Module 5

Inferences in FOL and Classical Planning

Contents

1. Inference in First Order Logic :

- Backward Chaining,
- Resolution

2. Classical Planning:

- Definition of Classical Planning,
- Algorithms for Planning as State-Space Search,
- Planning Graphs Inference in First Order Logic:

5.1.1Backward Chaining

- Backward chaining is a reasoning method that **starts with the goal and works backward through the inference rules** to find out whether the goal can be satisfied by the known facts.
- It's essentially **goal-driven reasoning,** where the system seeks to prove the hypothesis by breaking it down into **subgoals and verifying if the premises support them**.

Example

Consider the following knowledge base representing a simple diagnostic system:

1. ***If a patient has a fever, it might be a cold.***
2. ***If a patient has a sore throat, it might be strep throat.***
3. ***If a patient has a fever and a sore throat, they should see a doctor.***

Given the facts:

- The patient has a **fever.**
- The patient has a **sore throat.**

Backward chaining would proceed as follows:

1. **Start with the goal**: **Should the patient see a doctor?**
2. **Check the third rule**: Does the patient have a cold and a sore throat? **Yes.**
3. **Check the first and second rules**: Does the patient have a fever and sore throat? **Yes**.
4. **The goal is satisfied**: **The patient should see a doctor**.

Backward Chaining : Algorithm

These algorithms work backward from the goal, chaining through rules to find known facts that support the proof.

```
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″
```

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

Overview of the Algorithm

Goal: The purpose of the algorithm is to determine whether a query (goal) can be derived from a given knowledge base (KB).

Process:

- It uses **backward chaining**, meaning it starts with the goal and works backward by looking **for rules or facts in the knowledge** base that could satisfy the goal.
- The algorithm **returns substitutions (values or variables) that make the query true.**

Key Components:

- **FOL-BC-ASK**: This is the main function that starts the backward-chaining process by calling **`FOL-BC-OR`.**
- **FOL-BC-OR**: This function checks whether the goal can be satisfied by any rule in the KB. It iterates over applicable rules and tries to unify the goal with the rule's conclusions.
- **FOL-BC-AND**: This function handles multiple sub-goals. It ensures that all sub-goals are satisfied for the main goal to be true.

Key Terms Used

- **FOL-BC-ASK:** Entry point for the algorithm.
- **FOL-BC-OR**: Handles rules and checks if the goal is satisfied by any rule.
- **FOL-BC-AND:** Ensures all sub-goals are satisfied.
- **FETCH-RULES-FOR-GOAL**: Retrieves applicable rules for a goal.
- **UNIFY:** Matches terms by finding substitutions.
- **Standardize Variables**: Ensures variable names are unique to avoid conflict.
- **θ**: The substitution carried into the current function call.
- **θ′**: A substitution produced by solving the first sub-goal in FOL-BC-AND.
- **θ″:** A substitution produced by solving the remaining sub-goals using the updated θ′.

Key Takeaways

- Backward chaining focuses on **proving the goal by breaking** it into **smaller sub-goals** and matching them to rules in the KB.
- It uses **unification and substitutions** to ensure variable consistency.
- It **recursively checks all rules** until the **query is satisfied** or fails.

Step-by-Step Explanation with Example

Knowledge Base:

1. `Parent(x, y) ⇒ Ancestor(x, y)` (Rule 1)
2. `Parent(x, z) ∧ Ancestor(z, y) ⇒ Ancestor(x, y)` (Rule 2)
3. `Parent(John, Mary)` (Fact)
4. `Parent(Mary, Sam)` (Fact)

Query: `Ancestor(John, Sam)`

Execution Steps

Step 1: FOL-BC-ASK

- Query: `Ancestor(John, Sam)`
- Calls: `FOL-BC-OR(KB, Ancestor(John, Sam), { })`.

