)
- ∃p Missiles(p) ∧ Owns (A, p) ⇒ Sells (Robert, p, A) ...(4)
- Missile(p) ⇒ Weapons (p) ...(5)
- Enemy(p, America) ⇒ Hostile(p) ...(6)
- Enemy (A, America) ...(7)
- American(Robert). ...(8)

Backward Chaining - Example

- **Step-1:**
  - At the first step, we will take the goal fact. And from the goal fact, we will infer other facts, and at last, we will prove those facts true.

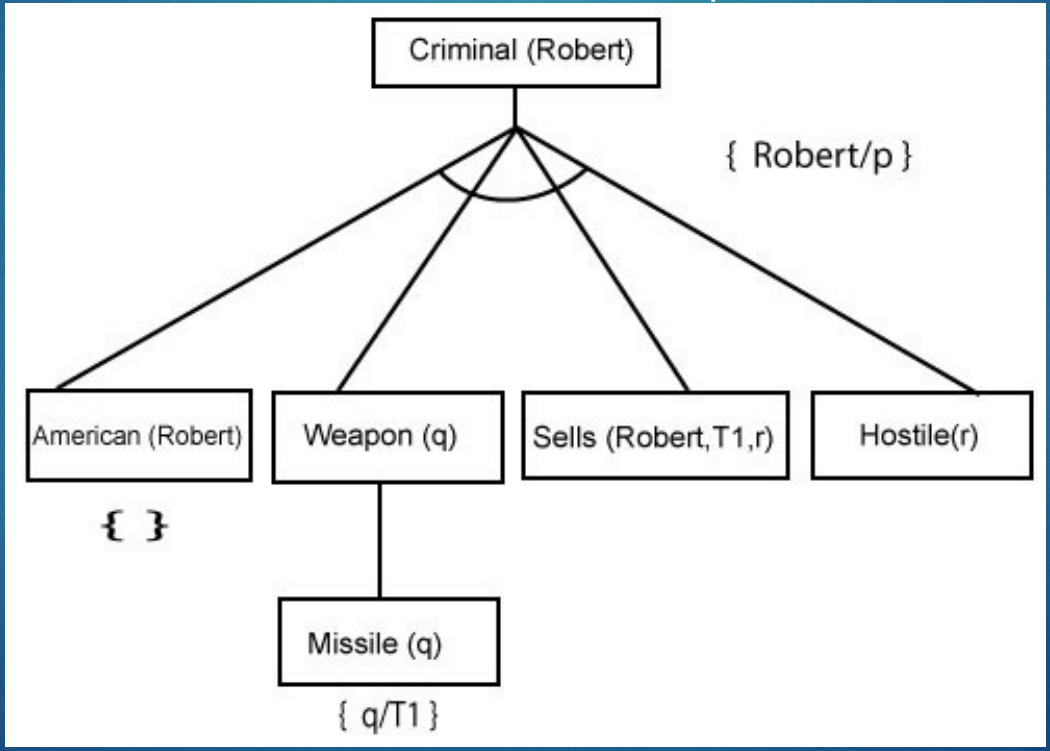
- **Step-2:**

  - At the second step, we will infer other facts form goal fact which satisfies the rules.
  - In Rule-1, the goal predicate Criminal (Robert) is present with substitution {Robert/P}. So we will add all the conjunctive facts below the first level and will replace p with Robert.



