Inference in First-Order Logic

Unit-IV

Propositional vs. First-Order Inference

Inference Rules for Quantifiers:

Suppose we have: 'All greedy kings are evil'.

$\forall x \ \text{King}(x) \wedge \text{Greedy}(x) \rightarrow \text{Evil}(x)$

Then we can infer any of the following sentences:

$$\begin{aligned}
&\mathit{King}(\mathit{John}) \wedge \mathit{Greedy}(\mathit{John}) \Rightarrow \mathit{Evil}(\mathit{John}) \\
&\mathit{King}(\mathit{Richard}) \wedge \mathit{Greedy}(\mathit{Richard}) \Rightarrow \mathit{Evil}(\mathit{Richard}) \\
&\mathit{King}(\mathit{Father}(\mathit{John})) \wedge \mathit{Greedy}(\mathit{Father}(\mathit{John})) \Rightarrow \mathit{Evil}(\mathit{Father}(\mathit{John})) . \\
&\quad \vdots
\end{aligned}$$

Propositional vs. First-Order Inference

Universal Instantiation (UI):

- This rule 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:
- $$\frac{\forall v \ \alpha}{\text{SUBST}(\{v/g\}, \alpha)}$$ for any variable 'v' and ground term 'g'.
- Every instantiation of a universally quantified sentence is entailed by it:
- **Notation:** *Subst({v/g}, α) means the result of substituting ground term g for variable v in sentence α*
- Ex: The three sentences given earlier are obtained with the substitutions:
- E.g., $\forall$x *King*(*x*) $\wedge$ *Greedy*(*x*) $\Rightarrow$ *Evil*(*x*) yields:

  King(*John*) $\wedge$ *Greedy*(*John*) $\Rightarrow$ *Evil*(*John*), *{x/John}*

  King(*Richard*) $\wedge$ *Greedy*(*Richard*) $\Rightarrow$ *Evil*(*Richard*), *{x/Richard}*

  King(*Father*(*John*)) $\wedge$ *Greedy*(*Father*(*John*)) $\Rightarrow$ *Evil*(*Father*(*John*)), *{x/Father(John)}*

Continued...

- **Existential Instantiation (EI):**
- For any sentence α, variable v and constant symbol k that does not appear elsewhere in KB,

$$\frac{\exists v \ \alpha}{\text{SUBST}(\{v/k\}, \alpha)}.$$

- For example, from the sentence ∃ x Crown(x) Λ OnHead(x, John)
- we can infer the sentence Crown(C_1) Λ OnHead(C_1, John) as long as C_1 does not appear elsewhere in the KB.
- where C_1 is a new constant symbol, called a Skolem constant
- Note: Existential and universal instantiation allows to "propositionalize" any FOL sentence or KB
  - EI produces one instantiation per EQ sentence
  - UI produces a whole set of instantiated sentences per UQ sentence

Continued...

- **Reduction to Propositional Inference:**
- It becomes *possible to reduce first-order inference to propositional inference*.
  - As an existentially quantified sentence can be replaced by one instantiation,
  - a universally quantified sentence can be replaced by the set of all possible instantiations.
- Ex: Suppose the KB is:

$$\forall x \ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John).$$

- We apply UI to the first sentence using all possible ground term substitutions from the vocabulary of the KB,
- In this case, { x / John } and { x / Richard }.
- We obtain: King(John) Λ Greedy(John) → Evil(John)
- King(Richard) Λ Greedy(Richard) → Evil(Richard)
- We discard the universally quantified sentence.
- Now the KB is essentially *propositional* if we view the ground atomic sentences -- King(John), Greedy(John) and so on as *propositional symbols*.

Unification and Lifting

- Unification and Lifting: The inference of Evil(John) from the sentences
- $\forall x$ King(x) $\wedge$ Greedy(x) $\rightarrow$ Evil(x)
- King(John)
- Greedy(John)
- seems completely obvious to a human being. How to make it completely obvious to a computer?
- A first-order inference rule:
- For this to work, we find some x such that x is a king and x is greedy.
- In this case, the substitution { x / John } achieves that aim.
- Suppose that instead of knowing Greedy(John), we know that everyone is greedy.
- $\forall y$ Greedy(y)
- Then, we would still be able to conclude that: Evil(John).
- Because we know that John is a king (given) and John is greedy (since everyone is greedy).
- i.e., the substitution is { x / John, y / John }

Continued...

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

$$\frac{p_1', \; p_2', \; \ldots, \; p_n', \; (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{\text{SUBST}(\theta, q)} .$$

- There are n atomic sentences p_i ‘ and one implication to this rule.
- The conclusion is the result of applying the substitution θ to the consequent q.
- For our example: p_1 ‘ is King(John) p_1 is King(x)
- p_2‘ is Greedy(y) p_2 is Greedy(x)
- θ is { x / John, y / John } q is Evil(x)
- SUBST(θ , q) is Evil(John)
- Generalized Modus Ponens is a lifted version of Modus Ponens.

Continued...

- **Unification:** It is a process of finding substitutions that make different logical expressions look identical.
- It takes two sentences and returns a **unifier** for them, if one exists.
- UNIFY (p, q) = θ where SUBST (θ, p) = SUBST (θ, q).
- Examples: 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**
- In the last one, the problem arises because of the usage of the same variable x in both the sentences.
- If we standardize apart one of the two sentences (that is, renaming its variables to avoid clashes),
- then we have: UNIFY (Knows(John, x), Knows(z, Elizabeth)).
- So, the substitution is { x / Elizabeth, z / John }.

Continued...

- Most General Unifier (MGU):
- There could be more than one unifier in some cases.
- Ex: UNIFY (Knows(John, x), Knows(y, z))
- could return { y / John, x / z } or { y / John, x / John, z / John }.
- The first unifier gives Knows(John, z),
- The second unifier gives Knows(John, John).
- -- could be obtained from the first by an additional substitution { z / John }.
- We say that the first unifier is more general than the second, because it places fewer restrictions on the values of the variables.
- For every unifiable pair of expressions, there is a single **Most General Unifier (MGU)**.
- **a most general unifier (MGU) is the most general set of substitutions that can be found for two expressions.**
- In this case, it is { y / John, x / z }.

Continued...

An algorithm for computing MGUs is follows:

```
function UNIFY(x, y, θ=empty) returns a substitution to make x and y identical, or failure
  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(ARGS(x), ARGS(y), UNIFY(OP(x), OP(y), θ))
  else if LIST?(x) and LIST?(y) then
      return UNIFY(REST(x), REST(y), UNIFY(FIRST(x), FIRST(y), θ))
  else return failure

function UNIFY-VAR(var, x, θ) returns a substitution
  if {var/val} ∈ θ for some val then return UNIFY(val, x, θ)
  else if {x/val} ∈ θ for some val then return UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) then return failure
  else return add {var/x} to θ
```

Figure 9.1 The unification algorithm. The arguments x and y can be any expression: a constant or variable, or a compound expression such as a complex sentence or term, or a list of expressions. The argument θ is a substitution, initially the empty substitution, but with $\{var/val\}$ pairs added to it as we recurse through the inputs, comparing the expressions element by element. In a compound expression such as $F(A,B)$, OP(x) field picks out the function symbol F and ARGS(x) field picks out the argument list (A,B).

