Module 4

First Order Logic and Inferences in FOL

Contents

Properties of PL

1. Propositional logic is a **declarative language** because its semantics is based on a truth relation between sentences and possible worlds.
2. It also has sufficient **expressive power to deal with partial information**, using disjunction and negation.
3. Propositional logic has a third property that is desirable in representation languages, namely, **compositionality.** In a compositional language, the meaning of a sentence is a function of the meaning of its parts.

   For example, the meaning of “$S1,4 \land S1,2$” is related to the meanings of “$S1,4$” and “$S1,2$.”

Drawbacks of Propositional Logic

- **Propositional logic** (PL) is declarative and assumes the world contains facts, so it guides us on how to represent information *in a logical form* and draw conclusions.
- We can only represent information as either **true or false** in propositional logic.
- Expressive power of Propositional logic is very limited and lacks to describe an environment with many objects
- If you want to represent complicated sentences or natural language statements, PL is not sufficient.
- **Examples:** PL is not enough to represent the sentences below, so we require powerful logic (such as FOL).
  1. I love mankind. It’s the people I can’t stand!
  2. I like to eat mangos.

What is First Order Logic (FOL)?

1. FOL is also called *predicate logic*. A much more expressive language than the propositional logic. It is a powerful language used to develop information about an **object and express the relationship** between objects.
2. FOL not only assumes that does the world contains facts (like PL does), but it also assumes the following:
   1. **Objects**: A, B, people, numbers, colors, wars, theories, squares, pit, etc.
   2. **Relations**: It is unary relation such as red, round, sister of, brother of, etc.
   3. **Function**: father of, best friend, third inning of, end of, etc.

First Order Logic Sentences

For each of the following English sentences, write a corresponding sentence in FOL.

1. The only good extraterrestrial is a drunk extraterrestrial.
   $\forall x. ET(x) \land Good(x) \to Drunk(x)$

2. The Barber of Seville shaves all men who do not shave themselves.
   $\forall x. \neg Shaves(x, x) \to Shaves(BarberOfSeville, x)$

3. There are at least two mountains in England.
   $\exists x, y. Mountain(x) \land Mountain(y) \land InEngland(x) \land InEngland(y) \land x \neq y$

4. There is exactly one coin in the box.
   $\exists x. Coin(x) \land InBox(x) \land \forall y. (Coin(y) \land InBox(y) \to x = y)$

5. There are exactly two coins in the box.
$\exists x, y. Coin(x) \wedge InBox(x) \wedge Coin(y) \wedge InBox(y) \wedge x \neq y \wedge \forall z.(Coin(z) \wedge InBox(z) \rightarrow (x = z \vee y = z))$

6. The largest coin in the box is a quarter.
$\exists x. Coin(x) \wedge InBox(x) \wedge Quarter(x) \wedge \forall y.(Coin(y) \wedge InBox(y) \wedge \neg Quarter(y) \rightarrow Smaller(y, x))$

7. No mountain is higher than itself.
$\forall x. Mountain(x) \rightarrow \neg Higher(x, x)$

8. All students get good grades if they study.
$\forall x. Student(x) \wedge Study(x) \rightarrow GetGoodGrade(x)$

Objects Relations and Functions

- **Objects:** people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries ...
- **Relations**: these can be unary relations or properties such as red, round, bogus, prime, multistoried ..., or more general n-ary relations such as brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, ...
- **Functions**: father of, best friend, third inning of, one more than, beginning of ..

Examples

- “One plus two equals three.”
  - **Objects**: one, two, three, one plus two;
  - **Relation**: equals;
  - **Function**: plus. (“One plus two” is a name for the object that is obtained by applying the function “plus” to the objects “one” and “two.” “Three” is another name for this object.)
- “Squares neighboring the wumpus are smelly.”
  - **Objects**: wumpus, squares;
  - **Property**: smelly; Relation: neighboring.
- “Evil King John ruled England in 1200.”
  - **Objects:** John, England, 1200;
  - **Relation**: ruled;
  - **Properties**: evil, king.

Types of Languages

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0, 1]$
Fuzzy logic	facts with degree of truth $\in [0, 1]$	known interval value

Basic Elements of FOL

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >, <,Sister, Father.......
Function	sqrt, LeftLegOf, Sqrt, LessThan, Sin(θ).......
Connectives	$\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\Leftrightarrow$
Equality	==
Quantifier	$\forall$, $\exists$

Syntax and Semantics of FOL

1. Models for first-order logic
2. Symbols and interpretations
3. Terms
4. Atomic sentences
5. Complex sentences
6. Quantifiers: Universal quantification (∀) /Existential quantification (∃)
7. Equality
8. An alternative semantics? : Data base Semantics

Models for FOL

- They have objects in them!
- The domain of a model is the set of objects or domain elements it contains.
- The domain is required to be nonempty—every possible world must contain at least one object
- The objects in the model may be related in various ways.
- Models in first-order logic require total functions, that is, there must be a value for every input tuple