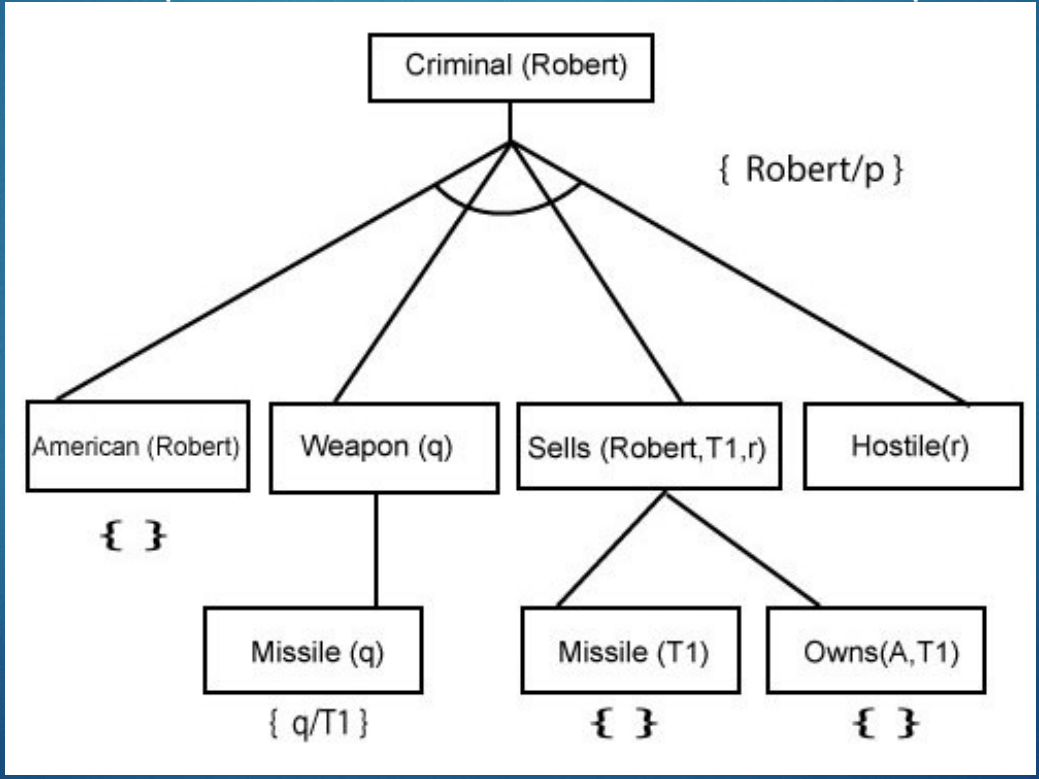
Backward Chaining - Example

- **Step-3:**
  - Extract further fact Missile(q) which infer from Weapon(q), as it satisfies Rule-(5). Weapon (q) is also true with the substitution of a constant T1 at q.



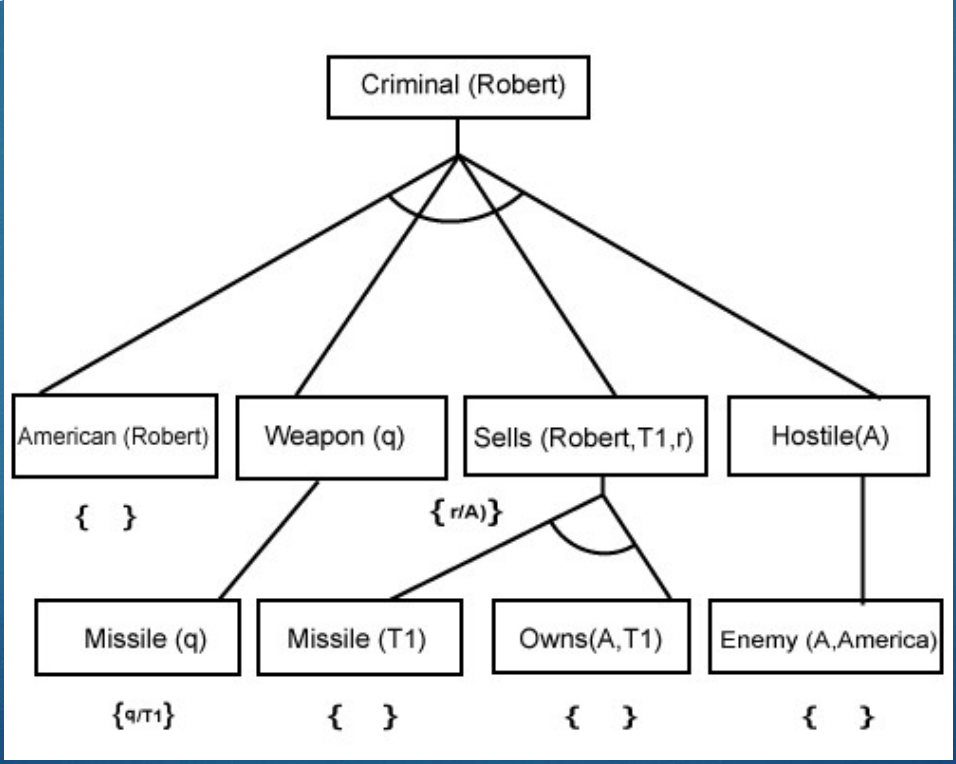
Backward Chaining - Example

- **Step-4:**
  - Infer facts Missile(T1) and Owns(A, T1) form Sells(Robert, T1, r) which satisfies the **Rule- 4**, with the substitution of A in place of r. So these two statements are proved here.