Continued...

Explanation:

1. If x and y are both variables or constants, if x and y are identical, then return NIL.

   θ is empty or θ is as it is.

   Ex: King(John), King(John) or King(x), King(x)

2. If x is a variable and y is a constant, then return constant for variable

   Ex: King(x), King(John) return { x / John }

3. If x is a constant and y is a variable, then return constant for variable

   Ex: Evil(Richard), Evil(x) return { x / Richard }

4. If x and y are COMPOUND expressions, call UNIFY with

   first arguments of x and y,

   second arguments of x and y,

   …….

   nth arguments of x and y.

Continued...

Ex: Father(m, David, Bill) (m is the father of David and Bill)

(args[x] : m, David and Bill)

Father(Taylor, p, Bill) (Taylor is the father of p and Bill)

(args[y] : Taylor, p and Bill)

θ is { m / Taylor, p / David, NIL }

5. If x is a list and y is a list, then

UNIFY First elements of x and y

and then Rest of the elements of x and y.

Ex: [m, David, Bill] [Taylor, p, Bill]

first rest first rest

θ is { m / Taylor, p / David, NIL }

8. OCCUR-CHECK (if a variable itself occurs inside a complex term)

Ex: F(x, x), F(G(x), G(x))

If G(x) is written for x, we have: F(G(x), G(x)), F(G(G(x)), G(G(x)))-- never possible to eliminate x

Continued...

Algorithm: Unify(L1, L2) (From: AI – Elaine Rich & Kevin Knight)

1. If L1 or L2 are both variables or constants, then
   (a) If L1 and L2 are identical, then return NIL.
   (b) Else if L1 is a variable, then if L1 occurs in L2 then return { FAIL }, else return (L2 / L1) (return L2 for L1)
   (c) Else if L2 is a variable, then if L2 occurs in L1 then return { FAIL }, else return (L1 / L2) (return L1 for L2)
   (d) Else return FAIL.
2. If the initial predicate symbols in L1 and L2 are not identical, then return { FAIL }.
3. If L1 and L2 have a different number of arguments, then return { FAIL }.
4. Set SUBST to NIL. (At the end of the procedure, SUBST will contain all the substitutions used to unify L1 and L2.)
5. For i← 1 to number of arguments in L1:
   (a) Call Unify with the i^{th} argument of L1 and i^{th} argument of L2, putting result in S.
   (b) If S contains FAIL, then return { FAIL }.
   (c) If S is not equal to NIL, then
      (i) Apply S to the remainder of both L1 and L2.
      (ii) SUBST := APPEND(S, SUBST)
6. Return SUBST.

Continued...

Explanation:

	Step	Algorithm returns	
1.(a)	Teacher(x), ¬Teacher(x) or Teacher(Pradeep), ¬Teacher(Pradeep)	NIL	
1.(b)	f(x, x), f(g(x), g(x))	FAIL	
	(if we write g(x) for x, then it will be: f(g(x), g(x)), f(g(g(x)), g(g(x))), it will never be possible to eliminate x (OCCUR_CHECK))		
	f(x), f(John)	{ x / John }	
	f(x), f(g(y))	{ x / g(y) }	
1.(c)	f(g(x)), f(x)	FAIL	(OCCUR_CHECK, same as above)
	f(John), f(z)	{ z / John }	
1.(d)	Return FAIL.		

Continued...

Step	Algorithm returns
2. Man(John), Teacher(Bob)	FAIL
3. StudentOf(Lalitha, Btech), Student(Lalitha)	FAIL
4. SUBST = NIL	
5. FounderOf(BillGates, Microsoft), FounderOf(x, y)	S = { x / BillGates }
	S = { x / BillGates, y / Microsoft }
6. Return SUBST.	

Storage and Retrieval:

(primitive functions underlying TELL and ASK)

STORE (s) -- stores a sentence s into the KB

FETCH(q) -- returns all unifiers such that the query q unifies with some sentence in the KB

Ex: Knows(John, x) -- a query, an instance of fetching

(finds all facts that unify with Knows(John, x)

Forward Chaining

(i) First-Order Definite Clauses:

- 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.

  Ex: $\forall x \ \text{King}(x) \wedge \text{Greedy}(x) \rightarrow \text{Evil}(x)$

  $\text{King}(\text{John})$

  $\text{Greedy}(y)$

- Unlike propositional literals, first-order literals can include variables in which case, those variables are assumed to be universally quantified. (usually, we omit them)
- Many of the KBs can be converted into a set of definite clauses.
- Example: '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.'
- We will prove that 'Colonel West is a Criminal'.

Continued...

Sentences in FOL:

1. It is a crime for an American to sell weapons to hostile nations.

   American(x) Λ Weapon(y) Λ Sells(x, y, z) Λ Hostile(z) → Criminal(x)

✓ Nono has some missiles.

   ∃x Owns(Nono, x) Λ Missile(x) is transformed into two definite clauses.

2. Owns(Nono, M_1)

3. Missile(M_1)

4. All of its missiles were sold to it by Colonel West.

   ∀x Missile(x) Λ Owns(Nono, x) → Sells(West, x, Nono)

5. Missiles are weapons.

   Missile(x) → Weapon(x)

6. An enemy of America counts as “hostile”.

   Enemy(x, America) → Hostile(x)

Continued...

7. West is an American.

   American(West)

8. The country Nono is an enemy of America.

   Enemy(Nono, America)

- *Forward chaining* considers atomic sentences and tries to satisfy the premises of rules.
- This leads to inference of new sentences and thereby proof of a goal.
- To prove 'West is Criminal', all of the premises of 1 have to be satisfied.

- Use 3 in 4 and infer:

9. Sells(West, M_1, Nono)

- Use 3 in 5 and infer:

10. Weapon(M_1)

Use 8 in 6 and infer:

11. Hostile(Nono)

- Now use 7, 10, 9 and 11 in 1 and infer:

12. Criminal(West)

- Hence, 'West is Criminal' is proved.
- The substitutions are: { x / West, y / M_1 , z / Nono }

Continued...

(ii) 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

  while *true* **do**
    new $\leftarrow \{\}$ // *The set of new sentences inferred on each iteration*
    for each *rule* **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(*rule*)
      for each θ such that SUBST($\theta, p_1 \wedge \ldots \wedge p_n$) = SUBST($\theta, p'_1 \wedge \ldots \wedge p'_n$)
          for some $p'_1, \ldots, p'_n$ in KB
        $q' \leftarrow$ SUBST(θ, q)
        if q' does not unify with some sentence already in KB or *new* **then**
          add q' to *new*
          $\phi \leftarrow$ UNIFY(q', α)
          if ϕ is not *failure* **then return** ϕ
    if *new* $= \{\}$ **then return** *false*
    add *new* to KB

Figure 9.3 A conceptually straightforward, but inefficient, forward-chaining algorithm. On each iteration, it adds to KB all the atomic sentences that can be inferred in one step from the implication sentences and the atomic sentences already in KB. The function STANDARDIZE-VARIABLES replaces all variables in its arguments with new ones that have not been used before.