Step 2: FOL-BC-OR

- Goal: `Ancestor(John, Sam)`
- Fetch rules for `Ancestor`:
    1. Rule 1: `Parent(x, y) ⇒ Ancestor(x, y)`
    2. Rule 2: `Parent(x, z) ∧ Ancestor(z, y) ⇒ Ancestor(x, y)`

Case 1: Use Rule 1

- `lhs = Parent(x, y)`, `rhs = Ancestor(x, y)`.
- Unify `Ancestor(John, Sam)` with `Ancestor(x, y)`:
    - Substitution: `θ = {x=John, y=Sam}`.
- Sub-goal: `Parent(John, Sam)`.

Step 3: FOL-BC-AND

- Goals: `[Parent(John, Sam)]`
- Calls: `FOL-BC-OR(KB, Parent(John, Sam), {x=John, y=Sam})`.

Step 4: FOL-BC-OR

- Goal: `Parent(John, Sam)`
- Check the KB:
  - Facts: `Parent(John, Mary)` (no match for `Sam`).
  - Rule 1 fails.

Step 4: FOL-BC-OR

- Goal: `Parent(John, Sam)`
- Check the KB:
  - Facts: `Parent(John, Mary)` (no match for `Sam`).
  - Rule 1 fails.

Case 2: Use Rule 2

- `lhs = Parent(x, z) ∧ Ancestor(z, y)`, `rhs = Ancestor(x, y)`.
- Unify `Ancestor(John, Sam)` with `Ancestor(x, y)`:
  - Substitution: `θ = {x=John, y=Sam}`.
- Sub-goals:
  - `goals = [Parent(John, z), Ancestor(z, Sam)]`.

Step 5: FOL-BC-AND

- Goals: `[Parent(John, z), Ancestor(z, Sam)]`.

1. **First sub-goal (`Parent(John, z)`):**
    - Calls: `FOL-BC-OR(KB, Parent(John, z), θ)`.
    - Matches: `Parent(John, Mary)`.
    - Substitution: `{z=Mary}`.
    - Update θ′: `{x=John, y=Sam, z=Mary}`.
2. **Second sub-goal (`Ancestor(z, Sam)`):**
    - Calls: `FOL-BC-OR(KB, Ancestor(Mary, Sam), θ’)`.
    - Unify with Rule 1: `Parent(x, y) ⇒ Ancestor(x, y)`.
    - Sub-goal: `Parent(Mary, Sam)`.

Step 6: FOL-BC-AND

- Goal: `[Parent(Mary, Sam)]`.
- Matches fact: `Parent(Mary, Sam)`.
- Substitution: `{x=Mary, y=Sam}`.
- Satisfies all sub-goals.

Final Result

- Combine all substitutions:
  - `{x=John, y=Sam, z=Mary}`.
- The query `Ancestor(John, Sam)` is **true**.