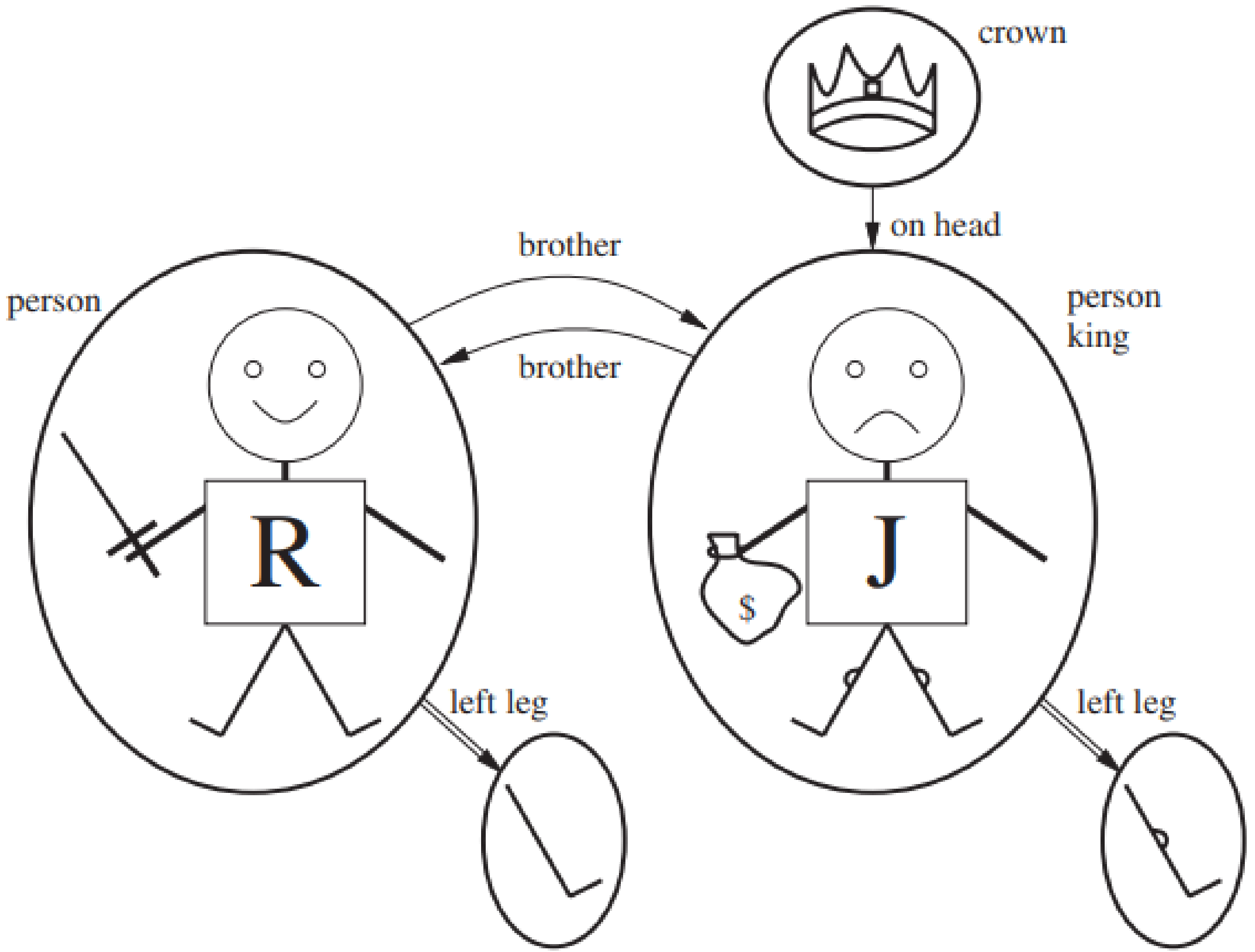

Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

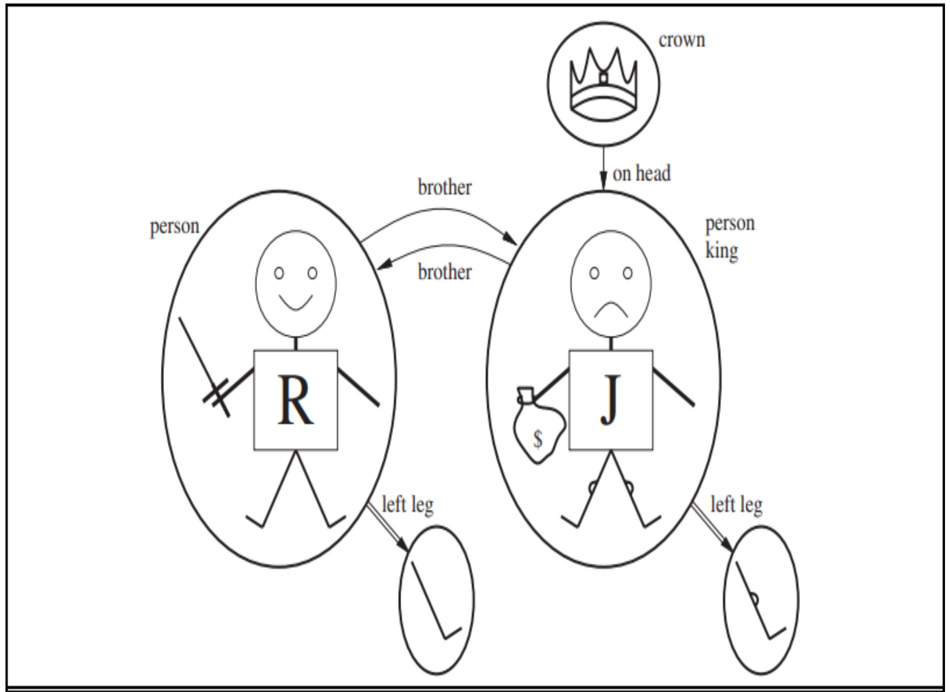

Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

Five objects:

1. Richard the Lionheart, King of England from 1189 to 1199;
2. His younger brother, the evil King John, who ruled from 1199 to 1215;
3. The left legs of Richard and John; and a c
4. Crown

Tuple : The brotherhood relation in this model is the set { <Richard the Lionheart, King John>, <King John, Richard the Lionheart> } .

Two binary relations : “brother” and “on head” relations are binary relations

Three unary relations/ properties : Person, King and Crown

One unary Function: Left Leg

Syntax of FOL

- The basic syntactic elements of first-order logic are the symbols that stand for objects, relations, and functions. The symbols, therefore, come in three kinds:
    1. **Constant symbols**, which stand for objects;
    2. **Predicate symbols**, which stand for relations; and
    3. **Function symbols**, which stand for functions.
- Convention : Symbols will begin with uppercase letters.
- Example
    - Constant symbols Richard and John;
    - Predicate symbols Brother , OnHead, Person, King, and Crown; and
    - the function symbol LeftLeg.
- Arity : Each predicate and function symbol comes with an arity that fixes the number of arguments

Syntax of FOL

- **Interpretation**: specifies exactly which objects, relations and functions are referred to by the constant, predicate, and function symbols.
- Examples :
  - **Richard** refers to Richard the Lionheart
  - **John** refers to the evil King John.
  - **Brother** refers to the brotherhood relation
  - **OnHead** refers to the “on head” relation that holds between the crown and King John;
  - **Person, King, and Crown** refer to the sets of objects that are persons, kings, and crowns
  - **LeftLeg** refers to the “left leg” function

The syntax of first-order logic with equality, specified in Backus–Naur form

$$
\begin{array}{rcl}
\mathit{Sentence} & \rightarrow & \mathit{AtomicSentence} \mid \mathit{ComplexSentence} \\
\mathit{AtomicSentence} & \rightarrow & \mathit{Predicate} \mid \mathit{Predicate}(\mathit{Term},\ldots) \mid \mathit{Term} = \mathit{Term} \\
\mathit{ComplexSentence} & \rightarrow & (\ \mathit{Sentence}\ ) \mid [\ \mathit{Sentence}\ ] \\
 & \mid & \neg\ \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \land \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \lor \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \Rightarrow \mathit{Sentence} \\
 & \mid & \mathit{Sentence} \Leftrightarrow \mathit{Sentence} \\
 & \mid & \mathit{Quantifier}\ \mathit{Variable},\ldots\ \mathit{Sentence} \\
 & & \\
\mathit{Term} & \rightarrow & \mathit{Function}(\mathit{Term},\ldots) \\
 & \mid & \mathit{Constant} \\
 & \mid & \mathit{Variable} \\
 & & \\
\mathit{Quantifier} & \rightarrow & \forall \mid \exists \\
\mathit{Constant} & \rightarrow & A \mid X_1 \mid \mathit{John} \mid \cdots \\
\mathit{Variable} & \rightarrow & a \mid x \mid s \mid \cdots \\
\mathit{Predicate} & \rightarrow & \mathit{True} \mid \mathit{False} \mid \mathit{After} \mid \mathit{Loves} \mid \mathit{Raining} \mid \cdots \\
\mathit{Function} & \rightarrow & \mathit{Mother} \mid \mathit{LeftLeg} \mid \cdots \\
\textsc{Operator Precedence} & : & \neg, =, \land, \lor, \Rightarrow, \Leftrightarrow
\end{array}
$$

In summary

- A model in first-order logic consists of a set of objects and an interpretation that maps constant symbols to objects, predicate symbols to relations on those objects, and function symbols to functions on those objects.
- Just as with propositional logic, entailment, validity, and so on are defined in terms of all possible models

Syntax and Semantics of FOL

1. Models for first-order logic
2. Symbols and interpretations
3. Terms
4. Atomic sentences
5. Complex sentences
6. Quantifiers: Universal quantification (∀) /Existential quantification (∃)
7. Equality
8. An alternative semantics? : Data base Semantics

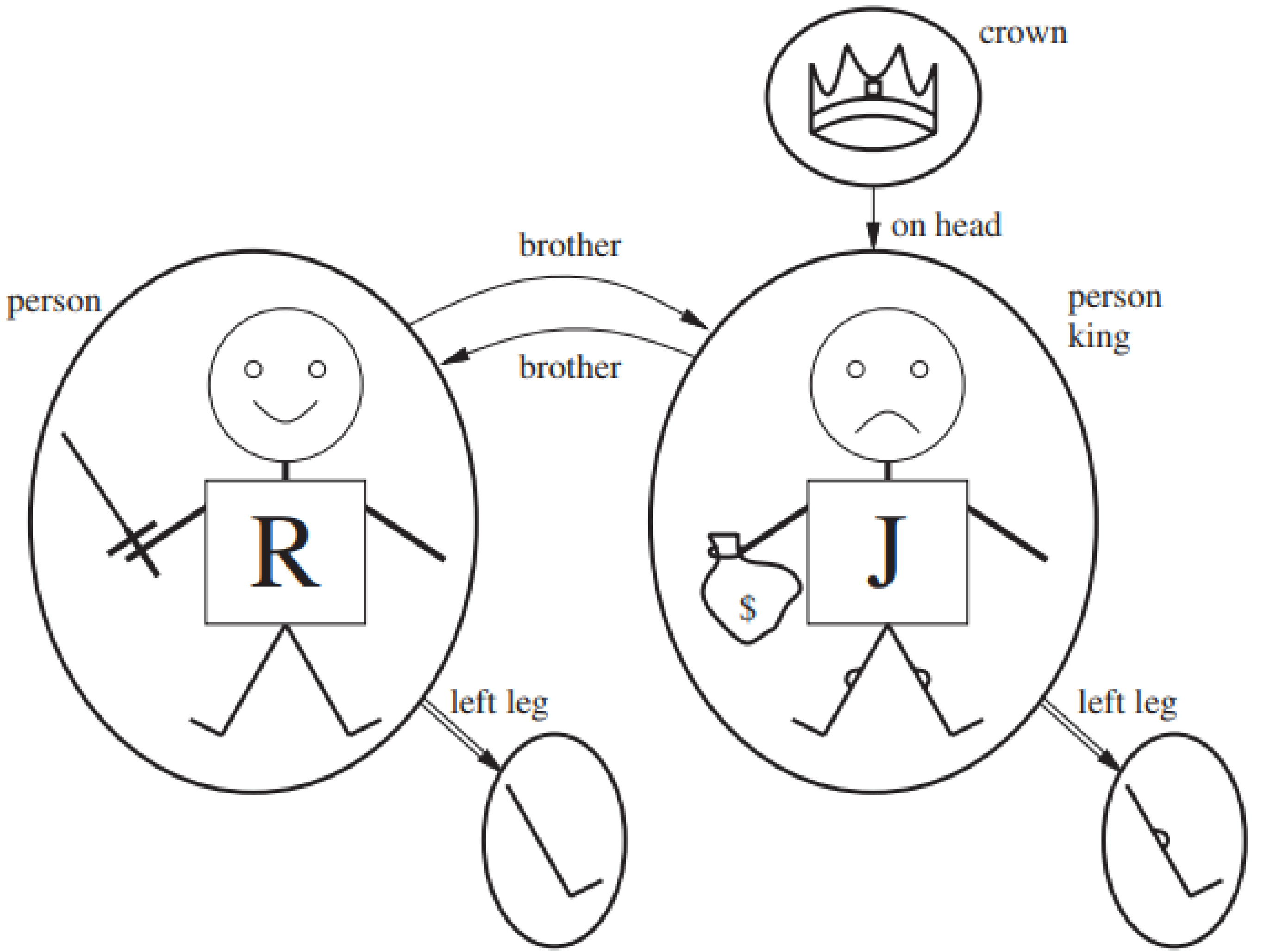

Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

3.Terms

- A term is a logical expression that refers to an object. Constant symbols are terms.
- It is not always convenient to have a distinct symbol to name every object.
  - **Example** : “**King John’s left leg**” rather than giving a name to his leg.
  - This is what function symbols are for: instead of using a constant symbol, we use **LeftLeg(John).**
- In the general case, a complex term is formed by a **function symbol** followed by a **parenthesized** list of terms as arguments to the function symbol.(It is not a subroutine or function call)

3.Terms

- **Formal Semantics** : **Consider a term f(t1,... ,tn).**
- The function symbol **f r**efers to some function in the model (call it **F**); the argument terms refer to objects in the domain (call them d1,... ,dn);
- **Example** : LeftLeg(John):
- The **LeftLeg** function symbol refers to the function and **John** refers to King John, then **LeftLeg(John)** refers to **King John's left leg.**

4.Atomic sentences

- An atomic sentence (**or atom for short**) is formed from a predicate symbol optionally followed by a parenthesized list of terms, such a
  - ***Brother (Richard, John) :*** This states, that Richard the Lionheart is the brother of King John.