Backward Chaining - Example

- **Step-5:**
  - Infer the fact **Enemy(A, America)** from **Hostile(A)** which satisfies Rule- 6. And hence all the statements are proved true using backward chaining.



Difference between backward chaining and forward chaining

- Forward chaining as the name suggests, start from the known facts and move forward by applying inference rules to extract more data, and it continues until it reaches to the goal, whereas backward chaining starts from the goal, move backward by using inference rules to determine the facts that satisfy the goal.
- Forward chaining is called a **data-driven** inference technique, whereas backward chaining is called a **goal-driven** inference technique.
- Forward chaining is known as the **down-up** approach, whereas backward chaining is known as a **top-down** approach.
- Forward chaining uses **breadth-first search** strategy, whereas backward chaining uses **depth-first search** strategy.
- Forward and backward chaining both applies **Modus ponens** inference rule.

Difference between backward chaining and forward chaining

- Forward chaining can be used for tasks such as **planning, design process monitoring, diagnosis, and classification**, whereas backward chaining can be used for **classification and diagnosis tasks**.
- Forward chaining can be like an exhaustive search, whereas backward chaining tries to avoid the unnecessary path of reasoning.
- In forward-chaining there can be various ASK questions from the knowledge base, whereas in backward chaining there can be fewer ASK questions.
- Forward chaining is slow as it checks for all the rules, whereas backward chaining is fast as it checks few required rules only.

Resolution

- **Conjunctive normal form for first-order logic**
- First-order resolution requires that sentences be in **conjunctive normal form** (CNF), that is, a conjunction of clauses, where each clause is a disjunction of literals
  $\forall x \; American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
  becomes, in CNF,
  $\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x, y, z) \lor \neg Hostile(z) \lor Criminal(x)$
- *Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence*
- In particular, the CNF sentence will be unsatisfiable just when the original sentence is unsatisfiable

Resolution

- The principal difference arises from the need to eliminate existential quantifiers
- "Everyone who loves all animals is loved by someone,"
  $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y\ Loves(y, x)]$
- **Eliminate implications**:
  $\forall\ x\ [\neg\forall\ y\ \neg Animal(y)\ \vee\ Loves(x, y)]\ \vee\ [\exists\ y\ Loves(y, x)]$.
- **Move $\neg$ inwards**: In addition to the usual rules for negated connectives, we need rules for negated quantifiers. Thus, we have
  $\neg\forall x\ p$ becomes $\exists\ x\ \neg p$
  $\neg\exists x\ p$ becomes $\forall\ x\ \neg p$.
- Our sentence goes through the following transformations:
  $\forall\ x\ [\exists\ y\ \neg(\neg Animal(y)\ \vee\ Loves(x, y))]\ \vee\ [\exists\ y\ Loves(y, x)]$
  $\forall\ x\ [\exists\ y\ \neg\neg Animal(y)\ \wedge\ \neg Loves(x, y)]\ \vee\ [\exists\ y\ Loves(y, x)]$
  $\forall\ x\ [\exists\ y\ Animal\ (y)\ \wedge\neg Loves(x, y)]\ \vee\ [\exists\ y\ Loves(y, x)]$

Resolution

- **Standardize variables**: For sentences like (∃x P(x))∨(∃x Q(x)) which use the same variable name twice, change the name of one of the variables
  ∀x [∃y Animal (y) ∧¬Loves(x, y)] ∨ [∃z Loves(z, x)]
- **Skolemize**: **Skolemization** is the process of removing existential quantifiers by elimination
- Existential Instantiation rule: translate ∃x P(x) into P(A), where A is a new constant
- If we blindly apply the rule to the two matching parts we get
  ∀ x [Animal (A) ∧ ¬Loves(x,A)] ∨ Loves(B, x) ,
  which has the wrong meaning entirely: it says that everyone either fails to love a particular animal A or is loved by some particular entity B
- Skolem entities to depend on x and z:
  ∀ x [Animal (F(x)) ∧¬Loves(x, F(x))] ∨ Loves(G(z), x)
  Here F and G are **Skolem functions**