Continued...

Explanation:

- Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts.
- The process repeats until the query is answered or new facts are added.

A fact is not new if it is just renaming of a known fact.

Ex: Likes(x, IceCream) and Likes(y, IceCream) are renaming of each other.

Their meanings are identical (everyone likes ice cream).

FOL-FC-ASK is:

- Sound, because every inference is just an application of Generalized Modus Ponens, which is sound.
- Complete for definite clause KBs; i.e., it answers every query whose answers are entailed by any KB of definite clauses.

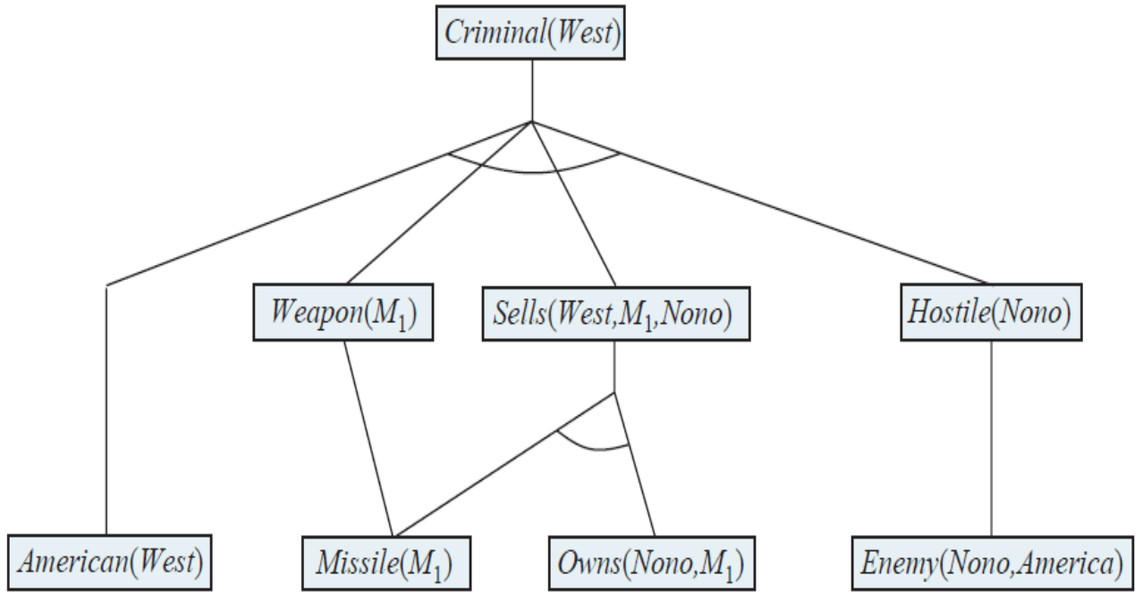


Figure 9.4 The proof tree generated by forward chaining on the crime example. The initial facts appear at the bottom level, facts inferred on the first iteration in the middle level, and facts inferred on the second iteration at the top level.

- Forward chaining -- from bottom to top in the diagram
- Bottom level -- initial facts
- Middle level -- facts inferred on the first iteration
- Top level -- facts inferred on the second iteration

1. It is a crime for an American to sell weapons to hostile nations.
   - American(x) ∧ Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) → Criminal(x)

✓Nono has some missiles.

∃x Owns(Nono, x) ∧ Missile(x) is transformed into two definite clauses.

2. Owns(Nono, M_1)
3. Missile(M_1)
4. All of its missiles were sold to it by Colonel West.

∀x Missile(x) ∧ Owns(Nono, x) → Sells(West, x, Nono)

5. Missiles are weapons.

   Missile(x) → Weapon(x)

6. An enemy of America counts as "hostile".

   Enemy(x, America) → Hostile(x)

7. West is an American.

   American(West)

8. The country Nono is an enemy of America.

   Enemy(Nono, America)

Backward Chaining

Algorithm:

```
function FOL-BC-ASK(KB, query) returns a generator of substitutions
  return FOL-BC-OR(KB, query, { })

function FOL-BC-OR(KB, goal, θ) returns a substitution
  for each rule in FETCH-RULES-FOR-GOAL(KB, goal) do
    (lhs ⇒ rhs) ← STANDARDIZE-VARIABLES(rule)
    for each θ′ in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do
      yield θ′

function FOL-BC-AND(KB, goals, θ) returns a substitution
  if θ = failure then return
  else if LENGTH(goals) = 0 then yield θ
  else
    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″
```

Figure 9.6 A simple backward-chaining algorithm for first-order knowledge bases.

Continued...

Explanation:

- The algorithm uses **compositions** of substitutions.
- Compose(θ_1, θ_2) is the substitution whose effect is identical to the effect of applying each substitution in turn.
- That is,

  SUBST(Compose(θ_1, θ_2) , p) = SUBST(θ_2, SUBST(θ_1, p))

- In the algorithm, the current variable bindings, which are stored in θ, are composed with the bindings resulting from unifying the goal with the clause head, giving a new set of current bindings for the recursive call.

Proof Tree: To prove that 'West is Criminal'.