- Atomic sentences can have **complex terms as arguments**:
  - **Married(Father (Richard), Mother (John))** : states that Richard the Lionheart's father is married to King John's mother

5.Complex Sentences

- We can use **logical connectives to construct more complex sentences**, with the same syntax and semantics as in propositional calculus
- **Example** : Here are four sentences that are true in the model
  1. **¬Brother (LeftLeg(Richard), John)**
  2. **Brother (Richard, John) ∧ Brother (John, Richard)**
  3. **King(Richard) ∨ King(John)**
  4. **¬King(Richard) ⇒ King(John)**

6. Quantifiers

- In first-order logic, quantifiers are symbols used to express the scope of variables in logical statements.
- **Quantifiers are essential** for expressing statements about collections of objects or individuals in a precise and concise manner within **first-order logic**. They allow for the formulation of statements that capture universal truths or existential claims about the elements of a domain.
- There are two main quantifiers:
  1. **the existential quantifier (∃)** and
  2. **the universal quantifier (∀).**

6. Quantifiers: Universal quantification (∀)

Universal Quantifier (∀): Denoted by the symbol "∀".

- It asserts that a **predicate or condition is true for all** instances of a variable in a given domain.
- For example, the statement **"∀x P(x)"** asserts that the predicate **P(x)** is true for all x in the domain.
- By convention, variables are lowercase letters.
- A variable is a **term all by itself**, and as such can also serve as the **argument of a function**—for example, **LeftLeg(x).**
- A term with no variables is called a **ground term.**

6. Universal Quantifiers: Examples

- $\forall x \; King(x) \Rightarrow Person(x)$.

6. Universal Quantifiers: Examples

- The universally quantified sentence **$\forall$ x King(x) $\Rightarrow$ Person(x)** is true in the original model if the sentence **King(x) $\Rightarrow$ Person(x)** is true under each of the five extended interpretations.
- That is, the universally quantified sentence is equivalent to asserting the following five sentences:

1. **Richard the Lionheart is a king $\Rightarrow$ Richard the Lionheart is a person.**
2. **King John is a king $\Rightarrow$ King John is a person.**
3. **Richard's left leg is a king $\Rightarrow$ Richard's left leg is a person.**
4. **John's left leg is a king $\Rightarrow$ John's left leg is a person.**
5. **The crown is a king $\Rightarrow$ the crown is a person**

6. Universal Quantifiers: Examples

- A **common mistake, made frequently** even by diligent readers who have read this paragraph several times, is to use conjunction instead of implication.
- The sentence **∀ x King(x) ∧ Person(x**) would be equivalent to asserting

1. Richard the Lionheart is a king ∧ Richard the Lionheart is a person,
2. King John is a king ∧ King John is a person,
3. Richard’s left leg is a king ∧ Richard’s left leg is a person,

6. Quantifiers: Existential quantification (∃)

- Denoted by the symbol **"∃".**
- It asserts that there exists at least one instance of a variable that satisfies a given predicate or condition.
- **For example,** the statement **"∃x P(x)"** asserts that there exists at least one x such that the predicate P(x) is true.

6. Quantifiers: Existential quantification (∃)

- Universal quantification makes statements about every object.
- Similarly, we can make a statement about **some object** in the universe **without naming it, by using an existential quantifier**.
- To say, for example, that King John has a crown on his head, we write **$\exists$ x Crown(x) $\wedge$ OnHead(x, John) .**
- $\exists$x is pronounced "There exists an x such that ..." or "For some x..."

6. Quantifiers: Existential quantification (∃)

- Intuitively, the sentence **∃ x P says that P is true** for at least one **object x.**
- More precisely, **∃ x P** is true in a given model if **P** is true in at least one extended interpretation that assigns **x to a domain element**.
- **That is, at least one of the following is true:**

1. Richard the Lionheart is a crown ∧ Richard the Lionheart is on John’s head;
2. King John is a crown ∧ King John is on John’s head;
3. Richard’s left leg is a crown ∧ Richard’s left leg is on John’s head;
4. John’s left leg is a crown ∧ John’s left leg is on John’s head;
5. The crown is a crown ∧ the crown is on John’s head

6. Quantifiers: Existential quantification (∃)

- Using ⇒ **with ∃** usually leads to a very weak statement, indeed.
- Consider the following sentence: **∃ x Crown(x) ⇒ OnHead(x, John)** .
- Applying the semantics, we see that the sentence says that at least one of the following assertions is true:

1. Richard the **Lionheart is a crown ⇒ Richard the Lionheart is on John's head**;
2. **King John is a crown ⇒ King John is on John's head**;
3. **Richard's left leg is a crown ⇒ Richard's left leg is on John's head**;

6. 1 : Nested Quantifiers

- Nested quantifiers in **First-Order Logic (FOL)** refer to situations where quantifiers are used within the scope of other quantifiers in a logical expression.
- This nesting allows for the expression of more complex relationships and properties involving multiple variables.
- **There are two types of quantifiers in FOL**: the universal quantifier (∀) and the existential quantifier (∃), and both can be used in nested configurations.

6. 1 : Nested Quantifiers : Examples and Interpretations

1. **$\forall x\ \exists y\ P(x,y)$:**
   - This statement means "**for every x, there exists a y such that the property P(x,y) holds**."
   - It asserts a universal condition on x and, for each x, an existential condition on y.
2. **$\exists x\ \forall y\ P(x,y)$:**
   - This statement means "**there exists an x such that for every y, the property P(x,y) holds**."
   - It asserts the existence of a particular x for which a universal statement about y is true.

6. 1 : Nested Quantifiers : Examples and Interpretations

- **Importance of Order :** The order of **nested quantifiers is crucial** because it can change the meaning of a statement.
- For instance, the two examples given above have significantly different meanings due to the order of quantification.
- In general, **changing the order of quantifiers** in a statement with nested quantifiers will result in a statement that expresses a different property or relationship.

6. 1 : Nested Quantifiers : Examples and Interpretations

- Nested quantifiers are widely used in mathematics, computer science, and philosophy to express complex statements about sets, functions, algorithms, and theoretical constructs. T
- hey are essential for defining concepts like "**for every natural number, there exists a prime number greater than it**"
- $(\forall x \in \mathbb{N}, \exists y\ (y > x \wedge \text{Prime}(y)))$
- or expressing constraints and properties in formal specifications and proofs.

6. 1 : Nested Quantifiers : Examples and Interpretations

- Siblinghood is a symmetric relationship :

  $\forall$ x,y Sibling(x,y) $\Leftrightarrow$ Sibling(y,x)
- “Everybody loves somebody” : **$\forall$ x $\exists$ y Loves(x,y)**
- “There is someone who is loved by everyone,”: **$\exists$ y $\forall$ x Loves(x,y)**
- The order of quantification is therefore very important. It becomes clearer if we insert parentheses.
  - $\forall$ x ($\exists$ y Loves(x,y)) **says that everyone has a particular property, namely, the property that they love someone**

6. 1 : Nested Quantifiers : Connections between ∀ and ∃

- The two quantifiers are actually intimately connected with each other, through **negation**.
- Asserting that everyone dislikes **Parsnips** is the same as asserting there does not exist someone who likes them, and vice versa:
- **$\forall$ x ¬Likes(x,Parsnips) is equivalent to ¬$\exists$ x Likes(x,Parsnips)**

6. 1 : Nested Quantifiers : Connections between ∀ and ∃

- "**Everyone likes ice cream**" means that there is no one who does not like ice cream:
- **$\forall$ x Likes(x,IceCream)** is equivalent to

$\neg\exists$ x $\neg$Likes(x,IceCream) .

6. 2 : De Morgans Rules for quantified and unquantified sentences

- $\forall\ x\ \neg P \equiv \neg\exists\ x\ P$
- $\neg\forall\ x\ P \equiv \exists\ x\ \neg P$
- $\forall\ x\ P \equiv \neg\exists\ x\ \neg P$
- $\exists\ x\ P \equiv \neg\forall\ x\ \neg P$

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

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

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

$P \vee Q \equiv \neg(\neg P \wedge \neg Q)$.