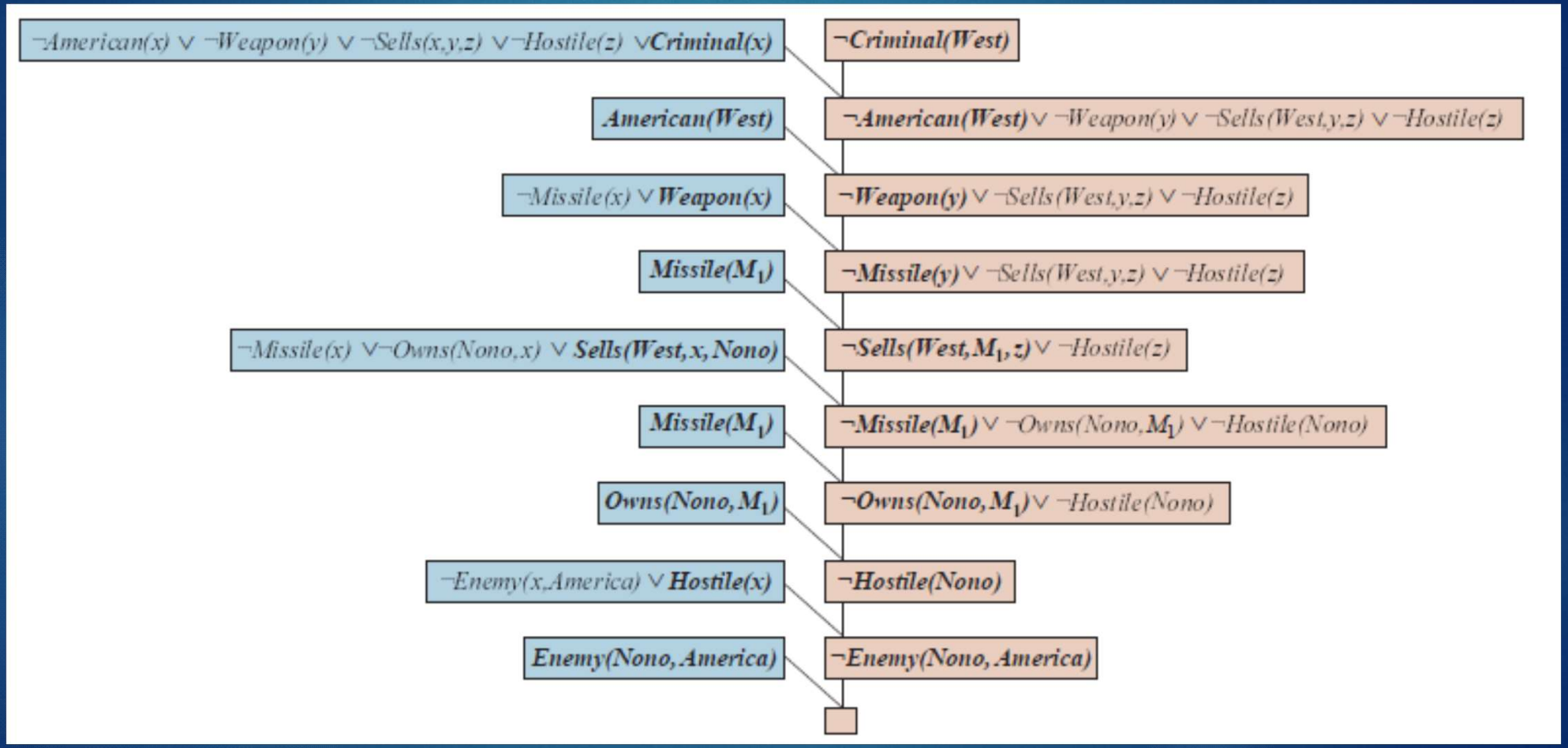
Resolution

- **Drop universal quantifiers**: At this point, all remaining variables must be universally quantified. Moreover, the sentence is equivalent to one in which all the universal quantifiers have been moved to the left. We can therefore drop the universal quantifiers:
  [Animal (F(x)) ∧ ¬Loves(x, F(x))] ∨ Loves(G(z), x)
- **Distribute ∨ over ∧**:
  [Animal (F(x)) ∨ Loves(G(z), x)] ∧ [¬Loves(x, F(x)) ∨ Loves(G(z), x)] .
- This step may also require flattening out nested conjunctions and disjunctions