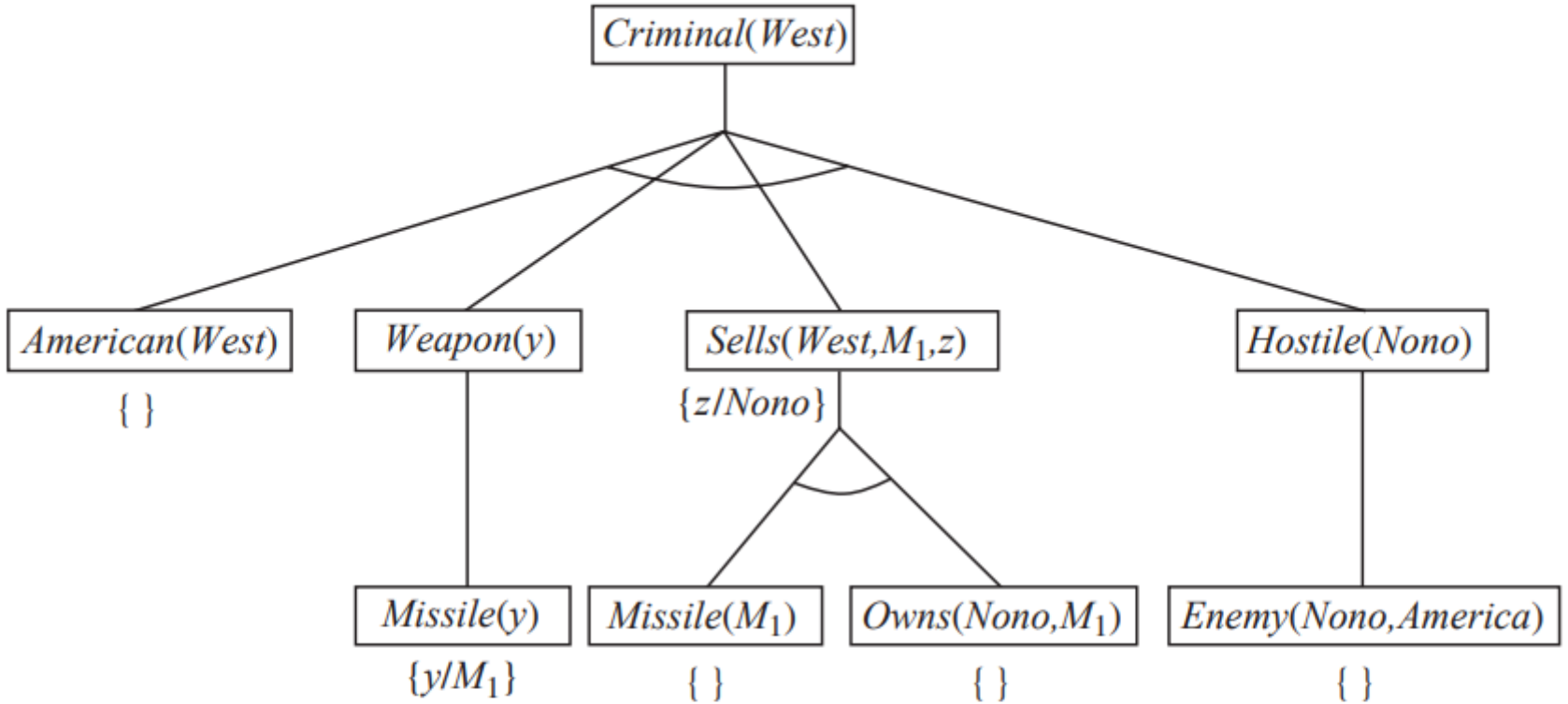


Figure 9.7 Proof tree constructed by backward chaining to prove that West is a criminal. The tree should be read depth first, left to right. To prove $Criminal(West)$, we have to prove the four conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. 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 $Nono$.

De Morgan's Laws in First-Order Logic (FOL)

De Morgan's Laws in **First-Order Logic (FOL)** are rules that describe how negation interacts with conjunction (∧) and disjunction (∨) in logical expressions. They also extend to quantifiers (∀and ∃) in FOL.

1. For Logical Connectives:

- $\neg(P \land Q) \equiv (\neg P \lor \neg Q)$

  (The negation of a conjunction is equivalent to the disjunction of the negations.)

- $\neg(P \lor Q) \equiv (\neg P \land \neg Q)$

  (The negation of a disjunction is equivalent to the conjunction of the negations.)

2. **For Quantifiers:**

De Morgan's Laws apply to quantifiers ($\forall$ and $\exists$) as well:

- $\neg(\forall x\, P(x)) \equiv \exists x\, (\neg P(x))$
  (The negation of "for all x, $P(x)$" is equivalent to "there exists an x such that $P(x)$ is not true.")
- $\neg(\exists x\, P(x)) \equiv \forall x\, (\neg P(x))$
  (The negation of "there exists an x such that $P(x)$" is equivalent to "for all x, $P(x)$ is not true.")

Example 1 (Connectives):

If $P(x) = x > 5$ and $Q(x) = x < 10$, then:

- $\neg(P(x) \wedge Q(x)) \equiv (\neg P(x) \vee \neg Q(x))$
- $\neg(x > 5 \wedge x < 10) \equiv (x \leq 5 \vee x \geq 10)$

Example 2 (Quantifiers):

If $P(x) = x > 5$, then:

- $\neg(\forall x\, P(x)) \equiv \exists x\, (\neg P(x))$

  "Not all x are greater than 5" is equivalent to "there exists some x that is not greater than 5."