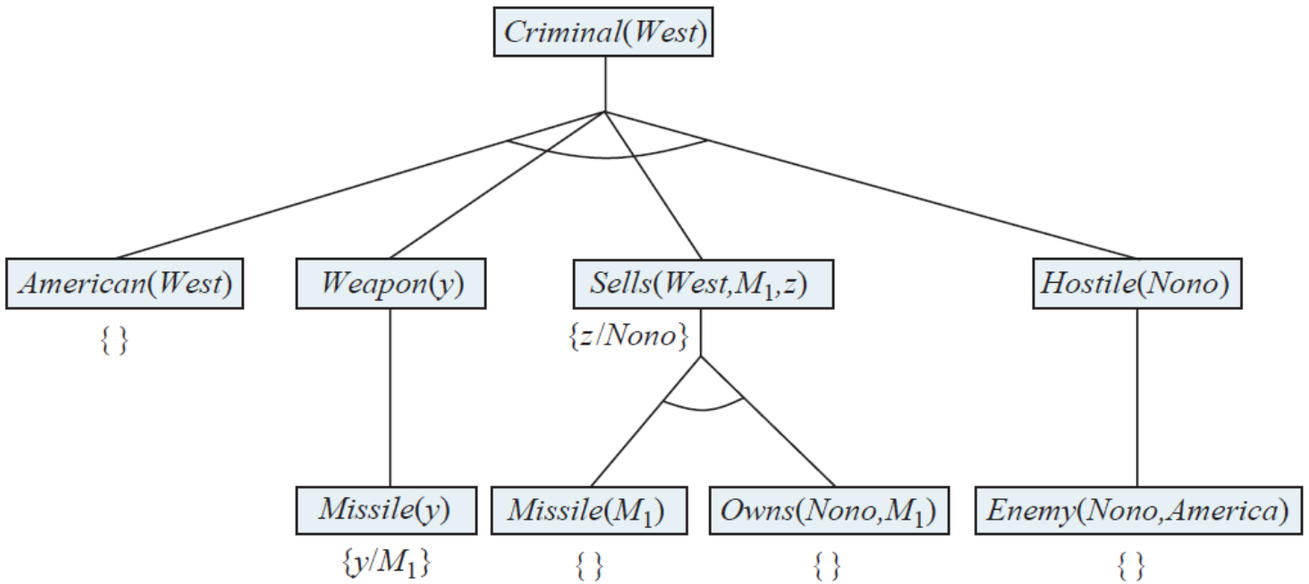


Figure 9.7 Proof tree constructed by backward chaining to prove that West is a criminal. The tree should be read depth first, left to right. To prove *Criminal*(*West*), we have to prove the four conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. Bindings for each successful unification are shown next to the corresponding subgoal. Note that once one subgoal in a conjunction succeeds, its substitution is applied to subsequent subgoals. Thus, by the time FOL-BC-ASK gets to the last conjunct, originally *Hostile*(*z*), *z* is already bound to *Nono*.

1. It is a crime for an American to sell weapons to hostile nations.
   - American(x) ∧ Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) → Criminal(x)

✓Nono has some missiles.

∃x Owns(Nono, x) ∧ Missile(x) is transformed into two definite clauses.

2. Owns(Nono, M_1)
3. Missile(M_1)
4. All of its missiles were sold to it by Colonel West.

∀x Missile(x) ∧ Owns(Nono, x) → Sells(West, x, Nono)

5. Missiles are weapons.

Missile(x) → Weapon(x)

6. An enemy of America counts as "hostile".

Enemy(x, America) → Hostile(x)

7. West is an American.

American(West)

8. The country Nono is an enemy of America.

Enemy(Nono, America)

Continued...

- Backward chaining -- from top to bottom in the diagram
- Middle and Bottom level -- read from left to right
- To prove 'Criminal(West)' -- prove the conjuncts (four) in the middle level
- To prove any middle-level goal -- go down to its next lower level

General Examples for Forward and Backward Chaining:

Consider:

1. A Λ B Λ C → Z
2. M → K
3. A
4. A Λ K → B
5. Y → M
6. Y
7. C

Prove Z.

Forward Chaining

(5) (2) (combine A and use 4)

A (given, proved) → Y (given, proved) → M → K →

A Λ K → B (proved) → C (given, proved) → Z (proved)

(4) Consider 7 from 1

Backward Chaining

(1) (3) (7) (4) (3) (2) (5)

Z ← A Λ B Λ C ← B Λ C ← B ← A Λ K ← K ← M ← Y ← Y proved from 6.

Resolution

- The resolution rule for first-order clauses is simply a lifted version of the propositional resolution Rule.
- Two clauses, which are assumed to be standardized apart so that they share no variables, can be resolved if they contain complementary literals.
- Propositional literals are complementary if one is the negation of the other; first-order literals are complementary if one unifies with the negation of the other.
- Thus, we have

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\text{SUBST}(\theta, \ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)}$$

where $\text{UNIFY}(\ell_i, \neg m_j) = \theta$. For example, we can resolve the two clauses

$$[Animal(F(x)) \vee Loves(G(x), x)] \quad \text{and} \quad [\neg Loves(u, v) \vee \neg Kills(u, v)]$$

by eliminating the complementary literals $Loves(G(x), x)$ and $\neg Loves(u, v)$, with the unifier $\theta = \{u/G(x), v/x\}$, to produce the **resolvent** clause

$$[Animal(F(x)) \vee \neg Kills(G(x), x)] .$$

Continued...

- This rule is called the binary resolution rule because it resolves exactly two literals.
- The binary resolution rule by itself does not yield a complete inference procedure.
- The full resolution rule resolves subsets of literals in each clause that are unifiable.
- An alternative approach is to extend factoring—the removal of redundant literals—to the first-order case.
- Propositional factoring reduces two literals to one if they are identical; first-order factoring reduces two literals to one if they are unifiable. The unifier must be applied to the entire clause.
- The combination of binary resolution and factoring is complete.
- Resolution proves that KB $\models \alpha$, by proving that KB $\wedge \neg \alpha$ unsatisfiable—that is, by deriving the empty clause.
- The algorithmic approach is identical to the propositional case.

Resolution…

- Resolution method in FOPL is an uplifted version of propositional resolution method.
- **In FOPL, the process to apply the resolution method is as follows:**
  - Conversion of facts into first-order logic.
  - Convert FOL statements into CNF
  - Negate the statement which needs to prove (proof by contradiction)
  - Draw resolution graph (unification).

Conversion from FOL to Conjunctive Normal Form (CNF):

1. Eliminate all implication (→) and rewrite
2. Move negation (¬)inwards and rewrite
3. Rename variables or standardize variables to unique variable.
4. Move all quantifiers to the left without changing their relative order.
5. Eliminate existential instantiation quantifier by elimination ∃ (Skolemization).
6. Drop Universal quantifiers ∀.
7. Convert the formula into a conjunction of disjuncts.

Resolution

- Conjunctive Normal Form (CNF):
- As in the propositional case, first-order resolution requires that sentences be in CNF -- that is a conjunction of clauses, where each clause is a disjunction of literals.
- Literals can contain variables, which are assumed to be universally quantified.

$$\forall x, y, z \; American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

becomes, in CNF,

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x) .$$

- *Note that an every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence.*
- Procedure to convert FOL into CNF :
- Ex: Everyone who loves all animals is loved by someone.

$$\forall x \; [\forall y \; Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \; Loves(y, x)] .$$

-

Continued...

The steps are as follows:

1. Eliminate Implications/biconditional:

$$\forall x\ \neg[\forall y\ Animal(y) \Rightarrow Loves(x,y)] \vee [\exists y\ Loves(y,x)]$$
$$\forall x\ \neg[\forall y\ \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\ Loves(y,x)]\,.$$

2. Move ¬ inwards: We have

$$\neg\forall x\ p \quad \text{becomes} \quad \exists x\ \neg p$$
$$\neg\exists x\ p \quad \text{becomes} \quad \forall x\ \neg p\,.$$

Our sentence goes through the following transformations:

$$\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ Loves(y,x)]\,.$$
$$\forall x\ [\exists y\ \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]\,.$$
$$\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]\,.$$

Continued...