Resolution

- The sentences in CNF are
- ¬American(x) ∨ ¬Weapon(y) ∨ ¬Sells(x, y, z) ∨ ¬Hostile(z) ∨ Criminal (x)
- ¬Missile(x) ∨ ¬Owns(Nono, x) ∨ Sells(West, x, Nono)
- ¬Enemy(x,America) ∨ Hostile(x)
- ¬Missile(x) ∨Weapon(x)
- Owns(Nono,M1) Missile(M1)
- American(West) Enemy(Nono,America)

Resolution


¬American(x) ∨ ¬Weapon(y) ∨ ¬Sells(x,y,z) ∨ ¬Hostile(z) ∨ Criminal(x)
¬Criminal(West)
American(West)
¬American(West) ∨ ¬Weapon(y) ∨ ¬Sells(West,y,z) ∨ ¬Hostile(z)
¬Missile(x) ∨ Weapon(x)
¬Weapon(y) ∨ ¬Sells(West,y,z) ∨ ¬Hostile(z)
Missile(M1)
¬Missile(y) ∨ ¬Sells(West,y,z) ∨ ¬Hostile(z)
¬Missile(x) ∨ ¬Owns(Nono,x) ∨ Sells(West,x,Nono)
¬Sells(West,M1,z) ∨ ¬Hostile(z)
Missile(M1)
¬Missile(M1) ∨ ¬Owns(Nono,M1) ∨ ¬Hostile(Nono)
Owns(Nono,M1)
¬Owns(Nono,M1) ∨ ¬Hostile(Nono)
¬Enemy(x,America) ∨ Hostile(x)
¬Hostile(Nono)
Enemy(Nono,America)
¬Enemy(Nono,America)

Resolution

- Everyone who loves all animals is loved by someone.
- Anyone who kills an animal is loved by no one.
- Jack loves all animals.
- Either Jack or Curiosity killed the cat, who is named Tuna.
- Did Curiosity kill the cat?