7.Equality

- In First-Order Logic (FOL), equality is a fundamental concept that allows the expression of the notion that two terms denote the same object.
- **Syntax**: In the syntax of FOL, an equality statement typically looks like a = b where *a* and *b* are terms in the logic. Terms can be variables, constants, or any expression that refers to objects in the domain of discourse.
- **Semantics**: The semantics of equality states that a = b is true if and only if *a* and *b* refer to the same object in the domain of discourse.

7.Equality: Examples

- **Father (John)= Henry** says that the object referred to by Father (John) and the object referred to by Henry are the same.
- To say that Richard has at least two brothers, we would write
- **$\exists$ x,y Brother (x, Richard) $\land$ Brother (y, Richard) $\land$ $\neg$(x = y) .**

8.Any Alternative Semantics : Database Semantics

- One proposal that is very popular in database systems works as follows.
- **First, we** insist that every **constant symbol** refer to a distinct object—the so-called unique-names assumption.
- **Second,** we assume that atomic sentences not known to be true are **in fact false—the** closed-world assumption.
- Finally, we invoke **domain closure**, meaning that each model contains no more domain elements than those named by the **constant symbols**.

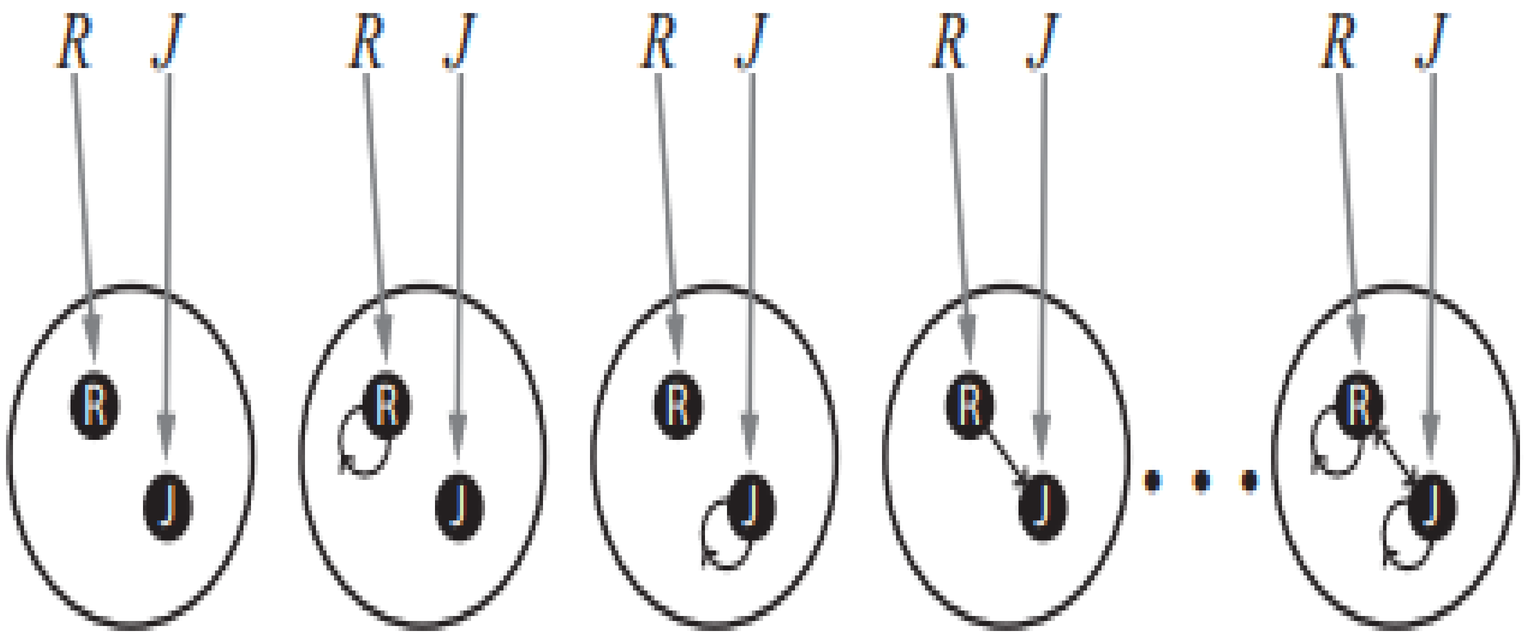

Figure 8.5 Some members of the set of all models for a language with two constant symbols, R and J, and one binary relation symbol, under database semantics. The interpretation of the constant symbols is fixed, and there is a distinct object for each constant symbol.

Using First Order Logic

- **In this section, we discuss systematic representations of some simple domains.**
- **In knowledge representation, a domain is just some part of the world about which we wish to express some knowledge.**

Assertions and **queries** in first-order logic

- Sentences are added to a knowledge base using **TELL**, exactly as in propositional logic. Such sentences are called **assertions**
- We can ask questions of the knowledge base using **ASK**. Questions asked with ASK are called **queries or goals.**

Examples

TELL(KB, King(John)) .
TELL(KB, Person(Richard)) .
TELL(KB, ∀ x King(x) ⇒ Person(x)) .

ASK(KB, King(John))
ASK(KB, Person(John))
ASK(KB, ∃ x Person(x)) .

ASKVARS : Substitution or binding List

- **ASKVARS(KB, Person(x))** yields a stream of answers. In this case there will be two answers: {x/John} and {x/Richard}. Such an answer is called a **substitution or binding list. It** will bind the variables to specific values.

- **Note: i**f KB has been told **King(John)** ∨ **King(Richard),** then there is **no binding to x for the query ∃ x King(x)**, even though the query is true.

Example: The domain of family relationships, or kinship domain

- **This domain includes facts such as**
  - **"Elizabeth is the mother of Charles"** and
  - **"Charles is the father of William"** and rules such as
  - **"One's grandmother is the mother of one's parent**."
- Clearly, the objects in our domain are people. We have two unary predicates, **Male and Female**.
- **Kinship relations**—parenthood, brotherhood, marriage, and so on—are represented by binary predicates: ***Parent, Sibling, Brother , Sister , Child, Daughter , Son, Spouse, Wife, Husband, Grandparent, Grandchild , Cousin, Aunt, and Uncle.***

- **One's mother is one's female parent:** $\forall m,c\ Mother(c) = m \Leftrightarrow Female(m) \land Parent(m,c)$.
- **One's husband is one's male spouse:** $\forall w,h\ Husband(h,w) \Leftrightarrow Male(h) \land Spouse(h,w)$.
- **Male and female are disjoint categories:** $\forall x\ Male(x) \Leftrightarrow \neg Female(x)$.
- **Parent and child are inverse relations:** $\forall p,c\ Parent(p,c) \Leftrightarrow Child(c,p)$
- **A grandparent is a parent of one's parent:**

$$\forall g,c\ Grandparent(g,c) \Leftrightarrow \exists p\ Parent(g,p) \land Parent(p,c) .$$

- **A sibling is another child of one's parents:**

$$\forall x,y\ Sibling(x,y) \Leftrightarrow x\ 6= y \land \exists p\ Parent(p,x) \land Parent(p,y) .$$

Axioms : Each of these sentences can be viewed as an axiom of the kinship domain. They provide the basic factual information from which useful conclusions can be derived. Our kinship axioms are also definitions; they have the form **$\forall$ x,y P(x,y) $\Leftrightarrow$...** The axioms define the **Mother function and the Husband, Male, Parent, Grandparent, and Sibling predicates** in terms of other predicates.

Some are theorems—that is, they are entailed by the axioms. For example, consider the **assertion that siblinghood is symmetric: $\forall$ x,y Sibling(x,y) $\Leftrightarrow$ Sibling(y,x)** .

Numbers

NatNUM

We describe here the theory of natural numbers or non-negative integers.Natural numbers are defined recursively

- 0 is a natural number : **NatNum(0)** .
- For every object n, if n is a natural number, then S(n) is a natural number :

  $\forall$ n NatNum(n) $\Rightarrow$ NatNum(S(n))

So the natural numbers are 0, S(0), S(S(0)), and so on.

Axioms

$\forall$ n, 0 $\neq$ S(n) .

$\forall$ m,n m $\neq$ n $\Rightarrow$ S(m) $\neq$ S(n) .

Note : We can also write S(n) as n + 1

Numbers

Definition : Addition is defined in terms of the successor function:

- $\forall m\ NatNum(m) \Rightarrow +(0,m) = m$.
- $\forall m,n\ NatNum(m) \land NatNum(n) \Rightarrow +(S(m),n) = S(+(m,n))$

or

- $\forall m,n\ NatNum(m) \land NatNum(n) \Rightarrow (m + 1) + n = (m + n) + 1$

Note : The use of infix notation(like m+1,m+n,etc) is an example of **syntactic sugar,** that is, an extension to or abbreviation of the standard syntax that does not change the semantics.

Sets

- The domain of sets is also fundamental to mathematics as well as to commonsense reasoning. We will use the normal vocabulary of set theory as syntactic sugar.
- The **empty set** is a constant written as **{ }**.
- There is one **unary predicate**, **Set**, which is true of sets.
- The **binary predicates** are **x∈ s** (x is a member of set s) and **s1 ⊆ s2** (set **s1** is a subset, not necessarily proper, of set **s2**).
- The binary functions are
    - **s1 ∩ s2** (the intersection of two sets),
    - **s1 ∪ s2** (the union of two sets), and
    - **{x|s}** (the set resulting from adjoining element x to set s).