3. Standardize variables:

Sentences like (∀x P(x)) V (∃x Q(x)) can be written as (∀x P(x)) V (∃y Q(y))

Thus we have,

$$\forall x \ [\exists y \ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z \ Loves(z,x)] .$$

4. Skolemize:

- *Skolemization* is the process of removing existential quantifiers by elimination.
- ∃x P(x) can be written as P(A) where A is a new constant.
- So, our sentence becomes: $\forall x \ [Animal(A) \wedge \neg Loves(x,A)] \vee Loves(B,x)$

- 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'.
- In fact, our original sentence allows – 'each person to fail to love a different animal or to be loved by a different person'.
-

Continued...

Thus, we want the Skolem entities to depend on x. A better way is:

$$\forall x \ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x) .$$

(F and G are Skolem Functions)

General Rule: The arguments of the Skolem function are all the universally quantified variables in whose scope the existential quantifier appears.

5. Drop Universal quantifiers:

At this point, all remaining variables must be universally quantified. We can drop the universal quantifiers.

$$[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x) .$$

6. Distribute Λ over V:

$$[Animal(F(x)) \vee Loves(G(x), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(x), x)] .$$

The sentence is now in CNF and consists of two clauses.

Continued...

The sentences (in Slides 17 & 18) :

1. American(x) Λ Weapon(y) Λ Sells(x, y, z) Λ Hostile(z) → Criminal(x)
2. Owns(Nono, M_1)
3. Missile(M_1)
4. Missile(x) Λ Owns(Nono, x) → Sells(West, x, Nono)
5. Missile(x) → Weapon(x)
6. Enemy(x, America) → Hostile(x)
7. American(West)
8. Enemy(Nono, America)

Continued...

The sentences in CNF are

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x,y,z) \vee \neg Hostile(z) \vee Criminal(x)$$
$$\neg Missile(x) \vee \neg Owns(Nono,x) \vee Sells(West,x,Nono)$$
$$\neg Enemy(x,America) \vee Hostile(x)$$
$$\neg Missile(x) \vee Weapon(x)$$
$$Owns(Nono,M_1) \qquad Missile(M_1)$$
$$American(West) \qquad Enemy(Nono,America)\,.$$

Continued...

The sentences in Clause Form:

1. $\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x)$

2. Owns(Nono, M_1)
3. Missile(M_1)

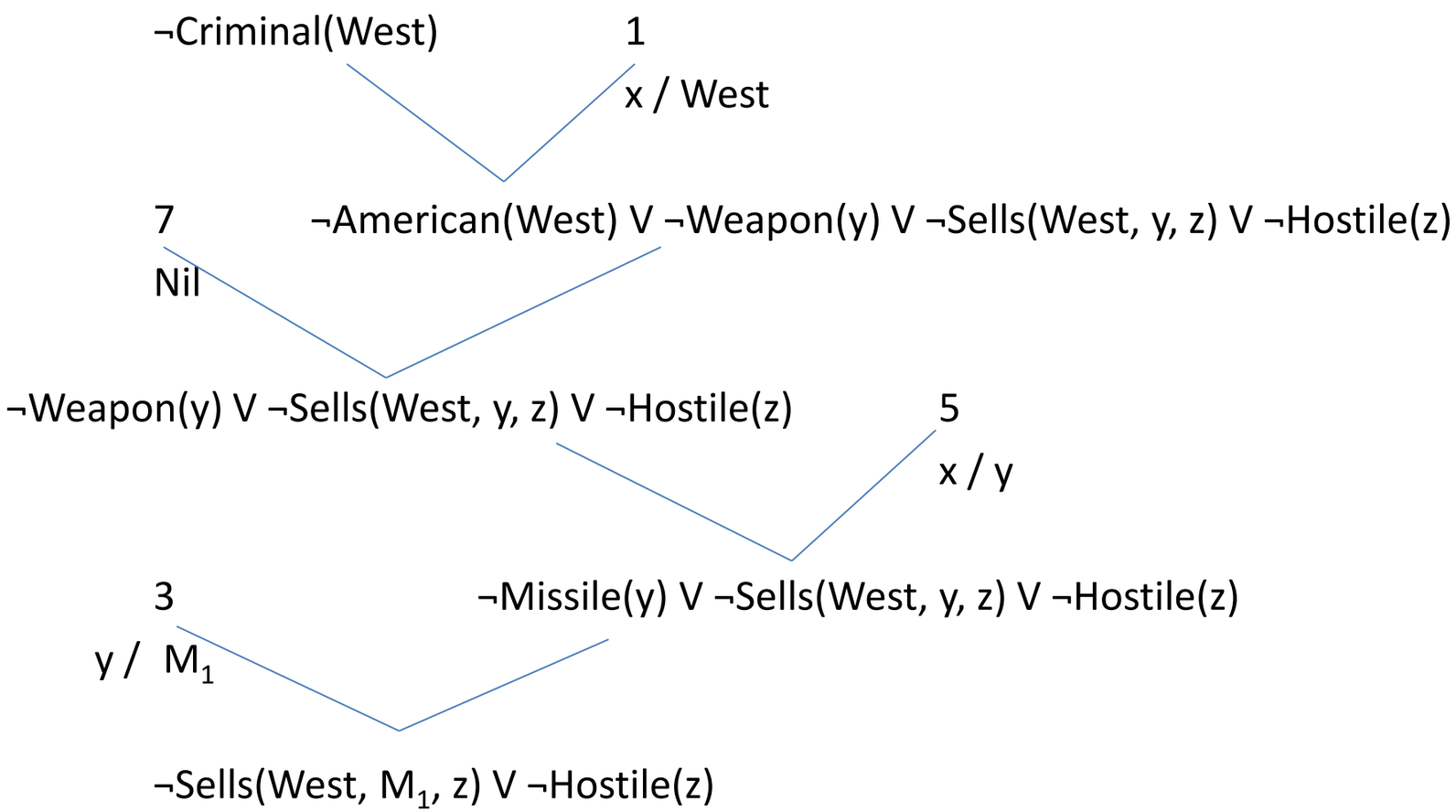
4. $\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$.

5. $\neg Missile(x) \vee Weapon(x)$.
6. $\neg Enemy(x, America) \vee Hostile(x)$.