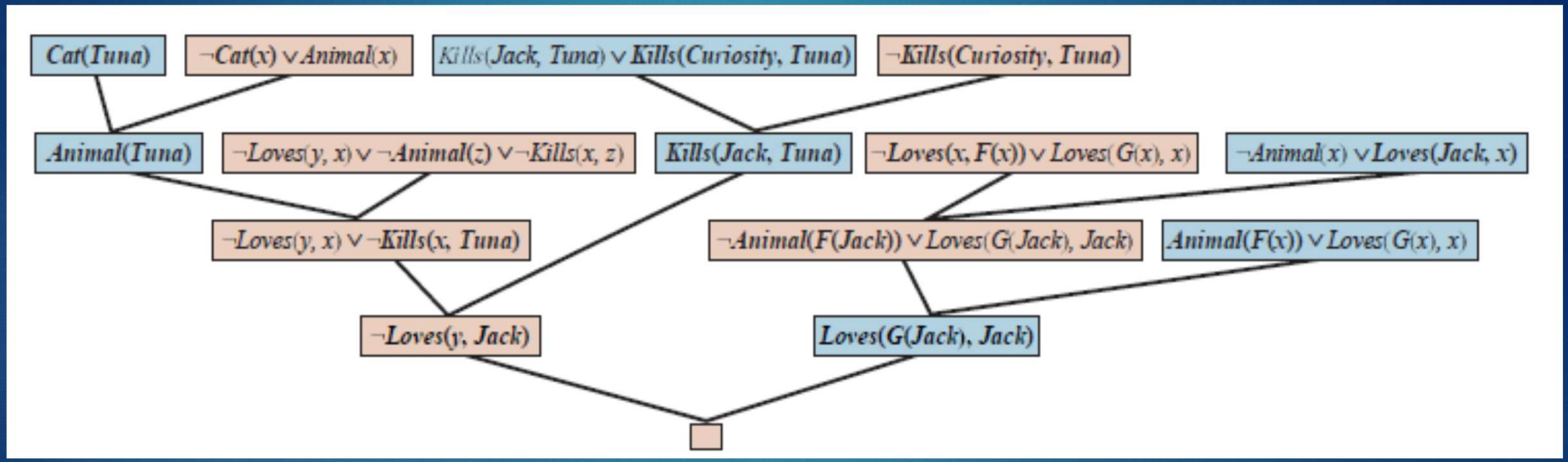
Resolution

- First, we express the original sentences, some background knowledge, and the negated goal
- G in first-order logic:
- A. $\forall x [\forall y \text{ Animal}(y) \Rightarrow \text{Loves}(x, y)] \Rightarrow [\exists y \text{ Loves}(y, x)]$
- B. $\forall x [\exists z \text{ Animal}(z) \wedge \text{Kills}(x, z)] \Rightarrow [\forall y \neg \text{Loves}(y, x)]$
- C. $\forall x \text{ Animal}(x) \Rightarrow \text{Loves}(\text{Jack}, x)$
- D. $\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})$
- E. $\text{Cat}(\text{Tuna})$
- F. $\forall x \text{ Cat}(x) \Rightarrow \text{Animal}(x)$
- $\neg$G. $\neg \text{Kills}(\text{Curiosity}, \text{Tuna})$

Resolution

- Now we apply the conversion procedure to convert each sentence to CNF:
- A1. Animal(F(x)) V Loves(G(x), x)
- A2. ¬Loves(x, F(x)) V Loves(G(x), x)
- B. ¬Loves(y, x) V ¬Animal (z) V ¬Kills(x, z)
- C. ¬Animal(x) V Loves(Jack, x)
- D. Kills(Jack, Tuna) V Kills(Curiosity, Tuna)
- E. Cat(Tuna)
- F. ¬Cat(x) V Animal (x)
- ¬G. ¬Kills(Curiosity, Tuna)

Resolution


Cat(Tuna)
¬Cat(x) ∨ Animal(x)
Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna)
¬Kills(Curiosity, Tuna)
Animal(Tuna)
¬Loves(y, x) ∨ ¬Animal(z) ∨ ¬Kills(x, z)
Kills(Jack, Tuna)
¬Loves(x, F(x)) ∨ Loves(G(x), x)
¬Animal(x) ∨ Loves(Jack, x)
¬Loves(y, x) ∨ ¬Kills(x, Tuna)
¬Animal(F(Jack)) ∨ Loves(G(Jack), Jack)
Animal(F(x)) ∨ Loves(G(x), x)
¬Loves(y, Jack)
Loves(G(Jack), Jack)

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: Example

- **Step-1: Conversion of Facts into FOL**

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

b. food(Apple) $\wedge$ food(vegetables)

c. $\forall x \forall y$: eats(x, y) $\wedge \neg$ killed(x) $\rightarrow$ food(y)

d. eats (Anil, Peanuts) $\wedge$ alive(Anil).

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

f. $\forall x$: $\neg$ killed(x) $\rightarrow$ alive(x)

g. $\forall x$: alive(x) $\rightarrow \neg$ killed(x)

} **added predicates.**

h. likes(John, Peanuts)

Resolution: Example

- **Step-2: Conversion of FOL into CNF**
  - **Eliminate all implication (→) and rewrite**

a. ∀x ¬food(x) V likes(John, x)
b. food(Apple) Λ food(vegetables)
c. ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
d. eats (Anil, Peanuts) Λ alive(Anil)
e. ∀x ¬ eats(Anil, x) V eats(Harry, x)
f. ∀x¬ [¬ killed(x)] V alive(x)
g. ∀x ¬ alive(x) V ¬ killed(x)
h. likes(John, Peanuts).

Resolution: Example

- **Move negation (¬)inwards and rewrite**∀x ¬ food(x) V likes(John, x)
  a. food(Apple) Λ food(vegetables)
  b. ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  c. eats (Anil, Peanuts) Λ alive(Anil)
  d. ∀x ¬ eats(Anil, x) V eats(Harry, x)
  e. ∀x ¬killed(x)] V alive(x)
  f. ∀x ¬ alive(x) V ¬ killed(x)
  g. likes(John, Peanuts).