One possible set of axioms of Sets is as follows:

1. The only sets are the empty set and those made by adjoining something to a set:
   - **$\forall$ s Set(s) $\Leftrightarrow$ (s = { }) $\vee$ ($\exists$ x,s2 Set(s2) $\wedge$ s = {x|s2})**
2. The empty set has no elements adjoined into it. In other words, there is no way to decompose { } into a smaller set and an element:
   - **$\neg\exists$ x,s {x|s} = { }** .
3. Adjoining an element already in the set has no effect:
   - **$\forall$ x,s x$\in$ s $\Leftrightarrow$ s = {x|s} .**
4. The only members of a set are the elements that were adjoined into it. We express this recursively, saying that x is a member of s if and only if s is equal to some set s2 adjoined with some element y, where either y is the same as x or x is a member of s2:
   - **$\forall$ x,s x$\in$ s $\Leftrightarrow$ $\exists$ y,s2 (s = {y|s2} $\wedge$ (x = y $\vee$ x$\in$ s2)) .**

One possible set of axioms of Sets is as follows:

5. A set is a subset of another set if and only if all of the first set's members are members of the second set:
   - **$\forall$ s1,s2 s1 $\subseteq$ s2 $\Leftrightarrow$ ($\forall$ x x$\in$ s1 $\Rightarrow$ x$\in$ s2) .**

6. Two sets are equal if and only if each is a subset of the other:
- **$\forall$ s1,s2 (s1 = s2) $\Leftrightarrow$ (s1 $\subseteq$ s2 $\wedge$ s2 $\subseteq$ s1) .**

7. An object is in the **intersection** of two sets if and only if it is a member of both sets:
- **$\forall$ x,s1,s2 x$\in$ (s1 $\cap$ s2) $\Leftrightarrow$ (x$\in$ s1 $\wedge$ x$\in$s2) .**

8. An object is in the union of two sets if and only if it is a member of either set:
- **$\forall$ x,s1,s2 x$\in$ (s1 $\cup$ s2) $\Leftrightarrow$ (x$\in$ s1 $\vee$ x$\in$s2) .**

Lists

- Lists are similar to sets. The differences are that lists are *ordered and the same element can appear more than once in a list*.
- **Nil** is the constant list with no elements;
- **Cons, Append, First, and Rest** are functions; and
- **Find** is the predicate that does for lists what Member does for sets.
- **List?** is a predicate that is true only of lists.
- The empty list is [].
- The term **Cons(x,y)**, where y is a nonempty list, is written **[x|y]**.
- The term **Cons(x, Nil)** (i.e., the list containing the element x) is written as **[x]**.
- A list of several elements, such as **[A,B,C]**, corresponds to the nested term **Cons(A, Cons(B, Cons(C, Nil))).**

Wumpus World

- The wumpus agent receives a percept vector with five elements. A typical percept sentence would be
  - ***Percept([Stench, Breeze, Glitter , None, None], 5)*** .
- Here, Percept is a **binary predicate**, and **Stench** and so on are constants placed in a list
- The actions in the wumpus world can be represented by logical terms:
  - **Turn(Right), Turn(Left), Forward, Shoot, Grab, Climb .**
- To determine which is best, the agent program executes the query
  - **ASKVARS(∃ a BestAction(a, 5)) ,**
  - which returns a binding list such as **{a/Grab}.** The agent program can then return Grab as the action to take
- The raw percept data implies certain facts about the current state. For example:
  - **∀ t,s,g,m,c Percept([s, Breeze,g,m,c],t) ⇒ Breeze(t) ,**
  - **∀ t,s,b,m,c Percept([s,b, Glitter ,m,c],t) ⇒ Glitter (t) ,**

and so on. These rules exhibit a trivial form of the reasoning process called perception,

Wumpus World

- Simple “reflex” behavior can also be implemented by quantified implication sentences.
  - For example, we have **$\forall$ t Glitter (t) $\Rightarrow$ BestAction(Grab,t)** .

- Adjacency of any two squares can be defined as
  $\forall$ x,y,a,b Adjacent([x,y], [a,b]) $\Leftrightarrow$
  $(x = a \land (y = b - 1 \lor y = b + 1)) \lor (y = b \land (x = a - 1 \lor x = a + 1))$

- We can then say that objects can only be at one location at a time:
  $\forall$ x,s1,s2,t At(x,s1,t) $\land$ At(x,s2,t) $\Rightarrow$ s1 = s2

- Given its current location, the agent can infer properties of the square from properties of its current percept. For example, if the agent is at a square and perceives a breeze, then that square is breezy:
  $\forall$ s,t At(Agent,s,t) $\land$ Breeze(t) $\Rightarrow$ Breezy(s) .

Wumpus World

- The agent can deduce where the pits are (and where the wumpus is)
  $\forall s\ Breezy(s) \Leftrightarrow \exists r\ Adjacent(r,s) \land Pit(r)$.
- **Axiom :**
  $\forall t\ HaveArrow(t + 1) \Leftrightarrow (HaveArrow(t) \land \neg Action(Shoot,t))$.

Module 4: Chapter 2

Inference in First Order Logic:

a. Propositional Versus First Order Inference,
b. Unification,
c. Forward Chaining,
d. Backward Chaining
e. Resolution

Propositional Versus First Order Inference,

1. Inference rules for quantifiers
2. Reduction to propositional inference

1. Inference rules for quantifiers

- Consider axiom stating that all greedy kings are evil:
  - $\forall$ x King(x) $\land$ Greedy(x) $\Rightarrow$ Evil(x) .
- Then it seems quite permissible to infer any of the following sentences:
  - King(John) $\land$ Greedy(John) $\Rightarrow$ Evil(John)
  - King(Richard) $\land$ Greedy(Richard) $\Rightarrow$ Evil(Richard)
  - King(Father (John)) $\land$ Greedy(Father (John)) $\Rightarrow$ Evil(Father (John)) .
  - ---------------------

a) The rule of Universal Instantiation (UI for short)

- The rule of **Universal Instantiation (UI for short)** says that we can infer any sentence obtained by substituting a ground term (a term without variables) for the variable.
- To write out the inference rule formally, we use **SUBST(θ, α)** denote the result of applying the substitution θ to the sentence α. Then the rule is written

$$\frac{\forall v \;\; \alpha}{\textsc{Subst}(\{v/g\}, \alpha)}$$

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

b) The rule for Existential Instantiation

- 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,

$$\frac{\exists v \;\; \alpha}{\textsc{Subst}(\{v/k\}, \alpha)} \, .$$

For example, from the sentence

$$\exists x \;\; Crown(x) \land OnHead(x, John)$$

we can infer the sentence

$$Crown(C_1) \land OnHead(C_1, John)$$

as long as C_1 does not appear elsewhere in the knowledge base.

2. Reduction to propositional inference (Propositionalization)

- The existentially quantified sentence can be replaced by one instantiation, and universally quantified sentence can be replaced by the set of all possible instantiations.
- For example, suppose our knowledge base contains just the sentences

FOL Inference

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

Propositional logic Inference

$$King(John) \land Greedy(John) \;\Rightarrow\; Evil(John)$$
$$King(Richard) \land Greedy(Richard) \;\Rightarrow\; Evil(Richard) \,,$$

Unification

- Generalized Modus Ponens is a lifted version of Modus Ponens—it raises Modus Ponens from ground (variable-free) propositional logic to first-order logic.
- Generalized Modus Ponens: For atomic sentences pi , pi ′ , and q, where there is a substitution θ

such that $\text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i)$, for all i,

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

There are $n+1$ premises to this rule: the n atomic sentences p_i' and the one implication. 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 $Evil(x)$
$\text{SUBST}(\theta, q)$ is $Evil(John)$.	

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:

$$\text{UNIFY}(p, q) = \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q) \ .$$

Example : Suppose we have a query **AskVars(Knows(John, x)):** whom does John know? Answers to this query can be found by finding all sentences in the knowledge base that unify with Knows(John, x). Here are the results of unification with four different sentences that might be in the knowledge base:

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 .

Unification

- In first-order logic**, unification is** a process used to find a common instantiation for two predicates or terms such that they become identical.
  - It's a fundamental operation in logic programming and automated reasoning, allowing for the comparison and integration of different logical expressions.
  - Unification is essential for tasks such as theorem proving, pattern matching, and resolution in logic-based systems.
- **A substitution**, on the other hand, is a mapping of variables to terms.
  - It's essentially a set of assignments that replaces variables in logical expressions with specific terms, thereby creating a new expression that may be simpler or more specific than the original one.
  - Substitutions are used to represent the results of unification and are crucial for maintaining consistency and correctness in logical inference.

```
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
```

```
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
```

Figure 9.1 The unification algorithm. The algorithm works by comparing the structures of the inputs, element by element. The substitution θ that is the argument to UNIFY is built up along the way and is used to make sure that later comparisons are consistent with bindings that were established earlier. In a compound expression such as $F(A, B)$, the OP field picks out the function symbol F and the ARGS field picks out the argument list (A, B).