7. American(West)
8. Enemy(Nono, America)
9. ¬Criminal(West) (Goal is negated [refutation], converted to a clause and added)

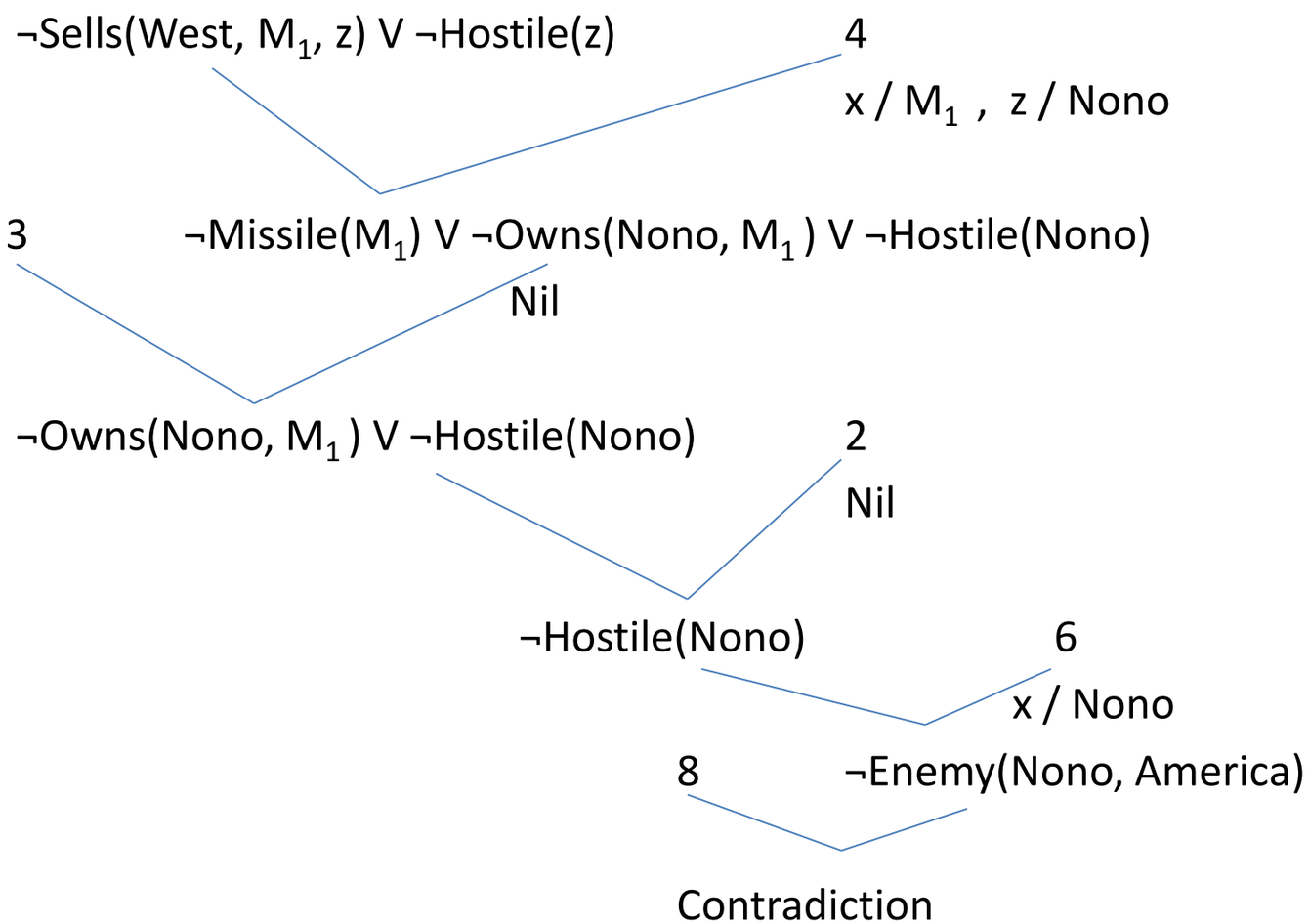
Continued...

Proof:



Continued...

Proof Continued:



Hence, 'West is Criminal' is proved.

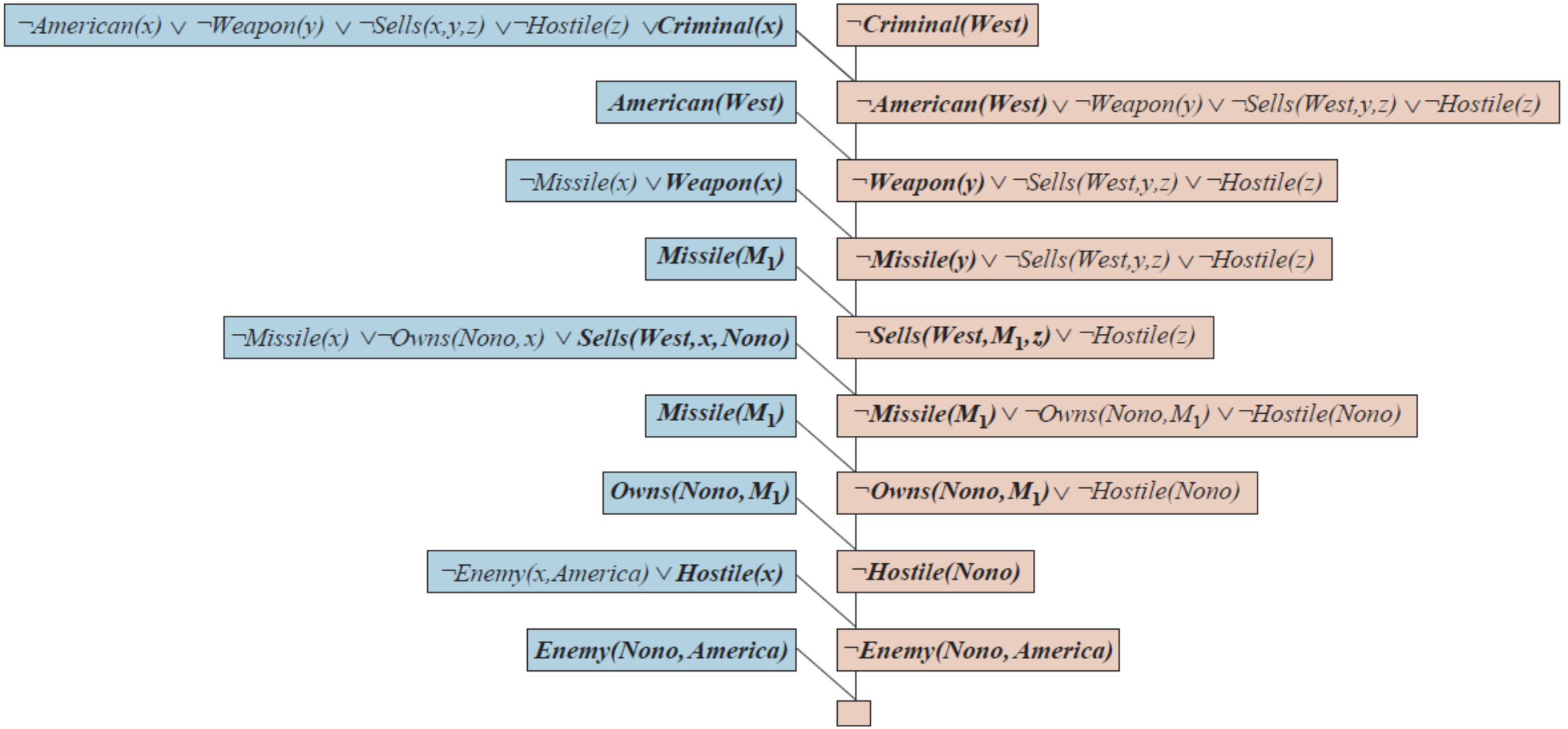


Figure 9.10 A resolution proof that West is a criminal. At each resolution step, the literals that unify are in bold and the clause with the positive literal is shaded blue.