- $\neg(\exists x\, P(x)) \equiv \forall x\, (\neg P(x))$

  "There does not exist an x greater than 5" is equivalent to "all x are not greater than 5."

Distributive law in First Order Logic

1. **Distributive Law of $\vee$ over $\wedge$:**

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

(Disjunction distributes over conjunction.)

2. **Distributive Law of $\wedge$ over $\vee$:**

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

(Conjunction distributes over disjunction.)

3. **Distributive Law with $\forall$:**

- $\forall x\,(P(x) \land Q(x)) \equiv (\forall x\,P(x)) \land (\forall x\,Q(x))$
  (Universal quantifier distributes over conjunction.)
- $\forall x\,(P(x) \lor Q(x)) \not\equiv (\forall x\,P(x)) \lor (\forall x\,Q(x))$
  *(Universal quantifier does **not** distribute over disjunction in general.)*

4. **Distributive Law with $\exists$:**

- $\exists x\,(P(x) \lor Q(x)) \equiv (\exists x\,P(x)) \lor (\exists x\,Q(x))$
  (Existential quantifier distributes over disjunction.)
- $\exists x\,(P(x) \land Q(x)) \not\equiv (\exists x\,P(x)) \land (\exists x\,Q(x))$
  *(Existential quantifier does **not** distribute over conjunction in general.)*

What is Tautology ?

- A **tautology** is a statement or logical formula that is always **true**, regardless of the truth values of its individual components.
- **Example: PV¬P**
  This means "P or not P." No matter whether P is true or false, one of them will always be true. **Hence, it's a tautology**

P	$\neg P$	$P \vee \neg P$
True	False	True
False	True	True

The result is always **true**, so $P \vee \neg P$ is a tautology.

Common Tautologies in Logic:

1. $P \vee \neg P$ ("Law of excluded middle")
2. $P \implies P$ ("Self-implication")
3. $(P \wedge Q) \implies P$ ("Conjunction implies one operand")

- In essence, tautologies are statements that cannot be false and are always true in classical logic.

Conjunctive Normal Form

- **A formula is in CNF if it is a conjunction (AND) of clauses, where each clause is a disjunction (OR) of literals.**

CNF Examples

1. $(A \vee B) \wedge (\neg A \vee C)$

   This expression has two clauses: $A \vee B$ and $\neg A \vee C$.

2. $(P \vee Q \vee R) \wedge (\neg P \vee \neg Q)$

   This expression also has two clauses: $P \vee Q \vee R$ and $\neg P \vee \neg Q$.

3. $(A \vee B \vee \neg C) \wedge (\neg A \vee \neg B \vee D) \wedge (C \vee D)$

   This expression has three clauses: $A \vee B \vee \neg C$, $\neg A \vee \neg B \vee D$, and $C \vee D$.

4. $(P \vee Q)$

   This is already in CNF, with one clause: $P \vee Q$.

5. $(A \wedge B) \vee (\neg C \wedge D)$

   This expression is not in CNF because it's a disjunction of conjunctions. To convert it to CNF, you'd need to apply distribution, obtaining $(A \vee \neg C) \wedge (A \vee D) \wedge (B \vee \neg C) \wedge (B \vee D)$.

Steps to Convert a Formula to CNF:

1. **Eliminate Biconditionals (⇔):** Replace each biconditional (↔) with an equivalent expression in terms of conjunction (∧), disjunction (∨), and negation (¬).
    - Example: Replace $A \Leftrightarrow B$ with $(A \Rightarrow B) \wedge (B \Rightarrow A)$

2. **Eliminate Implications** (⇒): Replace each implication (⇒) with an equivalent expression using conjunction and negation.
    - Example: Replace $A \Rightarrow B$ with $\neg A \vee B$

3. **Move Negations Inward** (¬): Apply De Morgan's laws and distribute negations inward to literals.
    - $\neg(\neg A) \equiv A$
    - $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$
    - $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$

4. **Distribute Disjunctions Over Conjunctions**: Apply the distributive law to ensure that disjunctions are only over literals or conjunctions of literals.
   Suppose we have the following formula: **$H=(P \wedge Q) \vee (R \wedge S \wedge T)$** .
   Now, let's distribute disjunctions over conjunctions: **$H=(P \vee (R \wedge S \wedge T)) \wedge (Q \vee (R \wedge S \wedge T))$**

Example: CNF Sentence

- **∀ x American(x) ∧ Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal(x)** becomes, in CNF,
- **¬American(x) ∨ ¬Weapon(y) ∨ ¬Sells(x, y, z) ∨ ¬Hostile(z) ∨ Criminal(x)** .