Unification is the process of finding a substitution that makes two logical expressions identical. The algorithm takes two expressions, x and y, and attempts to find a substitution (θ) that makes them identical.
Here's a breakdown of how the algorithm works:

Base case: If the substitution **θ i**s already marked as a failure, then it returns failure immediately.
Identity check: If **x** and **y** are identical, it means no further unification is needed, and the current substitution **θ** can be returned.
Variable check: If **x** is a variable, it calls the **UNIFY-VAR** function with **x** as the variable and **y** as the expression. If **y** is a variable, it calls **UNIFY-VAR** with y as the variable and x as the expression.
Compound expression check: If both **x** and **y** are compound expressions, it recursively calls **UNIFY** on their arguments and operators.
List check: If both **x** and **y** are lists, it recursively calls UNIFY on their first elements and their remaining elements.
Failure case: If none of the above conditions are met, it returns failure, indicating that x and y cannot be unified.
The UNIFY-VAR function is used when one of the expressions **(x or y)** is a variable. It attempts to create a substitution based on the variable and the expression it's being unified with.

The UNIFY-VAR function is used when one of the expressions (x or y) is a variable. It attempts to create a substitution based on the variable and the expression it's being unified with.

Substitution check: If the substitution already contains a mapping for the variable, it recursively calls UNIFY with the mapped value and the expression x.

Reverse substitution check: If the expression is already in the substitution, it recursively calls UNIFY with the variable and the mapped value.

Occur check: Checks for a possible occurrence of the variable in the expression, preventing infinite loops, and returns failure if such an occurrence is detected.

Substitution addition: If none of the above cases apply, it adds a new mapping to the substitution, indicating that the variable is unified with the expression.

Overall, the algorithm systematically traverses through the expressions, handling variables, compounds, lists, and checking for failures, until it either finds a successful substitution or determines that unification is not possible.

- Overall, the algorithm systematically traverses through the
    - **expressions,**
    - **handling variables**,
    - **Compound statements**,
    - **lists**, and
    - **checking for failures**,
- until it either finds a ***successful substitution*** or determines that unification is ***not possible***.

Example

Suppose we have the following two predicates:
1. Predicate **P(x,y)**
2. Predicate **Q(f(z),a)**

Here,
- **P** and **Q** are predicates,
- **x, y, and z** are variables, and
- **f and a** are constants.

Now, let's say we want to unify **P(x,y) with Q(f(z),a).**

We can use the given algorithm for unification to find a substitution that makes these two predicates identical.

1. Initially, θ is empty.

2. Start unifying the predicates: P(x,y) and Q(f(z),a)

Since **P** and **Q** are different, they can't be unified directly.

3. Unify the arguments: Unify **x** with **f(z)** and **y** with **a**

4. **Unify x with f(z):**
    - **x** is a variable, **f(z)** is a compound term.
    - Call **UNIFY-VAR(x, f(z), θ):**
        - Add **x/f(z) to θ**
        - **θ={x/f(z)}**

5. **Unify y with a:**
    - **y** is a variable, **a** is a constant.
    - Call UNIFY-VAR(y, a, θ):
        - Add **y/a to θ**
        - **θ={x/f(z),y/a}**

6.Finally, return θ:

θ={x/f(z),y/a}

So, the resulting substitution θ makes P(x,y) and Q(f(z),a) identical:

P(x,y){x/f(z),y/a}=Q(f(z),a)

Storage and retrieval :

- Underlying the **TELL** and **ASK** functions used to inform and interrogate a knowledge base are the more primitive **STORE** and **FETCH** functions. STORE(s) stores a sentence s into the knowledge base and FETCH(q) returns all unifiers such that the query q unifies with some.
- Given a sentence to be stored, it is possible to construct indices for all possible queries that unify with it. 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?**

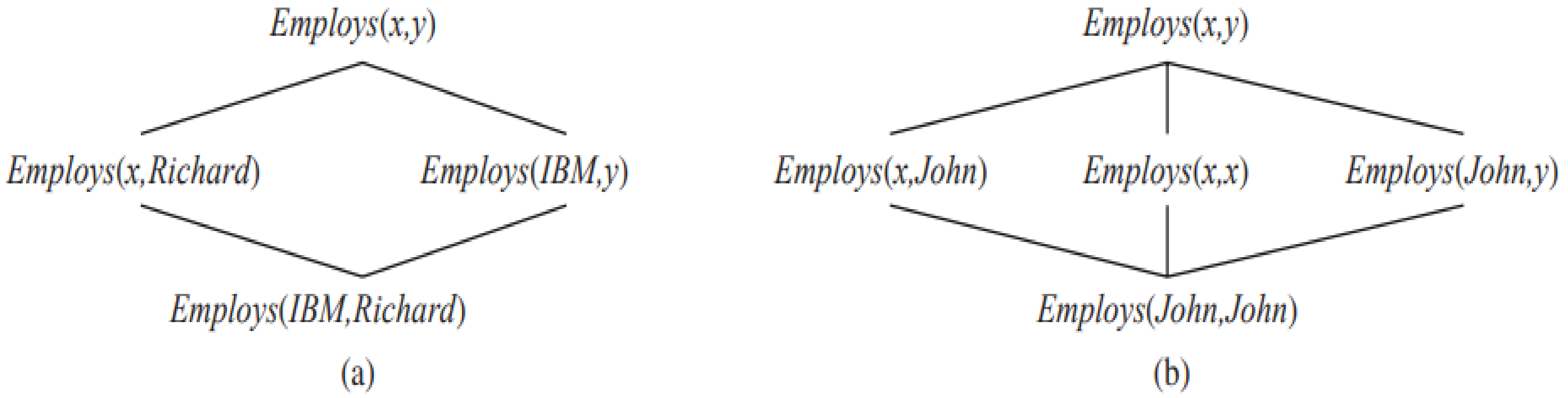

Figure 9.2 (a) The subsumption lattice whose lowest node is $Employs(IBM, Richard)$. (b) The subsumption lattice for the sentence $Employs(John, John)$.

Forward Chaining:

- Forward chaining is a reasoning method , starts with the known facts and uses inference rules to derive new conclusions until the goal is reached or no further inferences can be made.
- In essence, it proceeds forward from the premises to the conclusion.

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.**

- **Forward chaining would proceed as follows:**

1. Check the first rule: **Fever**? **Yes. Proceed**.
2. Check the second rule: **Sore throat**? **Yes. Proceed**.
3. Apply the third rule: The patient has a fever and sore throat, thus they should see a doctor.

Forward chaining is suitable for situations where there is a large amount of known information and the goal is to derive conclusions.

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.
- **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**. The following are first-order definite clauses:
  - King(x) ∧ Greedy(x) ⇒ Evil(x) .
  - King(John) .
  - Greedy(y) .

Forward Chaining,

- Unlike propositional literals, first-order literals can include variables, in which case those variables are assumed to be universally quantified.
- **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.***
- We will prove that **West** is a criminal.

First, we will represent these facts as first-order definite clauses.

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) .
2. "Nono . . . has some missiles."
   - The sentence ∃ x Owns(Nono, x)∧Missile(x) is transformed into two definite clauses by Existential Instantiation, introducing a new constant M1:
     - Owns(Nono, M1)
     - Missile (M1)
3. "All of its missiles were sold to it by Colonel West":
   - Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono) .
4. We will also need to know that missiles are weapons:
   - Missile(x) ⇒ Weapon(x)
5. and we must know that an enemy of America counts as "hostile":
   - Enemy(x, America) ⇒ Hostile(x) .
6. "West, who is American . . .":
   - American(West) .
7. "The country Nono, an enemy of America . . .":
   - Enemy(Nono, America) .

From these inferred facts, we can conclude that Colonel West is indeed a criminal since he sold missiles to a hostile nation, which is Nono.

“. . . it is a crime for an American to sell weapons to hostile nations”:

- **American(West) ∧ Weapon(Missile) ∧ Sells(West, Missile, Nono) ∧ Hostile(Nono) ⇒ Criminal(West) .**

DATALOG :

- This knowledge base contains no function symbols and is therefore an instance of the class of **Datalog** knowledge bases.
- **Datalog** is a language that is restricted to first-order definite clauses with no function symbols.
- **Datalog** gets its name because it can represent the type of statements typically made in relational databases.

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
    $new \leftarrow \{\}$
    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 *fail* **then return** ϕ
    add new to KB
  return *false*

Figure 9.3 A conceptually straightforward, but very 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.

Explanation of Algorithm

- This algorithm is an implementation of Forward Chaining with a **goal-directed query mechanism, specifically designed for First-Order Logic (FOL) knowledge bases**.
- It's called Forward Chaining with Ask (FOL-FC-ASK). Let's break down the steps:

Algorithm:

1. **Inputs:**
   - **KB**: The knowledge base, which consists of a set of first-order definite clauses.
   - **α:** The query, which is an atomic sentence.
2. **Loop until no new sentences are inferred:**
   - Initialize **new** as an empty set.
3. **Iterate through each rule in the knowledge base**:
   - Standardize the variables in the rule (**ensuring variable names are unique**).
   - For each **substitution θ** that makes the antecedent of the rule (\`p1 ∧ ... ∧ pn\`) match some subset of the KB:
     1. Apply the substitution to the consequent of the rule (\`q\`) to generate a new sentence **\`q'\`**.
     2. Check if **\`q'\`** unifies with some sentence already in the KB or \`new\`. If not, add \`q'\` to \`new\`.
     3. Attempt to unify **\`q'\`** with the query **\`α\`**. If unification succeeds (resulting in a substitution ϕ), return ϕ.
     4. **Update the knowledge base**:
        - Add the sentences in \`new\` to the KB
     5. **Repeat the loop** until no new sentences are inferred or until the query is proven or disproven.
4. **Output**:

- If the query is proven, return the substitution that makes it true.
- If the query is disproven (i.e., it cannot be proven true), return false.