- Example:
  - 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?
- The original sentences, and the negated goal G in first-order logic:

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$

B. $\forall x\ [\exists z\ Animal(z) \wedge Kills(x,z)] \Rightarrow [\forall y\ \neg Loves(y,x)]$

C. $\forall x\ Animal(x) \Rightarrow Loves(Jack,x)$

D. $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$

E. $Cat(Tuna)$

F. $\forall x\ Cat(x) \Rightarrow Animal(x)$

¬G. $\neg Kills(Curiosity,Tuna)$

- In CNF

A1. $Animal(F(x)) \vee Loves(G(x),x)$

A2. $\neg Loves(x,F(x)) \vee Loves(G(x),x)$

B. $\neg Loves(y,x) \vee \neg Animal(z) \vee \neg Kills(x,z)$

C. $\neg Animal(x) \vee Loves(Jack,x)$

D. $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$

E. $Cat(Tuna)$

F. $\neg Cat(x) \vee Animal(x)$

¬G. $\neg Kills(Curiosity,Tuna)$

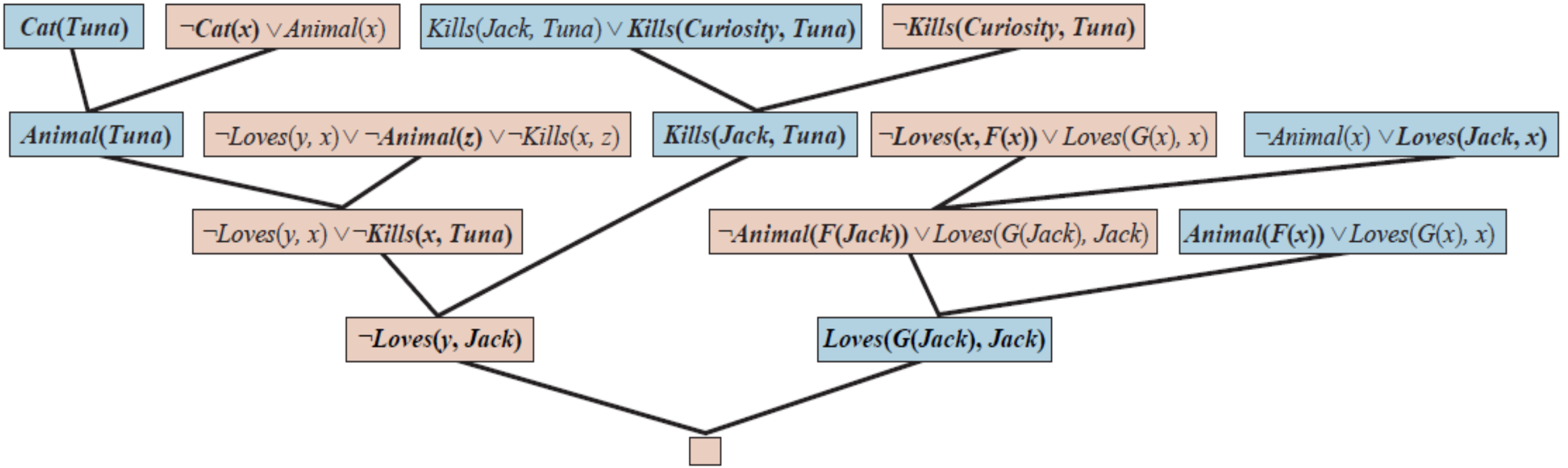


Figure 9.11 A resolution proof that Curiosity killed the cat. Notice the use of factoring in the derivation of the clause $Loves(G(Jack), Jack)$. Notice also in the upper right, the unification of $Loves(x, F(x))$ and $Loves(Jack, x)$ can only succeed after the variables have been standardized apart.

Continued...

Problem 1: From Exercise of E. Rich and Kevin Knight

Consider the following sentences:

1. John likes all kinds of food.
2. Apples are food.
3. Chicken is food.
4. Anything anyone eats and isn't killed by is food.
5. Bill eats peanuts and is still alive.
6. Sue eats everything Bill eats.

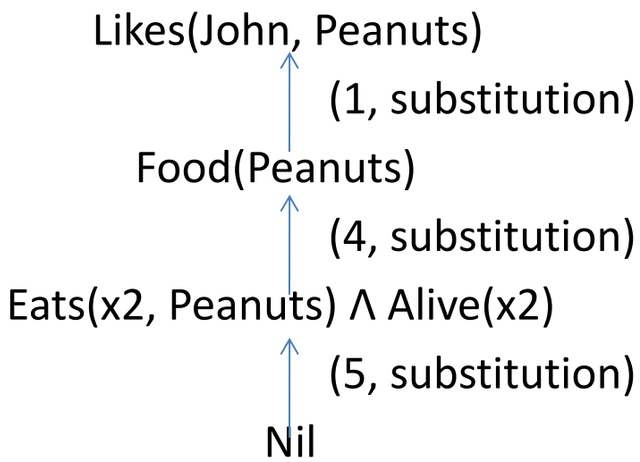
Prove that John likes Peanuts

(a) FOL (Predicate logic):

1. ∀x1 Food(x1) → Likes(John, x1)
2. Food(Apples)
3. Food(Chicken)
4. ∀x2, ∀ y Eats(x2, y) Λ Alive(x2) → Food(y)
5. Eats(Bill, Peanuts) Λ Alive(Bill)
6. ∀x3 Eats(Bill, x3) → Eats(Sue, x3)

Continued...

(b) Prove that ‘John likes peanuts’ using backward chaining



Continued...

(c) Convert the formulas of part (a) into Clause form

1. ¬Food(x1) V likes(John, x1)

2. Food(Apples)

3. Food(Chicken)

4. ¬Eats(x2, y) V ¬Alive(x2) V Food(y)

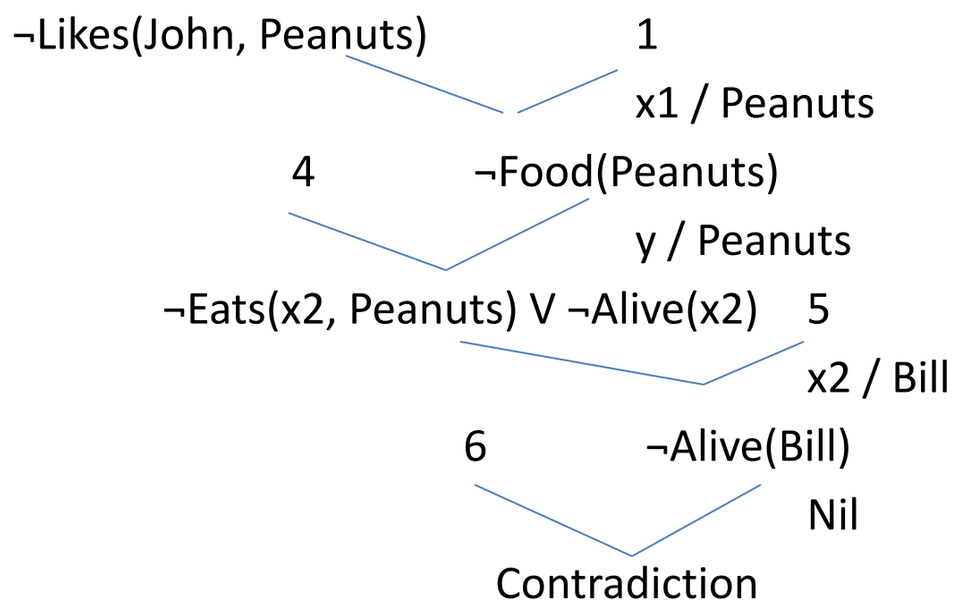
5. Eats(Bill, Peanuts)

6. Alive(Bill)

7. ¬Eats(Bill, x3) V Eats(Sue, x3)

Continued...