Example

We illustrate the procedure by translating the sentence

- “Everyone who loves all animals is loved by someone,” or
- **∀ x [∀ y Animal(y) ⇒ Loves(x, y)] ⇒ [∃ y Loves(y, x)]** .

Steps

- **Eliminate implications**: ∀ x [¬∀ y ¬Animal(y) ∨ Loves(x, y)] ∨ [∃ y Loves(y, x)] .
- **Move ¬ inwards**: In addition to the usual rules for negated connectives, we need rules for negated quantifiers. Thus, we have
  - ¬∀ x p becomes ∃ x ¬p
  - ¬∃ x p becomes ∀ x ¬p .
- **Our sentence goes through the following transformations**:
  - ∀ x [∃ y ¬(¬Animal(y) ∨ Loves(x, y))] ∨ [∃ y Loves(y, x)] .
  - ∀ x [∃ y ¬¬Animal(y) ∧ ¬Loves(x, y)] ∨ [∃ y Loves(y, x)] .
  - ∀ x [∃ y Animal(y) ∧ ¬Loves(x, y)] ∨ [∃ y Loves(y, x)] .
- **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. This avoids confusion later when we drop the quantifiers. Thus, we have
  - ∀ x [∃ y Animal(y) ∧ ¬Loves(x, y)] ∨ [∃ z Loves(z, x)] .

- **Skolemize**: Skolemization is the process of removing existential quantifiers by elimination. Translate ∃ x P(x) into P(A), where A is a new constant.
  - **Example :**
    - ∀ x [Animal(A) ∧ ¬Loves(x, A)] ∨ Loves(B, x) ,
    - ∀ x [Animal(F(x)) ∧ ¬Loves(x, F(x))] ∨ Loves(G(z), x) . Here F and G are Skolem functions.
- **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)] .

5.1.2 Resolution

- Resolution is a **fundamental inference rule** used in automated theorem proving and logic programming.
- It is based on the principle of **proof by contradiction**.
- Resolution **combines logical sentences** in the form of clauses to derive new sentences.

Resolution Rule

- The resolution rule states that if there are **two clauses** that contain complementary literals (**one positive, one negative**) then these literals can be **resolved, leading to a new clause** that is inferred from the **original clauses**.

Example 1:

Consider two logical statements:

1. PVQ
2. ¬PVR

Applying resolution: Resolve the statements by eliminating P:

- PVQ
- ¬PVR
- Resolving P and ¬P: **QVR**

The resulting statement **QVR is a new clause** inferred from the original two. Resolution is a key component of **logical reasoning in FOL**, especially in tasks like automated theorem proving and knowledge representation.

Example 2

Clause 1: (PVQVR)

Clause 2:(¬PV¬QVS)

Apply Resolution

- **Resolving P and ¬P: (QVR)V(¬QVS)**
- **Resolving Q and ¬Q gives (RVS)**

This is the resolvent.