Resolution: Example

- **Rename variables or standardize variables**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
  d. eats (Anil, Peanuts) Λ alive(Anil)
  e. ∀w¬ eats(Anil, w) V eats(Harry, w)
  f. ∀g ¬killed(g)] V alive(g)
  g. ∀k ¬ alive(k) V ¬ killed(k)
  h. likes(John, Peanuts).

Resolution: Example

- **Eliminate existential instantiation quantifier by elimination.**
  - In this step, we will eliminate existential quantifier ∃, and this process is known as **Skolemization**.
  - But in this example problem since there is no existential quantifier so all the statements will remain same in this step

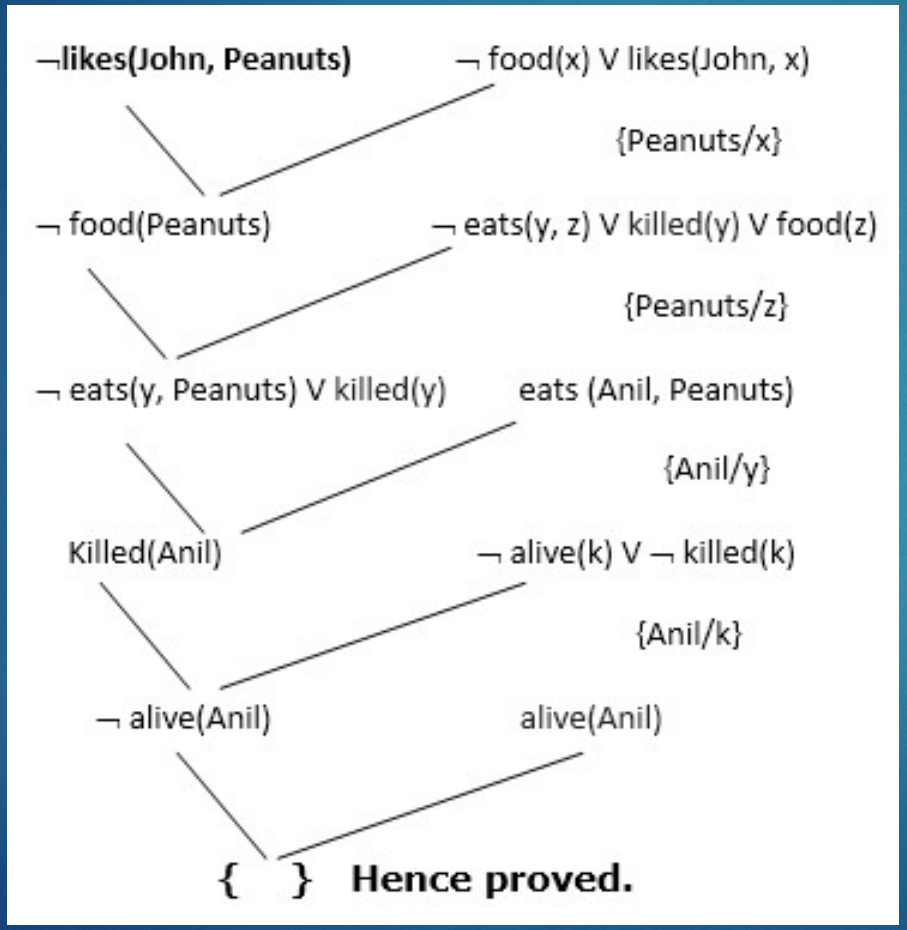
Resolution: Example

- **Drop Universal quantifiers.**
  - ¬ food(x) V likes(John, x)
  - food(Apple)
  - food(vegetables)
  - ¬ eats(y, z) V killed(y) V food(z)
  - eats (Anil, Peanuts)
  - alive(Anil)
  - ¬ eats(Anil, w) V eats(Harry, w)
  - killed(g) V alive(g)
  - ¬ alive(k) V ¬ killed(k)
  - likes(John, Peanuts).

Resolution: Example

- **Distribute conjunction ∧ over disjunction ¬.**
  This step will not make any change in this problem.
- **Step-3: 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)

Resolution: Example



Thank You!