(d) Prove that ‘John likes peanuts’ using resolution.



So, ‘John likes peanuts’ is proved.

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.**

Step-1: Conversion of Facts into FOL

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

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

b. food(Apple) ∧ food(vegetables)

c. $\forall x \forall y$: eats(x, y) ∧ ¬ killed(x) → food(y)

d. eats (Anil, Peanuts) ∧ alive(Anil).

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

f. $\forall x$: ¬ killed(x) → alive(x) } **added predicates.**

g. $\forall x$: alive(x) →¬ killed(x) }

h. likes(John, Peanuts)

Step-2: 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.

- **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).

- **Move negation (¬)inwards and rewrite**
  a. ∀x ¬ food(x) V likes(John, x)
  b. food(Apple) Λ food(vegetables)
  c. ∀x ∀y ¬ eats(x, y) V 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).

- **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).

- **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.

- **Drop Universal quantifiers.**
  In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.

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).

- **Distribute conjunction ∧ over disjunction ¬.**

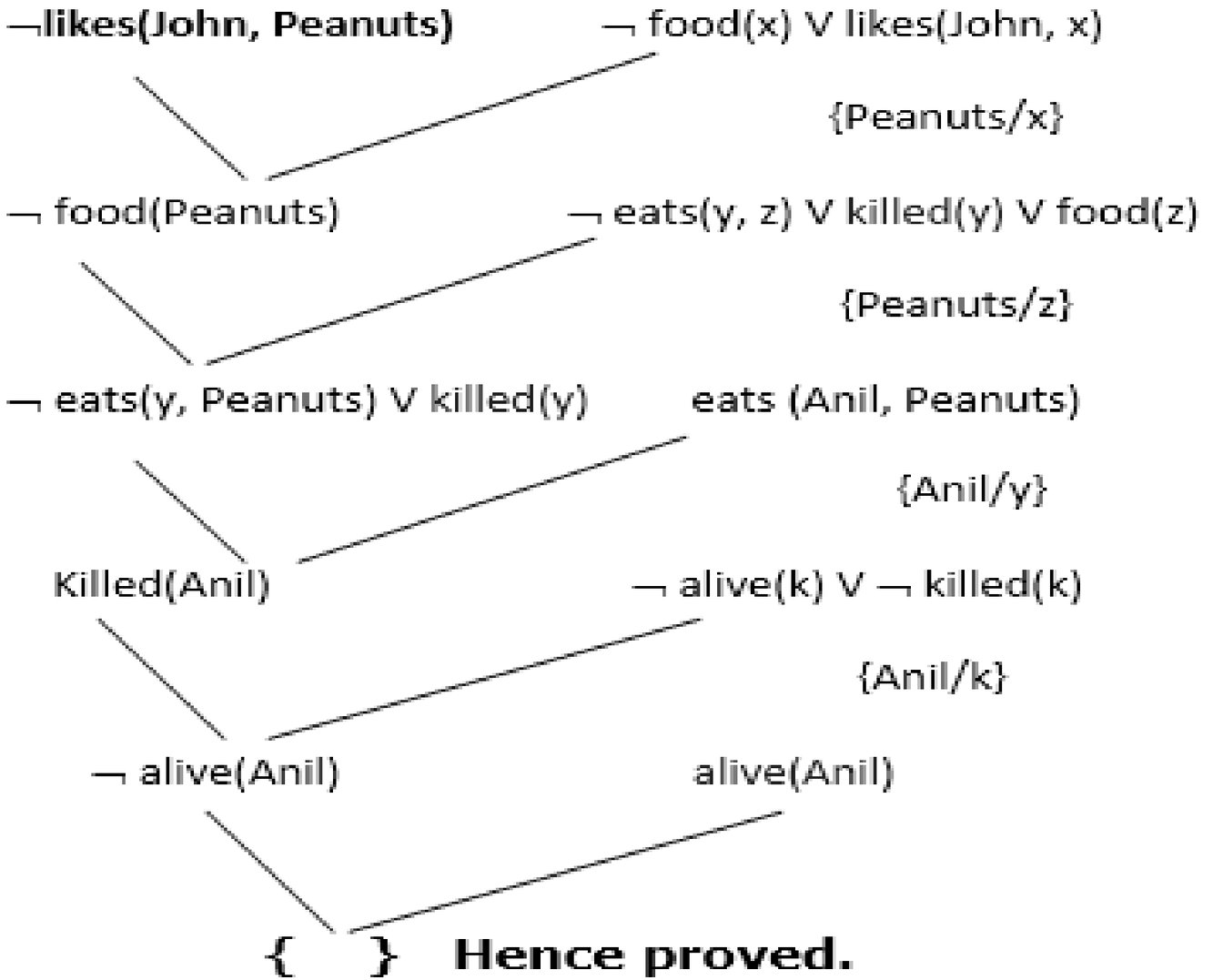
  This step will not make any change in this problem.

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)

Step-4: 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:


¬likes(John, Peanuts)
¬ food(x) V likes(John, x)
{Peanuts/x}
¬ food(Peanuts)
¬ eats(y, z) V killed(y) V food(z)
{Peanuts/z}
¬ eats(y, Peanuts) V killed(y)
eats (Anil, Peanuts)
{Anil/y}
Killed(Anil)
¬ alive(k) V ¬ killed(k)
{Anil/k}
¬ alive(Anil)
alive(Anil)
{ } Hence proved.

Continued...

Problem 2:

Given the following information for a database:

1. If x is on top of y, y supports x.
2. If x is above y and they are touching each other, x is on top of y.
3. A cup is above a book.
4. A cup is touching a book.

   Translate these statements into clausal form. Show that 'Supports(Book, Cup)' is true using resolution.

Problem 3:

Represent the following facts in first-order logic and convert them into clause form.

Use resolution to find that 'Ravi is the spy'.

1. One of Raman, Ravi, Raghu and Ramesh is the spy.
2. Raman is not the spy.
3. Spies wear light coloured dresses and do not attract attention of others.
4. Raghu was wearing a dark coloured suit.
5. Ramesh was the centre of attention on that evening.