Example 3

Premises:

p
$(p \wedge q) \Rightarrow r$
$(s \vee t) \Rightarrow q$

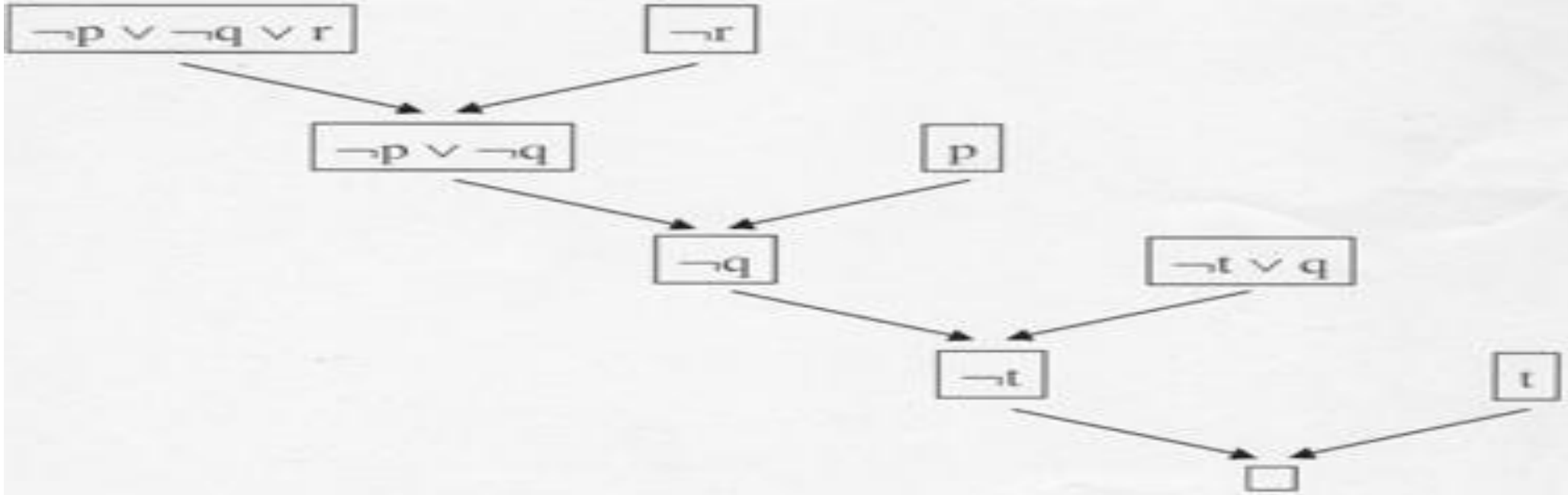
t

$\longrightarrow$

p
$\neg p \vee \neg q \vee r$
$\neg s \vee q$
$\neg t \vee q$
t

CNF

A resolution proof of r:



Proof By Resolution Process includes the following steps in general

1. **Initial Set of Clauses (Knowledge Base)**
2. **Negate the Conclusion**:
3. **Apply Resolution**
4. **Continue Resolving**
5. **Conclusion**
6. **Termination**

Example

- Let's consider a simplified example of a knowledge base for the **Wumpus World scenario** and demonstrate **proof by resolution** to establish the **unsatisfiability of a certain statement**.
- In Wumpus World**, an agent explores a grid containing** a Wumpus (a monster), **pits, and gold**. Apply the resolution to prove **P[1,2].**

Knowledge Base (KB)

1. **W[1,1] ∨ P[1,2]**
2. **¬W[1,1]∨¬P[1,2]**
3. **B[1,2]⇒P[1,2]**
4. **¬B[1,2]⇒¬P[1,2]**

Convert the Knowledge Base (KB) into CNF

1. $W[1,1] \vee P[1,2]$
2. $\neg W[1,1] \vee \neg P[1,2]$
3. $B[1,2] \Rightarrow P[1,2]$
4. $\neg B[1,2] \Rightarrow \neg P[1,2]$

1. $W[1,1] \vee P[1,2]$
2. $\neg W[1,1] \vee \neg P[1,2]$
3. $\neg B[1,2] \vee P[1,2]$
4. $B[1,2] \vee \neg P[1,2]$

Negated Conclusion:

Let's say we want to prove the negation of the statement:

¬PitIn[1,2]

Apply Resolution:

1. **W[1,1] ∨ P[1,2] , ¬P[1,2] resolves into W[1,1]**
2. **¬W[1,1]∨¬P[1,2], W[1,1] resolves into ¬P[1,2]**
3. **¬B[1,2] ∨ P[1,2] , ¬P[1,2] resolves into ¬B[1,2]**
4. **B[1,2] ∨¬P[1,2], ¬B[1,2] resolves into ¬P[1,2]**

Applying resolution, we end up with: **¬P[1,2] ,** Which is not empty and also there is not further any clauses to continue. This gives conclusion that our negation conclusion is **False** and **P[1,2]** is true for the given knowledge base.