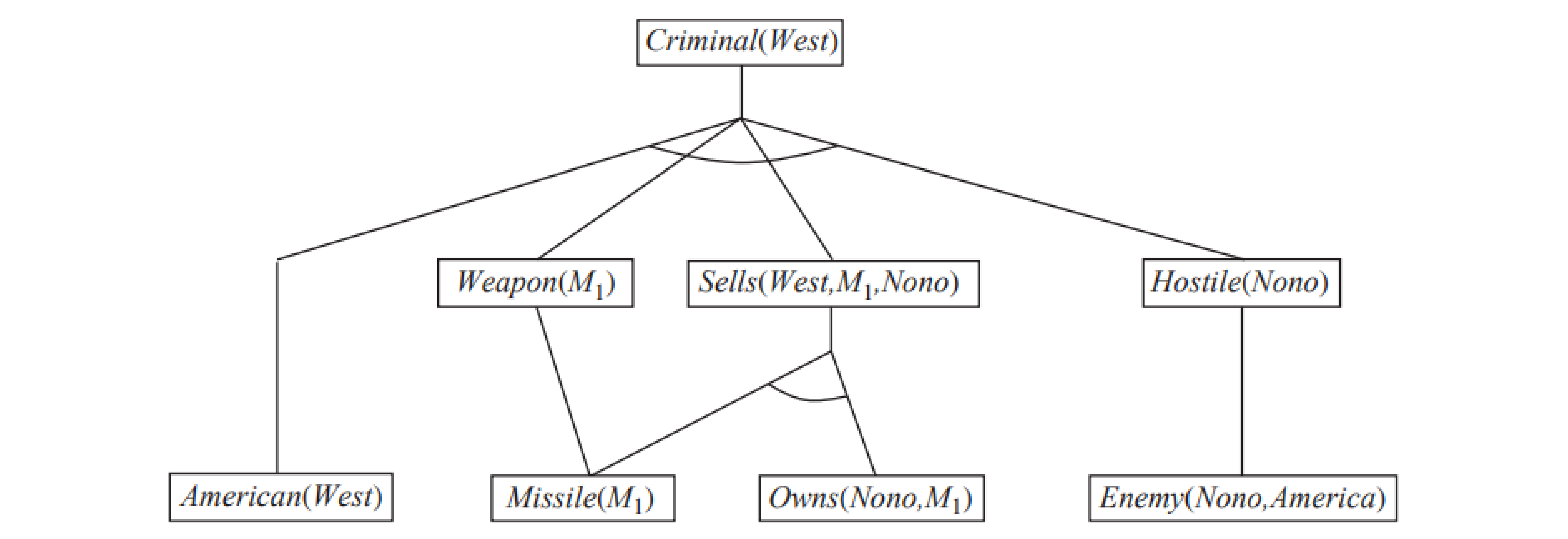

Figure 9.4 The proof tree generated by forward chaining on the crime example. The initial facts appear at the bottom level, facts inferred on the first iteration in the middle level, and facts inferred on the second iteration at the top level.

Backward 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 is useful when there is a specific goal to be achieved, and the system can efficiently backtrack through the inference rules to determine whether the goal can be satisfied.

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.

Explanation of Algorithm

This algorithm represents Backward Chaining, a goal-driven reasoning method used in automated theorem proving and reasoning systems for First-Order Logic (FOL). Let's break down the steps:

Input:

KB: The knowledge base, consisting of a set of first-order definite clauses.

query: The query for which we want to find solutions..

Algorithm:

FOL-BC-ASK:

- This function initiates the backward chaining process by calling **FOL-BC-OR** with an empty substitution θ.

FOL-BC-OR:

- This generator function yields substitutions that satisfy the goal by applying rules from the knowledge base.
- It iterates over each rule in the knowledge base that matches the goal.
- It standardizes the variables in the rule to avoid variable name conflicts.
- For each possible substitution **θ'**, it calls **FOL-BC-AND** to handle the antecedent of the rule.

FOL-BC-AND:

- This generator function yields substitutions that satisfy a list of goals.
- If **θ** is a failure, it returns.
- If there are no goals left, it yields the current substitution **θ**.
- Otherwise, it recursively processes each goal in the list:
  - It retrieves the first goal and the rest of the goals.
  - For each possible substitution **θ'** generated by processing the first goal, it recursively calls FOL-BC-OR to handle the rest of the goals.

Output: The output is a generator that yields substitutions satisfying the query.

- **Backward chaining starts with the goal (the query)** and recursively decomposes it into subgoals until it reaches atomic sentences or predicates. It then searches the knowledge base for rules that can prove these subgoals. If a rule's consequent unifies with a subgoal, it recursively tries to satisfy the antecedent of the rule by decomposing it into further subgoals. This process continues until either the query is satisfied or no further rules can be applied.
- **The algorithm uses substitution to maintain the bindings of variables** as it traverses through the goals and rules. It applies unification to match the goal with the rule's consequent, ensuring compatibility.
- **Overall, Backward Chaining is an effective method** for reasoning backward from the goal to the known facts in the knowledge base, thereby determining whether the query can be satisfied and generating possible solutions in the form of substitutions.

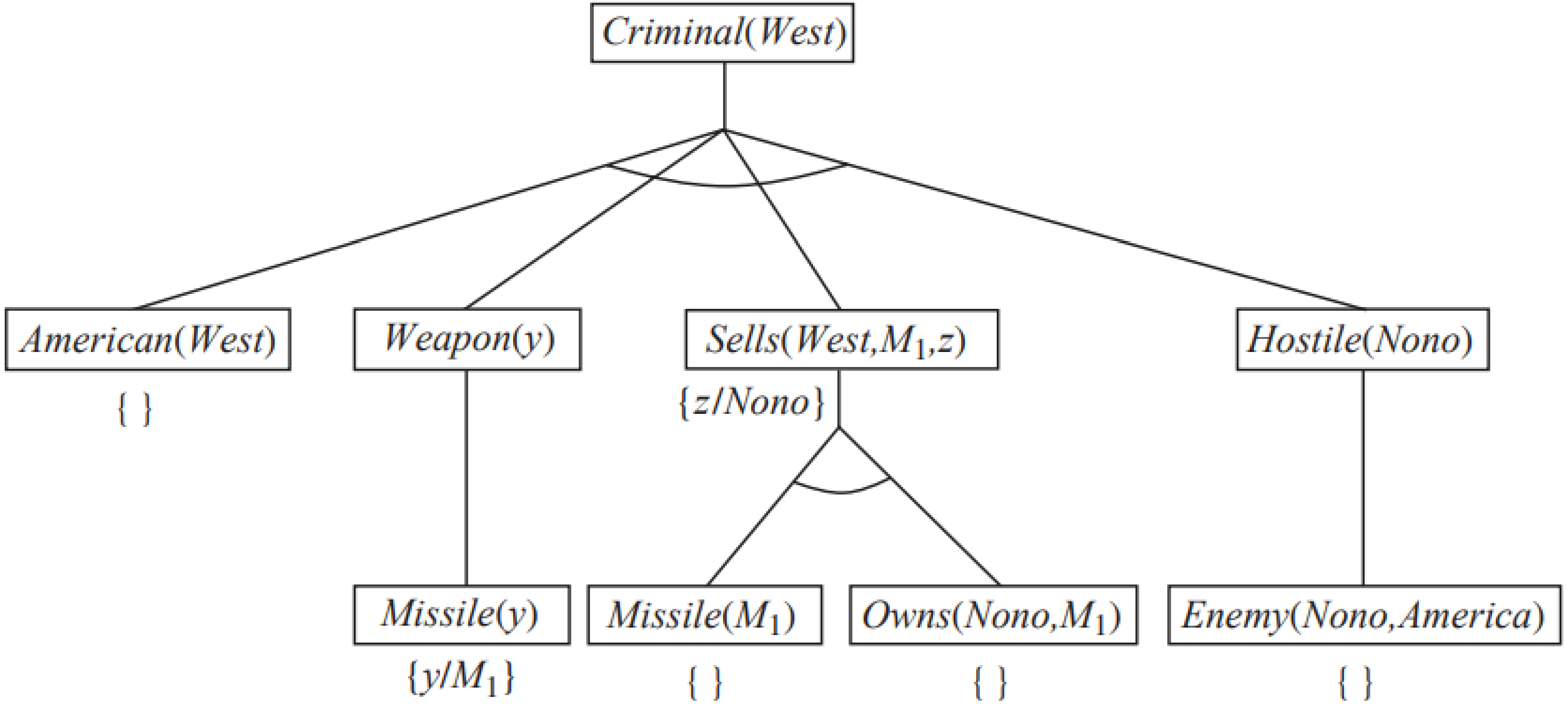

Figure 9.7 Proof tree constructed by backward chaining to prove that West is a criminal. The tree should be read depth first, left to right. To prove *Criminal*(*West*), we have to prove the four conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining.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

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.
- 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:

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.