The resolution inference 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.**

The resolution inference rule

- 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 unifier $\theta = \{u/G(x), v/x\}$, to produce the **resolvent** clause

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

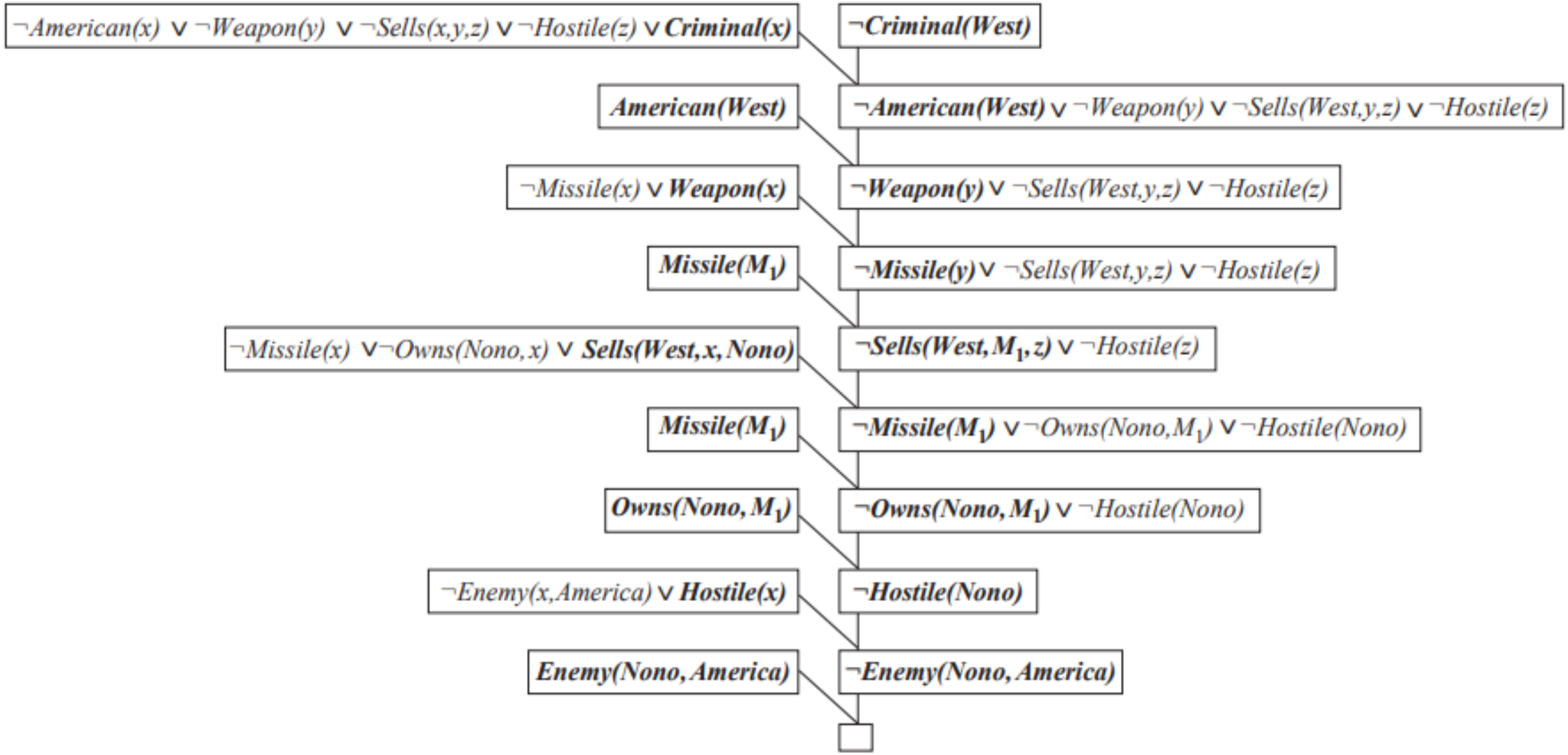


Figure 9.11 A resolution proof that West is a criminal. At each step, the literals that unify are in bold.