Resolution

- Conjunctive normal form for first-order logic : As in the propositional case, 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.
- Literals can contain variables, which are assumed to be universally quantified. For example, the 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)** .

Resolution

- Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence.
- The procedure for conversion to CNF is similar to the propositional case, The principal difference arises from the need to eliminate existential quantifiers.
- **We illustrate the procedure by translating the sentence**
- "Everyone who loves all animals is loved by someone," or
- **$\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y\ Loves(y, x)]$** .

Steps

- **Eliminate implications**: $\forall$ x [¬$\forall$ y ¬Animal(y) $\lor$ Loves(x, y)] $\lor$ [$\exists$ y Loves(y, x)] .
- **Move ¬ inwards**: In addition to the usual rules for negated connectives, we need rules for negated quantifiers. Thus, we have
  - ¬$\forall$ x p becomes $\exists$ x ¬p
  - ¬$\exists$ x p becomes $\forall$ x ¬p .
- **Our sentence goes through the following transformations**:
  - $\forall$ x [$\exists$ y ¬(¬Animal(y) $\lor$ Loves(x, y))] $\lor$ [$\exists$ y Loves(y, x)] .
  - $\forall$ x [$\exists$ y ¬¬Animal(y) $\land$ ¬Loves(x, y)] $\lor$ [$\exists$ y Loves(y, x)] .
  - $\forall$ x [$\exists$ y Animal(y) $\land$ ¬Loves(x, y)] $\lor$ [$\exists$ y Loves(y, x)] .
- **Standardize variables**: For sentences like ($\exists$ x P(x))$\lor$($\exists$ 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
  - $\forall$ x [$\exists$ y Animal(y) $\land$ ¬Loves(x, y)] $\lor$ [$\exists$ z Loves(z, x)] .

- **Skolemize**: Skolemization is the process of removing existential quantifiers by elimination. Translate $\exists x\, P(x)$ into $P(A)$, where A is a new constant.
  - **Example :**
    - $\forall x\ [Animal(A) \wedge \neg Loves(x, A)] \vee Loves(B, x)$,
    - $\forall x\ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(z), x)$. 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:
  - $[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(z), x)$.
- **Distribute ∨ over ∧:**

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

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.
- Thus We have

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

where $\textsc{Unify}(\ell_i, \neg m_j) = \theta$. 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

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

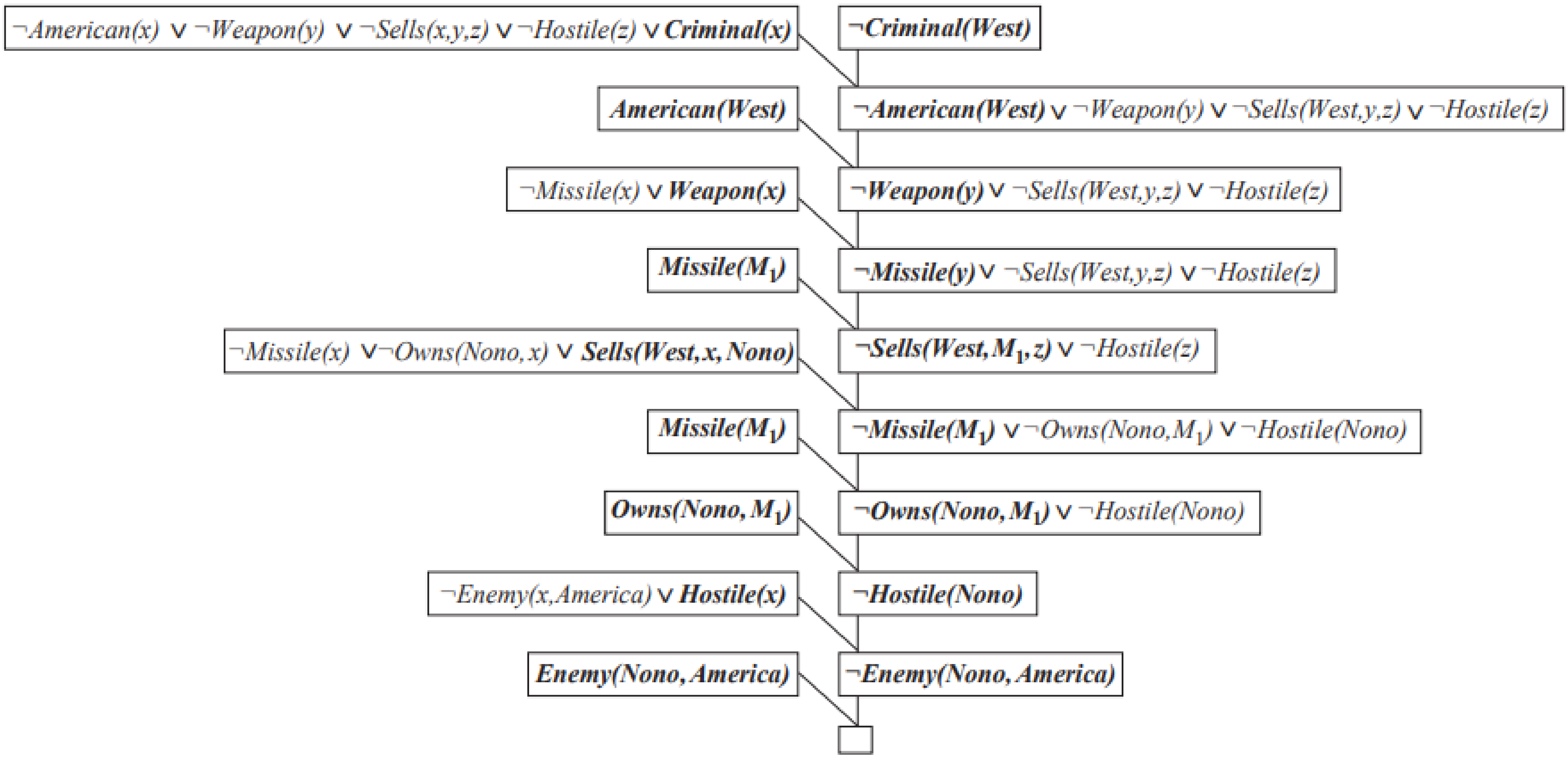

Figure 9.11 A resolution proof that West is a criminal. At each step, the literals that unify are in bold.

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

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

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

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. $Animal(F(x)) \lor Loves(G(x),x)$

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

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

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

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

E. $Cat(Tuna)$

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

¬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.

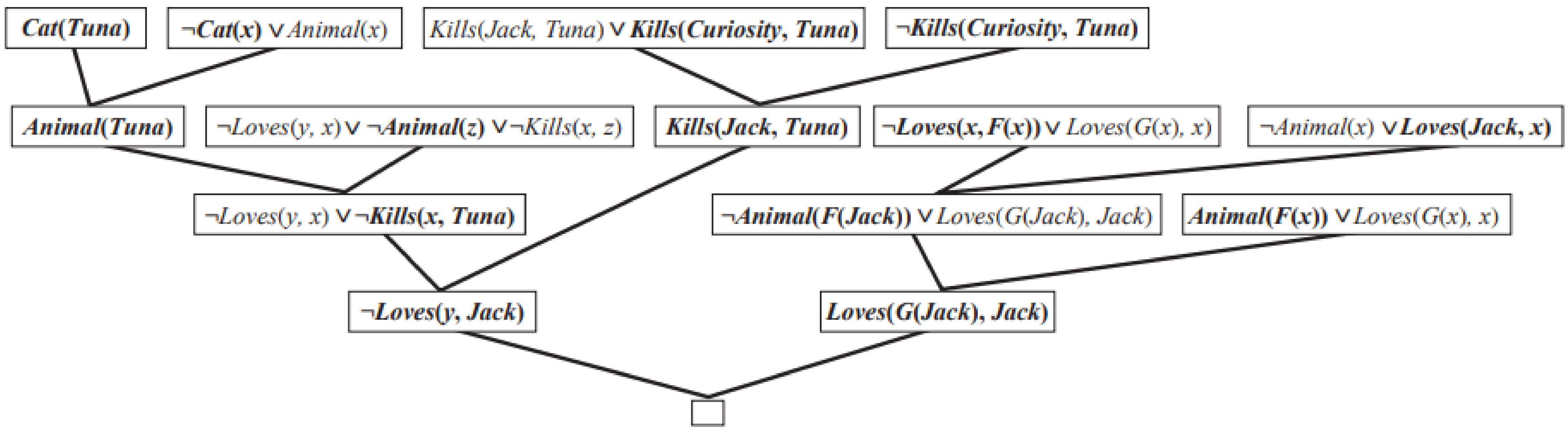

Figure 9.12 A resolution proof that Curiosity killed the cat. Notice the use of factoring in the derivation of the clause $Loves(G(Jack), Jack)$. Notice also in the upper right, the unification of $Loves(x, F(x))$ and $Loves(Jack, x)$ can only succeed after the variables have been standardized apart.

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***.