First, we express the original sentences, some background knowledge, 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. $\neg Kills(Curiosity,Tuna)$

Now we apply the conversion procedure to convert each sentence to 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. $\neg Kills(Curiosity,Tuna)$

Suppose Curiosity did not kill Tuna. We know that either Jack or Curiosity did; thus Jack must have. Now, Tuna is a cat and cats are animals, so Tuna is an animal. Because anyone who kills an animal is loved by no one, we know that no one loves Jack. On the other hand, Jack loves all animals, so someone loves him; so we have a contradiction. Therefore, Curiosity killed the cat.

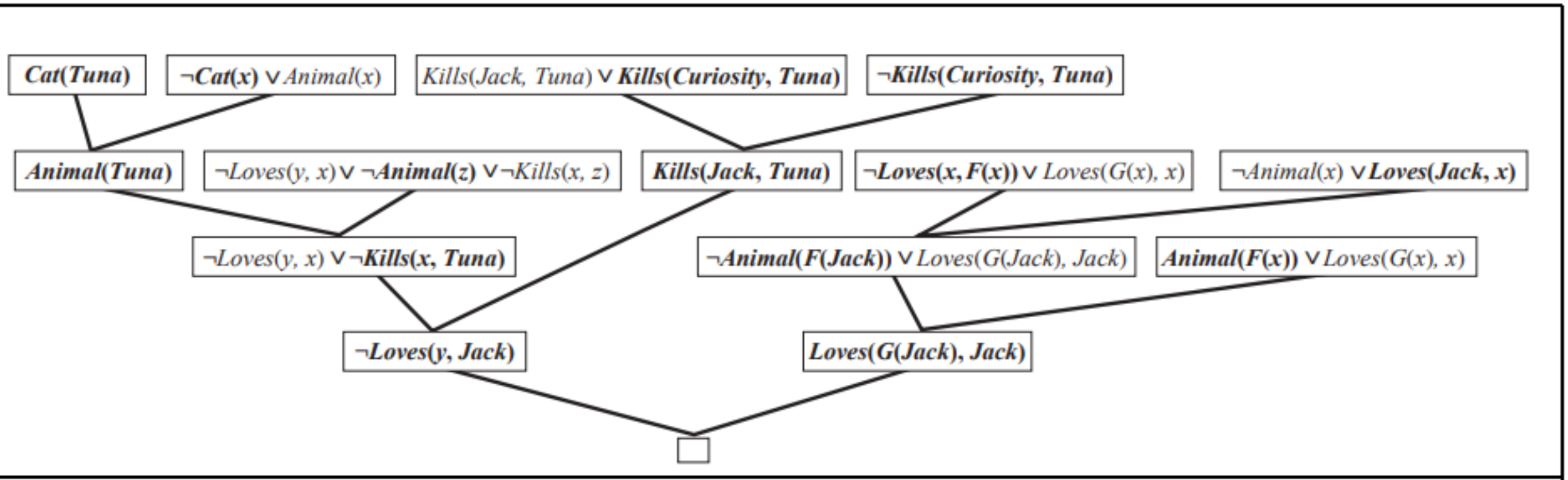


Figure 9.12 A resolution proof that Curiosity killed the cat. Notice the use of factoring in the derivation of the clause $Loves(G(Jack), Jack)$. Notice also in the upper right, the unification of $Loves(x, F(x))$ and $Loves(Jack, x)$ can only succeed after the variables have been standardized apart.

Summary

1. F**orward chaining starts** with known facts and moves forward to reach conclusions,
2. **Backward chaining** starts with the goal and moves backward to verify if the goal can be satisfied, and
3. **Resolution** is an inference rule used to derive new clauses by combining existing ones.

These ***techniques are essential for reasoning and inference in First-Order Logic systems***.